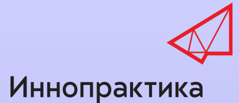
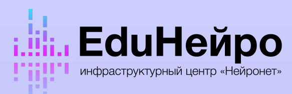
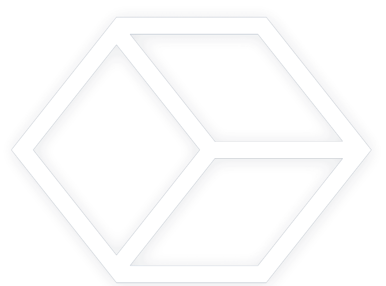




ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ В РОБОТОТЕХНИКЕ. РОБОФУТБОЛ

Данное методическое пособие было создано при поддержке негосударственного института развития «Иннопрактика» в целях работ по созданию методических материалов (методические пособия и видеолекции) в рамках программы инфраструктурного центра «Нейронет»



Оглавление

0.1 От авторов 7

0.2 Предисловие 8

0.3 Образовательная программа 9

1 Введение в робототехнику 17

1.1 Лекция 17

1.1.1 Автономные гуманоидные роботы 17

1.1.2 RoboCup 17

1.1.3 FIRA 18

1.1.4 Роботы 18

1.1.5 Цикл создания автономного робота-гуманоида 20

1.2 Семинар 24

1.2.1 Мир 24

1.2.2 Робот 25

1.2.3 Контроллер 26

1.3 Практическое занятие 27

2 Классическое зрение 28

2.1 Лекция 28

2.1.1 Особенности канала зрения 28

2.1.2 Постановка задачи детекции 29

2.1.3 Представление изображения 30

2.1.4 Цветовое пространство RGB . . . 30
2.1.5 Цветовое пространство HSV . . . 31
2.1.6 Компоненты связности . . . 34
2.1.7 Вопросы после лекции . . . 36

2.2 Семинар 39

2.2.1 Загрузка изображений с диска . . . 39

2.3 Распознавание паттернов 40

2.3.1 Фильтры . . . 42
2.3.2 МНК . . . 42
2.3.3 Преобразование Хафа . . . 43
2.3.4 RANSAC . . . 44
2.3.5 TinyYOLO . . . 45

2.4 Стерео 46

2.4.1 Цифровая камера . . . 46
2.4.2 Модель камеры . . . 48
2.4.3 Стерео . . . 49
2.4.4 Стереокамеры . . . 49
2.4.5 Работа с видео . . . 51
2.4.6 Распределения в каналах . . . 52
2.4.7 Сдвиг распределения в одном из каналов . . . 55
2.4.8 Построение маски объекта . . . 57
2.4.9 Морфологические операции . . . 58
2.4.10 Обработка связных компонент . . . 60
2.4.11 Работа с контурами . . . 63
2.4.12 Вычитание фона . . . 65

3 Нейронные сети . . . 68

3.1 Лекция 68

3.1.1 Задача обучения . . . 68
3.1.2 История . . . 69
3.1.3 Функции активации нейрона . . . 74
3.1.4 Связи нейронов . . . 78
3.1.5 Слои. Выходы . . . 79
3.1.6 Теорема Цыбенко . . . 79
3.1.7 Набор данных . . . 80
3.1.8 Функция потерь . . . 80
3.1.9 Обратное распространение ошибки . . . 81
3.1.10 Оптимизатор . . . 81
3.1.11 Эпоха . . . 82
3.1.12 Learning rate . . . 82

3.1.13 Цикл обучения . . . 82

3.2 Семинар **83**

3.2.1 Работа с тензорами в torch . . . 83
3.2.2 Реализация модели нейрона . . . 85
3.2.3 Ручное обучение нейрона . . . 88
3.2.4 Обучение нейрона с байесом . . . 91
3.2.5 Задача «XOR» . . . 96
3.2.6 Задача «Домик» . . . 99

3.3 Практическое занятие **107**

4 Классификация изображений . . . 108

4.1 Лекция **108**

4.1.1 Проблема анализа изображений. . . . 108
4.1.2 Сверточный слой . . . 108
4.1.3 Батч. Нормализация батча . . . 111
4.1.4 Метрики для классификации . . . 112
4.1.5 Метрики для изображений . . . 112
4.1.6 Обучение. Валидация. Тест . . . 113
4.1.7 Confusion matrix . . . 114
4.1.8 Переобучение . . . 114

4.2 Семинар **114**

4.2.1 Работа с датасетами в PyTorch . . . 115
4.2.2 Реализация нейросетевого классификатора изображений . . . 119
4.2.3 Реализация классификатора изображений на сверточных сетях . . . 125

4.3 Практическое занятие **128**

5 Детекция объектов . . . 129

5.1 Лекция **129**

5.1.1 О задаче детекции . . . 129
5.1.2 Наборы данных . . . 129
5.1.3 Метрики качества . . . 129
5.1.4 Методы решения . . . 131
5.1.5 Архитектуры нейронных сетей . . . 132
5.1.6 Single Shot MultiBox Detector (SSD) . . . 135
5.1.7 You Only Look Once (YOLO) . . . 136
5.1.8 Дополнительные виды архитектур . . . 136

5.2 Семинар **137**

5.2.1 Подготовка датасета . . . 137
5.2.2 Обучение YOLOv5 . . . 139

5.2.3 Подготовка встраиваемого решения и проверка результатов 141

5.3 Практическая работа 141

6 Локализация 151

6.1 Локализация 151

6.1.1 Задача локализации 151
6.1.2 Триангуляция 152
6.1.3 Одометрия 153
6.1.4 Комбинированный способ 153
6.1.5 Фильтр частиц 154

6.2 Семинар 156

6.2.1 Практическая работа 157

6.3 Практическое занятие 158

6.4 SLAM 162

6.4.1 Визуальная одометрия 162
6.4.2 Дескриптор особенностей 165
6.4.3 SLAM 174

7 Введение в ROS 179

7.1 Лекция 179

7.1.1 Зачем нужен ROS 179
7.1.2 Термины ROS 184
7.1.3 Использование ROS 189

7.2 Семинар 192

7.2.1 Подписчик-издатель 192
7.2.2 Клиент-сервис 196

7.3 Практическое занятие 200

8 Системы контроля версий 202

8.1 Лекция 202

8.1.1 История 202
8.1.2 Зачем и когда нужен гит? 203
8.1.3 Коммиты 204
8.1.4 Ветвление 205
8.1.5 Удачная модель ветвления 206

8.2 Семинар 207
8.2.1 Работа с google collaboratory 207
8.2.2 Работа с гитом 207
8.3 Практическое занятие 208

9 Дополнительные главы 209
9.1 Стабилизация обратного маятника 209
9.2 Симуляция упругих столкновений двумерных объектов 213
9.3 Обратная кинематика для плоского двузвенного манипулятора 218
9.4 Стерео 221
9.5 Визуализация трехмерных объектов 227

0.1 От авторов

Перед вами методическое пособие по робототехнике и искусственному интеллекту, подготовленное на основе курсов, прочитанных авторами в МФТИ и Физтех-лицее. Оно предназначено в первую очередь для изучения в кружках по робототехнике, о чем будет более развернуто сказано ниже, но подходит и для самостоятельного изучения.

Методическое пособие организовано следующим образом. В самом начале приведен рекомендуемый план изучения годового курса по робототехнике и искусственному интеллекту. Для каждого урока перечислены основные темы, которым его на наш взгляд стоит посвятить. Всё содержимое этих уроков изложено в методическом пособии в формате лекций, семинаров (кода с пояснениями), заданий для самостоятельного выполнения и контрольных вопросов.

Вместе с методическими материалами распространяются видео и примеры кода. Видео подразделяются на лекции, которые можно просто смотреть, и семинары. Семинары — это, как правило, или работа с кодом, или работа с программным обеспечением, и их рекомендуется выполнять параллельно с докладчиком.

Первая глава пособия представляет собой введение в предметную область, в которой рассказывается о том, что такое автонмные гуманоидные роботы, из каких основных модулей они состоят, и какие есть робототехнические соревнования.

За этим следуют несколько разделов, посвященных уже более предметному знакомству с конкретными подсистемами автономного робота. Первый из них — это классическое компьютерное зрение. В следующих нескольких главах рассказывается о нейронных сетях: от истории их развития и самых простых моделей до обучения современных детекторов. После этого изложены локализация и стратегия. Следующе после них две главы посвящены важным для робототехника фреймворкам — *ROS* и *git*. Наконец, последняя глава пособия — это собрание разных подходов, методов; и просто важных вещей из робототехники и смежных областей, в частности визуализации и простейших физических симуляторов.

Мы понимаем, что пособие такого объема не может обойтись без опечаток и неточностей. Сообщить о них, да и просто поделиться обратной связью по материалам можно через почту starkit.edu@gmail.com или в Телеграме @elijahmipt. Также наша лаборатория ведет канал на YouTube — StarkitRobots.

Авторы выражают благодарность Азеру Бабаеву, Егору Давыденко, а также коллективу Лаборатории Волновых Процессов и Систем Управления МФТИ и стажерам команды «Старкит» за помощь в подготовке этого пособия.

0.2 Предисловие

Хотя роботы в их современном виде появились только во второй половине XX века, мифы о самостоятельно движущихся механизмах, таких как Голем, существовали задолго до этого. Мечты о механических работниках, которые не устают и в точности выполняют все задания, так и оставались мечтами вплоть до появления первых программируемых механизмов.

Настоящий расцвет робототехники произошел вследствие совокупности факторов, в числе которых развитие интегральных схем, появление эффективных и мощных электродвигателей, создание аккумуляторов высокой емкости, а также развитие теории управления, теоретической механики, вычислительных методов и многих других дисциплин.

Во всем пособии между строк будет читаться отсылка к реальным применениям. Соревнования, которые занимают важное место в рекомендуемой структуре изучения курса, проводятся в симуляции. Этот инструмент позволяет уменьшить порог входа в робототехнику, абстрагироваться от работы со сложной и дорогостоящей механикой и электроникой.

Вся методичка и весь учебный курс в целом посвящены решению задач в применении к гуманоидным роботам. Во-первых, человекоподобная платформа — одна из самых сложных в создании и программировании, а во-вторых, подавляющее большинство подходов, методов и инструментов, изложенных в курсе, безболезненно переносятся на другие платформы.

0.3 Образовательная программа

Эта образовательная программа предназначена для использования преподавателями в кружках по робототехнике и рассчитана на один учебный год. Темы, предлагаемые к изучению, покрывают необходимый минимум знаний для участия в соревнованиях по робототехнике. Соревнованием, в котором ученики разных кружков будут соперничать друг с другом, будет гуманоидный робофутбол. Первый чемпионат проведут в ноябре, второй в мае, и они пройдут в симуляции.

Логичный вопрос, который должен возникнуть, — а почему это вообще нужно ученикам? Где им пригодится робофутбол? Не лучше ли им заняться «настоящей» учебой вместо каких-то игрушек? Краткий ответ: «Нет, не лучше, поскольку робофутбол — это как раз один из лучших способов познакомиться с робототехникой».

Актуальность

Робототехника не наука, а скорее совокупность задач из разных областей программирования, искусственного интеллекта, физики и математики. Это та самая настоящая практика, о которой мечтают и преподаватели, и ученики, и родители. Часто бывает, что в курсе информатики многие важные темы затрагиваются, но остаются без конкретного применения и не закрепляются. Если один ученик решил задачу с квадратичным временем работы, а другой с линейным, это может остаться незамеченным, и положительной обратной связи не возникнет. В случае же немедленного внедрения в робота код первого ученика просто будет работать быстрее, и из подобных вещей в итоге сложится место его команды в турнирной таблице. Примеры можно привести в рамках всех перечисленных дисциплин. Здорово, когда школьники знакомы с производными и интегрированием. Но при этом гораздо более ценно, когда они понимают, как именно ходьба гуманоидного робота связана с задачей стабилизации обратного маятника, как это просимулировать, и как должны быть устроены ноги робота, чтобы их момент инерции был невелик.

Задача робофутбола была выбрана лигой RoboCup как один из тех видов соревнований, в которых могут быть применены очень многие наработки из области робототехники. В их числе командная игра, ходьба по неровной поверхности, компьютерное зрение в меняющихся условиях, нахождение собственных координат. Команда Старкит МФТИ выиграла чемпионат мира в 2021 году и после этого приступила к созданию курса, который и предлагается к изучению.

Цель

Курс предназначен для того, чтобы познакомить слушателей со всеми основными составными частями робототехнической системы. Этот уровень должен быть достаточен для участия в соревнованиях по робофутболу школьного уровня, поэтому число занятий и глубина погружения в материал потребуют и от учеников, и от преподавателей большой отдачи. После успешного прохождения курса ученики овладеют теоретическими, математическими и программными методами, а также инструментами разработки и проектирования, достаточными для дальнейшего изучения робототехники на серьезном уровне.

Задачи

- **Развитие навыков** в области программирования, робототехники и искусственного интеллекта. Работа со сложными системами естественным образом потребует от учеников овладения теоретической и практической базой в предметной области
- **Получение опыта командной работы** на примере соревнований с другими командами близкого уровня
- **Стимулирование самостоятельной работы** для достижения поставленной задачи. В рамках разработки сложного робототехнического комплекса ученики будут многие сотни раз обращаться к документации, тематическим форумам, лекциям, видеоурокам

Сроки реализации

Вся программа разделена на девять месячных блоков. Два из них — ноябрьский и майский — посвящены соревнованиям. В сентябре и октябре ученики знакомятся с предметной областью, с составными частями ПО робота. Декабрь будет посвящен изучению методов локализации. В январе на занятиях будут изучены библиотеки и инструменты разработки и тестирования, а именно ROS и вещи, с которыми он может работать в связке. Курс в целом достаточно нагружен программными средствами и фреймворками, ученики познакомятся с git, OpenCV, PyTorch, Webots, ROS, OpenAI Gym. Опционально можно рассказать им и про LaTeX.

В течение учебного года вы можете отходить от программы, насколько считаете правильным, углубляясь в те или иные отрасли математики, физики, программирования или чего угодно еще, что вы считаете полезным. Главное, чтобы это помогало ученикам в написании кода для успешного участия в соревнованиях. Методические материалы, лекции и код призваны быть фундаментом, от которого вы можете отталкиваться в своем рассказе.

Форма занятий

Рекомендуемое число занятий — две пары в неделю, то есть четыре урока (академических часа), разделенных на две равные части. Такая продолжительность и частота позволят поддерживать погруженность в контекст, которую было бы трудно удержать с одним занятием в неделю или меньшей длительностью.

Типичный размер команды, при котором самопроизвольно возникает и поддерживается разумное распределение обязаннностей, роли в команде и эффективная коммуникация — это три-шесть человек. Если вы набрали в кружок десять учеников, можно предложить им разделиться на две команды, которые будут параллельно решать одни и те же задачи, а потом делиться опытом друг с другом.

С кружками часто бывает, что сначала школьников приходит много, они оптимистично настроены и заинтересованы, а потом интерес немного угасает, и их число естественным образом снижается. Нужно иметь это обстоятельство в виду и дать широкую огласку набору в кружки. Можно написать пост в школьный паблик, дать объявление в чате родителей, скинуть видео с игрой роботов. Поскольку робототехника на данный момент не изучается так же системно, как математика или физика, попадание в кружок — это

зачастую воля случая, и надо дать шанс попробовать и заинтересоваться как можно большему количеству школьников.

В меру сил имеет смысл поддерживать активное общение с учениками в чате, проводить дополнительные консультации и увеличивать продолжительность занятий, если в этом возникает необходимость. Практика показывает, что увлеченная аудитория способна заниматься робототехникой и программированием гораздо дольше одной пары.

Около раза в месяц будут проводиться общие для всех кружков онлайн-тестирования для проверки усвоения знаний.

Требования к слушателям, возраст учеников

Для быстрого старта в робототехнике и преподаватель, и ученики должны быть знакомы с языком программирования Python. В случае необходимости стоит начать именно с этого, изучив вместе со слушателями условные операторы, циклы, функции и основные структуры данных, такие как массивы и словари.

Жестких требований к возрасту слушателей нет, но разумно начинать изучать робототехнику, уже когда сформированы представления о программировании в целом. Поэтому имеет смысл отталкиваться от программы в конкретной школе, а также от уровня ученика. Отбор в кружок не должен быть слишком строгим, лучше дать человеку попробовать, чтобы не пропустить заинтересованного, который догонит общий уровень подготовки.

Погружение в предметную область предполагается постепенным: в начале слушатели в обзорном формате познакомятся с основными составными частями программного обеспечения робота, а за этим последует более детальное и формализованное изложение материала.

Для синхронизации в начале некоторых блоков предусмотрены лекции от сотрудников команды Старкит. Можно смотреть их отдельно, а потом формировать с их помощью свой рассказ, можно смотреть их с учениками. Цель этих лекций — с некоторой периодичностью рассказывать об устройстве той или иной области внутри робототехники с высоты птичьего полета, то есть это лекции скорее обзорные, чем технические.

Поскольку и дисциплина достаточно молодая, и кружки только будут открываться, вся эта затея потребует большого объема самостоятельной работы, подготовки, обучения. Крайне приветствуются вопросы любого уровня сложности, общение с коллегами и обмен опытом через форум.

Почитать о лиге ELSIROS и поставить на компьютер комплект программного обеспечения можно здесь. Все вопросы, касающиеся программирования роботов, предлагается обсуждать на специальном форуме. Конечно, общение в мессенджерах привычнее и быстрее, но использование форума позволит накопить базу часто задаваемых вопросов, для которых в соответствующих ветках уже будут ответы.

Введение[сентябрь]

1. **Знакомство с областью, современная робототехника [лекция от Старкит]**
   История робототехники, развитие шагающих механизмов, типы актуаторов, иерархия управления, составные части робототехнической системы, пионерские разработки в области, перспективы, проблемы и задачи современной робототехники
2. **Webots/ELSIROS**
   Робофутбол, основные задачи, установка программного обеспечения, настройка среды, запуск игры со стандартным кодом
3. **Настройка среды**
   Установка интерпретатора Python, установка Jupyter Notebook, получение доступа к камере, операции с массивами, Google Colaboratory
4. **Введение в классическое зрение, часть 1**
   Представление цифровых изображений, цветовые пространства, детектирование объектов по цвету, маски
5. **Введение в классическое зрение, часть 2**
   Обработка масок, морфологические операции, свертки, связные компоненты
6. **Слежение за мячом, часть 1**
   Получение кадров с камеры в симуляции, создание и настройка детектора мяча, управление сервомоторами в симуляции
7. **Слежение за мячом, часть 2**
   Среда и агент, обратная связь, цикл управления. Создание программы, следящей за движениями мяча на поле
8. **Системы контроля версий**
   Предпосылки к появлению систем контроля версий, история систем контроля версий, git, операции с git, полезные практики при работе с git

ELSIROS[октябрь]

1. **Структура проекта ELSIROS**
   Структура файлов и папок, основные модули, транспорт данных, допущения и условности симуляции
2. **Транспорт данных в ELSIROS**
   Обмен данными с помощью контроллера player
3. **Создание движений**
   Создание простого движения и его запуск, Pose designer
4. **Сенсоры и актуаторы**
   Считывание положений сервомоторов, управление сервомоторами, считывание показаний IMU, состояние падения
5. **Стратегия**
   Формализация игровых ситуаций, конечные автоматы, обратная связь, создание стратегии робота на простом примере
6. **Данные о мире**

Получение данных о положении мяча и робота из контроллера player, встраивание в существующий код

7. **Состояние игры**
   Получение данных о состоянии игры из контроллера player, встраивание в существующий код
8. **Стратегия**
   Работа со strategy designer (выбор направления удара), алгоритм работы полевого игрока и вратаря, модификация стратегии, запуск тестовых игр

Первые соревнования[ноябрь]

1. **Спортивная робототехника, опыт соревнований [лекция от Старкит]**
   Видеозаписи с выступлений, рассказ о различных видах спорта, роботах и занимательных случаях
2. **Планирование**
   Распределение задач, зон ответственности, создание плана работ
3. **Работа над кодом**
   Проверка осуществимости желаемых модификаций
4. **Работа над кодом**
5. **Работа над кодом**
6. **Предварительное тестирование**
   Запуск тестовых игр, выяснение причин нежелательного поведения робота, доработка кода
7. **Финальное тестирование, загрузка кода**
   Запуск тестовых игр, фиксация конфигурации кода, сборка архива, загрузка для участия в соревнованиях
8. **Подведение итогов**
   Рассказ от победившей команды, подведение итогов внутри команд

Локализация[декабрь]

1. **Простые методы локализации**
   Постановка задачи локализации, природа шума в измерениях, обзор существующих методов, триангуляция, альфа-бета фильтр
2. **Триангуляция**
   Реализация локализации с помощью триангуляции
3. **Фильтр частиц, часть 1**
   Предпосылки к разработке фильтра частиц, обзор метода, параметрическое пространство, генерация гипотез
4. **Фильтр частиц, часть 2**
   Колесо отсева, построение итоговой гипотезы, сходимость метода
5. **Работа с фильтром частиц**
   Визуализация фильтра частиц, настройка параметров, особенности работы

6. **Реализация фильтра частиц**
7. **Ориентирование в трехмерном пространстве**
   Практика с ARUCO метками, измерение расстояний, получение положения камеры относительно метки

Фреймворки[январь]

1. **Установка ROS**
2. **Введение в ROS**
   Топики, ноды, Publisher, Subscriber
3. **Практика с ROS**
   Сборка пакетов, простой пример из Publisher и Subscriber
4. **Визуализация в ROS**
   Работа с инструментами rviz и rqt
5. **ROS и Webots**
   Обмен данными, получение изображения с камеры

Классическое компьютерное зрение[февраль]

1. **Алгоритмы**
   Асимптотические сложности, алгоритмы графовых поисков
2. **Модель камеры**
   Pinhole модель, обзор типов камер, стереозрение, облака точек
3. **Фильтры**
   Сглаживание, выделение границ, свертки, фильтр Харриса
4. **Детектирование объектов**
   8 подходов к детектированию мяча
5. **Методы обработки изображений**
   Вид сверху (birdview), контуры, подсчет числа пальцев по веб-камере
6. **Ключевые точки**
   Ключевые точки, дескрипторы, SIFT, практика по сопоставлению
7. **Стереозрение**
   Параллакс, диспаратность, карты глубины, стереосопоставление
8. **Распознавание паттернов**
   Метод наименьших квадратов для нахождения линий, преобразование Хафа, RANSAC

Нейросети[март]

1. **Введение**
   Исторический экскурс, слои, формализация задач, градиентный спуск
2. **Практика в обучении сети**

Цикла обучения, вывод функции ошибки, гиперпараметры, обучение линейного слоя умножению на два

3. **Линейный оператор**
   Доработка цикла обучения, отслеживание сходимости параметров линейного слоя
4. **Домик**
   Задача классификации на датасете Домик
5. **MNIST, часть 1**
   Полносвязная нейронная сеть на MNIST
6. **Работа с данными в Pytorch**
   Аугментация данных, загрузка, разметка реальных изображений, автоматическая разметка в симуляторе
7. **MNIST, часть 2**
   Сверточная нейронная сеть на MNIST, свертки, подсчет числа обучаемых параметров в модели
8. **Примеры и практики из области**
   Готовые модели (yolo, mediapipe), дообучение (fine-tuning) на второй половине MNIST
9. **Сверточная сеть на мячах**
   Загрузка данных, доработка модели, тестирование на видеопотоке

Динамические системы[апрель]

1. **Обратный маятник, часть 1**
   Постановка задачи, стабилизация маятника, визуализация управляющего воздействия
2. **Обратный маятник, часть 2**
   Реализация простой симуляционной среды, добавление задержек, ошибок измерений
3. **Работа со средой**
   Исследование качества стабилизации от параметров системы, варьирование момента инерции, предельного момента, шумов и задержек
4. **Cart-pole**
   Стабилизация системы из каретки и стержня
5. **Симуляция физики**
   Реализация демо с объектами и гравитацией
6. **Визуализация**
   Визуализация трехмерных объектов, триангуляция поверхностей, освещение поверхностей
7. **Динамическая стабилизация кондо***
   Рассмотрение робота как обратного маятника, PID-контроллер

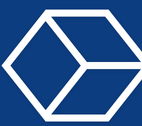
Вторые соревнования[май]

1. **Планирование**

Распределение задач, зон ответственности, создание плана работ

2. **Работа над кодом**
   Проверка осуществимости желаемых модификаций
3. **Работа над кодом**
4. **Работа над кодом**
5. **Предварительное тестирование**
   Запуск тестовых игр, выяснение причин нежелательного поведения робота, доработка кода
6. **Финальное тестирование, загрузка кода**
   Запуск тестовых игр, фиксация конфигурации кода, сборка архива, загрузка для участия в соревнованиях
7. **Подведение итогов**
   Рассказ от победившей команды, подведение итогов внутри команд

1. Введение в робототехнику



1.1 Лекция

1.1.1 Автономные гуманоидные роботы

Сложно представить искусственный интеллект в отрыве от того, чем он будет управлять, — робота. Обратимся к определению термина «робот»

> **Определение**
> Робот — автоматическое устройство, предназначенное для осуществления механических операций, которое действует по заранее заложенной программе.

Но просто роботы — это не так интересно. По своей сути машинка на пульте управления тоже является роботом. Давайте ограничим область робототехники **автономными** роботами. Смысл этого слова в том, что робот должен действовать сам, без или с минимальным участием человека. Выпишем, что должен уметь робот, чтобы мы смогли считать его автономным.

> **Определение**
> Составляющие автономности:
> - Локализация — возможность понимать свое положение в мире
> - Зрение — возможность понимать, что находится в мире
> - Стратегия — возможность принимать правильные решения
> - Питание — возможность работать без подзарядки некоторое время

На данном этапе мы рассматриваем автономных роботов, достаточно широкую и интересную область, включающую в себя наиболее сложную и перспективную часть робототехники — гуманоидных роботов, т.е. роботов, похожих на человека. В этом месте читателю предлагается задуматься о том насколько же все-таки робот может быть похож на человека. В настоящий момент, наука уже позволяет создавать искусственные мышцы, но, к сожалению, не позволяет использовать их в прикладных задачах. Да и если мы научимся делать точную копию человека, это будет уже не робототехника, а клонирование. Поэтому остановимся на следующих ограничениях:

- Совпадающее количество конечностей (две ноги, две руки, одна голова)
- Максимальное количество визуальных датчиков — два глаза (камеры)
- Отсутствие активных датчиков (запрещены сонары, лидары и даже компас)
- Пропорции человека (длина рук, положение центра масс, индекс массы тела и т.д.)

1.1.2 RoboCup

Robocup — международное университетское сообщество, насчитывающее более 500 команд из более чем 50 стран. Соревнования проводятся ежегодно с 1996 года и в данный момент насчитывают 10 major лиг и 3 junior лиги.

- Soccer – Small Size League
- Soccer – Middle Size League

- Soccer – Humanoid
- Soccer – Standard Platform League
- Soccer – Simulation League 2D & 3D
- Industrial – Logistics League
- Industrial – @Work League
- Rescue – Robot League
- Rescue – Simulation League
- @Home League

Самая престижная и конкурентная лига на данный момент Soccer — Humanoid. Она делится на две подлиги — Adult size (роботы высотой от 100 до 180 см) и Kid Size (роботы высотой от 40 до 100 см). А в чем же заключается задача робофутбола? Во-первых, он стремится приблизиться по правилам к обычному футболу, но сейчас существуют некоторые упрощения: поле 6 на 9 метров, всего 4 робота в одной команде, тайм длится 10 минут, а игра включает в себя 2 тайма. Во-вторых, условие победы — забить больше голов, чем команда оппонента. Важно отметить, что роботы полностью автономны и играют без какого-либо участия человека [1]. Робот сам принимает решение куда пинать, куда идти, когда и как вставать и т.д. Основная идеологическая цель лиги звучит так:

> *К середине 21-го века команда полностью автономных человекоподобных роботов-футболистов должна выиграть футбольный матч, соответствующий официальным правилам ФИФА, против победителя самого последнего чемпионата мира.*

Важно заметить, что обе эти лиги хоть и являются соревновательными, но направлены на сотрудничество команд-энтузиастов со всего мира с целью развития робототехники. А теперь давайте посмотрим, в каком состоянии сейчас это все находится. В настоящий момент может показаться, что это просто невозможно, но робототехника быстро развивается и наша команда верит, что в 2050 году роботы смогут играть с человеком на равных.

1.1.3 FIRA

Кроме Robocup существует еще одна не менее важная для робототехнического сообщества лига - FIRA. Цель лиги FIRA Sports — робот, который сможет выступать сразу во многих видах спорта. Для участия в этой лиге требуется всего один робот, что понижает порог входа. Роботу предстоит выступить в более десяти различных видах спортивных соревнований, включающих в себя стрельбу из лука, прыжки в длину, баскетбол, футбол и т.д.

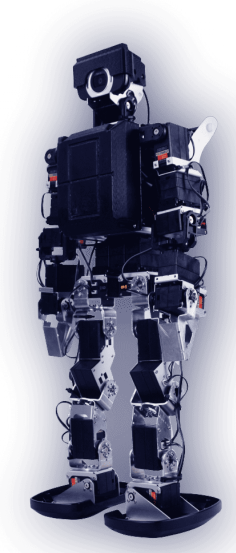
1.1.4 Роботы

Давайте рассмотрим роботов, которые имеются в лаборатории и частично разработаны нашей командой.

[1]Записи игр можно посмотреть на нашем YouTube канале **StarkitRobots**

- Имя: **SAHR**
- Бортовой ПК: Intel NUC
  - Intel Core i5 x64 1.2 GHz (4 ядра)
  - 8GB RAM
  - SSD 256Gb
- Камера: 2шт Point Grey BlackflyS 1440x960 50fps
- Сервомоторы Dynamixel:
  - 10 шт. MX-64
  - 10 шт. MX-106
- Корпус: фрезерованный алюминий и карбон, печатный пластик
- Рост: 73 см, вес: 7.1 кг

- Имя: **Robokit Atom**
- Бортовой ПК: LattePanda
  - Intel Cherry Trail x64 1.92 GHz (2 ядра)
  - 4GB RAM
  - SSD 64Gb
- Камера: Logitech HD Webcam C615 1280x720 30 fps
- Сервомоторы Kondo:
  - 10 шт. KRS-2572HV
  - 16 шт. KRS-2552RHV
- Корпус: фрезерованный алюминий, печатный пластик
- Рост: 50 см, вес: 2.1 кг



- Имя: **Robokit**
- Бортовой ПК: STM32F765VI
  - ARM Cortex M7 216 MHz
  - 512KB RAM
  - 2 MB flash
- Камера: OpenMV Smart Camera 320x240 75 fps
- Сервомоторы Kondo:
  - 6 шт. KRS-2572HV
  - 20 шт. KRS-2552RHV
- Корпус: фрезерованный алюминий, печатный пластик
- Рост: 45 см, вес: 1.8 кг

1.1.5 Цикл создания автономного робота-гуманоида

Теперь, рассмотрев примеры роботов, мы готовы перейти к обсуждению их программного обеспечения. Нам требуется дать роботу возможность действовать в изменяющейся среде полностью автономно. То есть, в первую очередь, необходимо получать информацию об изменениях в среде. В этом нам помогают сенсоры — камеры, датчики давления, микрофоны и т.д. Получив и обработав информацию с сенсоров (модуль **Vision**), а также рассчитав расстояние от робота до найденных объектов (модуль **Model**), мы готовы переходить к следующему этапу — получению информации о местоположении робота и других объектов в среде. За это отвечает модуль локализации. Далее, зная свое местоположение и местоположение объектов, роботу требуется выбрать, что делать дальше (модуль **Stratagy**). Затем выбранное действие передается в модуль **Motion**, а он в свою очередь генерирует и выставляет необходимые положения сервомоторов. Таким образом нам удалось декомпозировать такое сложное устройство как автономный робот всего на 5 смысловых модулей. Конечно, на практике, модули обрастают многочисленными и запутанными связями, но для простого автономного робота схема из таких 5 последовательных модулей показывает себя вполне рабочей.

Теперь давайте рассмотрим каждый модуль в отдельности, но сейчас мы не будем подробно останавливаться на них. Наша цель — понять их суть:

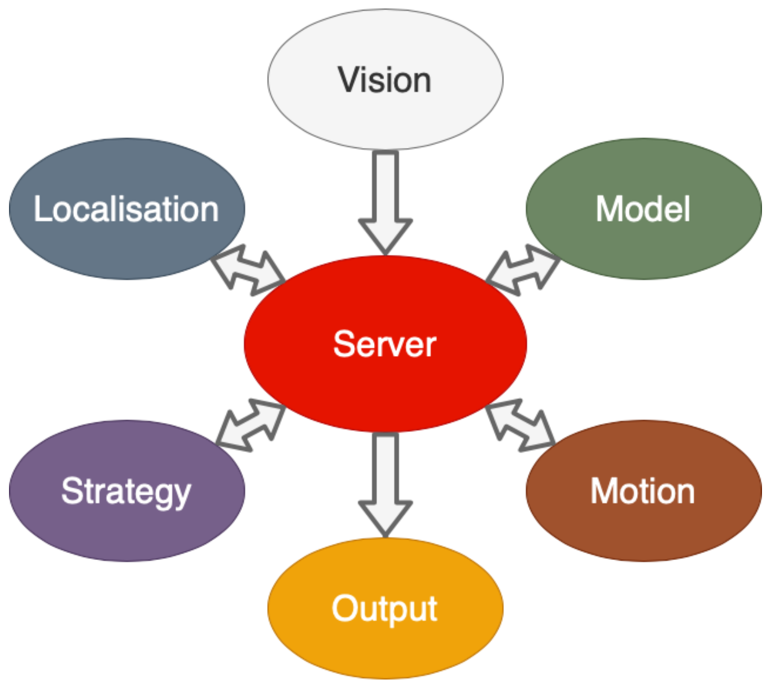


Рис. 1.1: Цикл создания автономного робота-гуманоида

Зрение

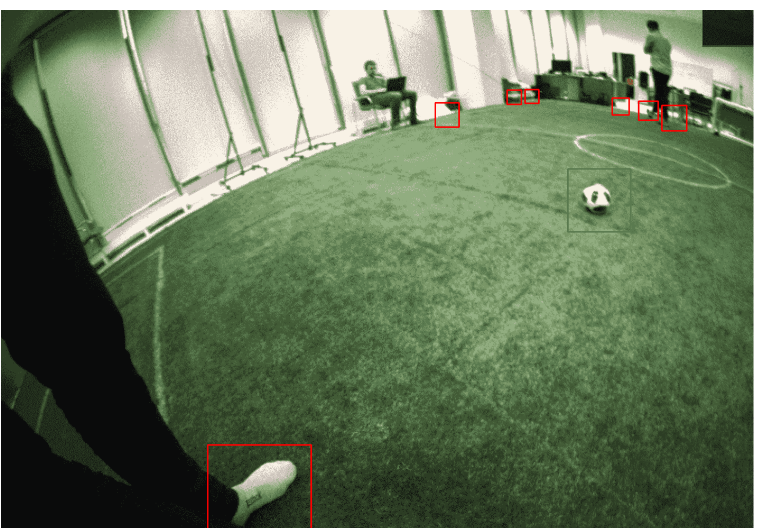


Рис. 1.2: Пример работы детектора мяча.

На рис. 1.2 вы можете видеть пример работы. Красными квадратами (bounding boxes) выделены гипотезы о местоположении мяча, а зеленым — сам мяч. Как можно видеть, система компьютерного зрения считает белый носок очень похожим на мяч. Это долгое время было для нас проблемой, но сейчас мы научили нейронную сеть различать их.

Модель

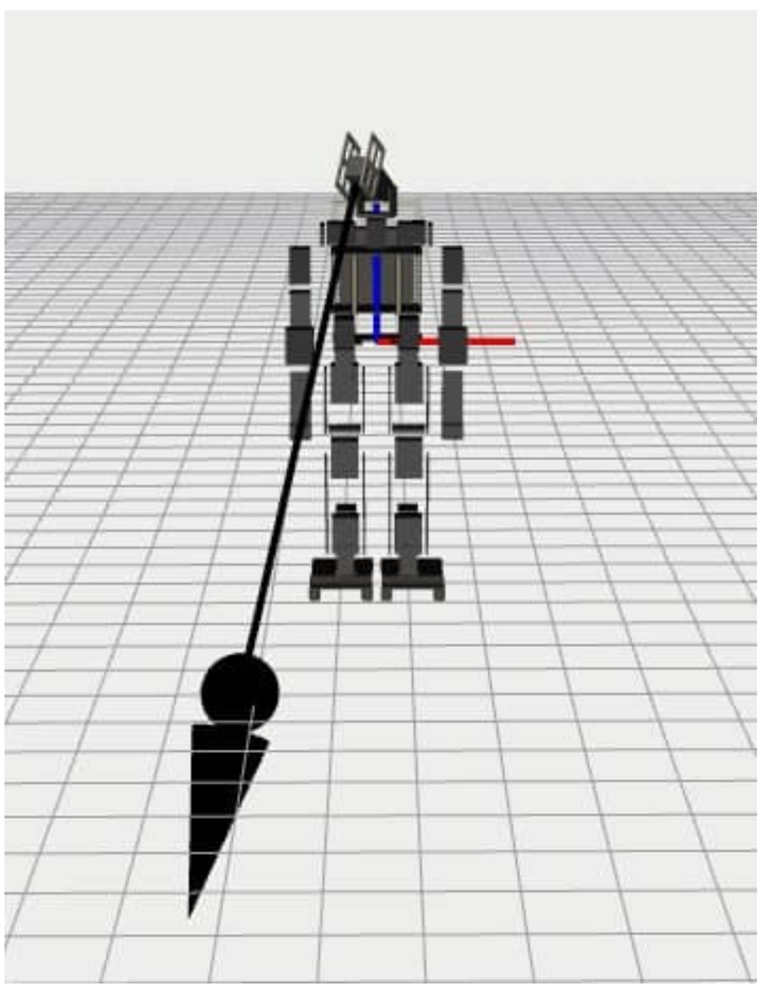


Рис. 1.3: Пример модели робота.

Следующий модуль — модель робота (рис. 1.3). С помощью нее мы понимаем как расположены суставы и, соответственно, как робот расположен в пространстве. Это необходимо для таких сложных движений как ходьба. Более того эта информация позволяет пересчитывать пиксельные координаты (то есть значения пикселей на изображении, которые отображают например мяч, или другой объект), в систему координат робота. То есть с помощью этого модуля мы можем понимать на каком расстоянии от робота расположен мяч.

Локализация

Перейдем к модулю локализации. Его цель — понять, где именно робот находится. Если это комната, то изначально нам нужно построить ее карту и далее ориентироваться по ней. В случае футбольного поля, нам достаточно информации о местоположении линий разметки и стоек ворот, чтобы понимать, где конкретно находится робот. Такая

система координат является неподвижной и задается программистом. Например, у нас центр системы координат находится точно в центре поля. Более того, когда мы знаем, где находится робот и знаем как относительно робота расположены объекты, мы можем найти их абсолютные координаты. То есть понимать, где именно в мире расположен объект.

Стратегия

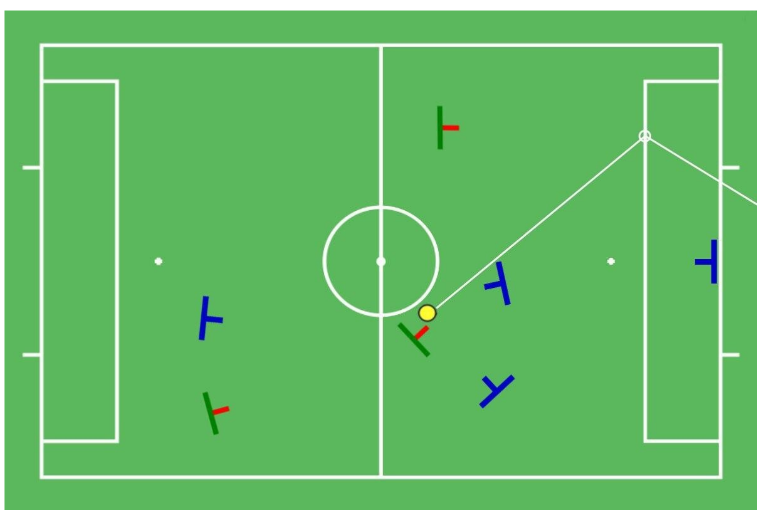


Рис. 1.4: Пример работы детектора мяча

Рассмотрим модуль стратегии. Он отвечает за поведение робота. Тут нам необходимо знать информацию о местоположении робота и объектов, с которыми он должен взаимодействовать. Давайте попробуем разобраться на простом примере (рис. 1.4). Допустим робот увидел мяч и знает, как этот мяч расположен относительно него. Он должен подойти и пнуть его. Так вот, чтобы понимать в каком направлении пинать, необходимо иметь представление о том, где находятся ворота. На самом деле стратегия занимается далеко не только этим. Она принимает на вход текущее состояние окружающей среды (в футболе это положение союзников, противников, мяча, свое собственное положение) и решает, что делать дальше. После того как робот понял, что нужно делать дальше, он должен совершить действие.

Движения

Это приводит нас к последнему модулю — модулю движения. В частном случае колесных роботов модуль движения зачастую достаточно прост. Но даже там, роботу необходимо плавно входить в повороты, останавливаться, разгоняться и так далее. Говоря о человекоподобных роботах, хорошие движения — половина успеха. Например, наш робот умеет ходить в разные стороны, пинать мяч несколькими способами и даже вставать, но к сожалению пока что падает чаще чем встает.

1.2 Семинар

В данном семинаре мы познакомимся с симулятором Webots. Он обладает многими плюсами — открытость кода, большая поддержка от комьюнити, удобство и простота в использовании. Мы разберемся с основными понятиями, научимся запускать мир, добавлять туда роботов и даже заставим их двигаться. Для начала работы требуется установить Webots. Актуальную версию можно скачать с сайта `https://cyberbotics.com/`. Сейчас мы ознакомимся с обучающей статьей[2] от создателей симулятора.

1.2.1 Мир

Мир (**World**) — это файл, содержащий информацию о том, где находятся объекты, как они выглядят, как взаимодействуют друг с другом, какого цвета небо, как определяется сила тяжести, трение, массы объектов и т.д. То есть он задает начальное состояние моделирования. Различные объекты называются узлами (**Nodes**) и иерархически организованы в дереве сцены (**Scene Tree**). Следовательно, узел может содержать подузлы. Мир хранится в файле с расширением **.wbt**. Файлы мира должны храниться непосредственно в каталоге *worlds*.

Упражнение 1 Приостановите текущее моделирование, нажав кнопку «Пауза». Симуляция остановлена, если остановлен виртуальный счетчик времени на главной панели инструментов. Создайте новый проект в меню **Wizards**, выбрав пункт меню **New Project Directory...** и следуйте инструкциям:

- Назовите каталог проекта *my_first_simulation* вместо предлагаемого *my_project*.
- Назовите файл мира **my_first_simulation.wbt** вместо предлагаемого **empty.wbt**.
- Установите все флажки, включая «Add a rectangle arena», который по умолчанию не отмечен.
- Нажмите кнопку «Finish» (Windows, Linux), чтобы закрыть это окно.

■

Вы только что создали свой самый первый мир Webots. На трехмерном изображении должна отображаться квадратная арена с клетчатым полом. Вы можете перемещать точку обзора в 3D-виде с помощью мыши: левой кнопки, правой кнопки и колеса.

В дереве сцены (**Scene Tree**) можно изменять узлы и поля. В настоящее время он должен перечислить следующие узлы:

- WorldInfo: содержит глобальные параметры моделирования.
- Точка обзора: определяет параметры камеры основной точки обзора.
- TexturedBackground: определяет фон сцены (вы должны увидеть горы вдалеке, если немного повернете точку обзора)
- TexturedBackgroundLight: определяет свет, связанный с указанным выше фоном.
- RectangleArena: определяет единственный объект, который мы видим на нашей сцене.

[2] `https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots?tab-language=python`

Каждый узел имеет несколько настраиваемых свойств, называемых полями. Давайте изменим эти поля, чтобы посмотреть, как меняется RectangleArena.

Упражнение 2 Дважды щелкните на RectangleArena в дереве сцены. Это должно открыть узел и отобразить его поля.

- Выберите поле floorTileSize и установите для него значение 0,25 0,25 вместо 0,5 0,5. Вы должны сразу увидеть эффект в 3D отображении мира.
- Выберите поле wallHeight и измените его значение на 0,05 вместо 0,1. Стена арены теперь должна быть ниже.

■

А сейчас давайте добавим новые объекты:

Упражнение 3 Дважды щелкните **RectangleArena** в дереве сцены, чтобы закрыть его, и выберите его. Нажмите кнопку «Add» в верхней части дерева сцены. В открывшемся диалоговом окне выберите **PROTO nodes (Webots Projects)** / **objects** / **factory** / **containers** / **WoodenBox (Solid)**. Посреди арены должен появиться большой ящик. Дважды щелкните по нему в дереве сцены, чтобы открыть его поля.

- Измените его размер (**size**) на 0,1 0,1 0,1 вместо 0,6 0,6 0,6.
- Измените его положение (**translation**) на 0 0,05 0 вместо 0 0,3 0.
- Теперь, удерживая нажатой клавишу «Shift», перетащите ящик в трехмерном виде и переместите его в какой-нибудь угол арены.
- Выберите ящик и нажмите ctrl-C, ctrl-V, чтобы скопировать и вставить его. Удерживая нажатой клавишу «Shift», перетащите новый ящик, чтобы переместить его в другое место. Создайте таким же образом третий ящик.
- Переместите все ящики из центра арены. Это можно сделать, удерживая нажатой клавишу «Shift» и перетащив правую кнопку мыши.
- Сохраните мир, нажав ctrl-s или на кнопку сохранения.

■

1.2.2 Робот

В данном разделе мы попробуем добавить колесного робота e-puck. Убедитесь, что симуляция приостановлена и что виртуальное время равно 0. Если это не так, сбросьте симуляцию с помощью кнопки «Reset». Когда мы изменяем мир в Webots и хотим его сохранить, очень важно, чтобы симуляция сначала была приостановлена и перезагружена в исходное состояние, то есть виртуальный счетчик времени на главной панели инструментов должен показывать 0:00:00:000. В противном случае при каждом сохранении положение каждого 3D-объекта может накапливать ошибки. Следовательно, любые модификации мира должны выполняться в таком порядке: приостановить, сбросить, изменить и сохранить симуляцию.

Нам не нужно создавать робота с нуля, нам просто нужно будет его импортировать. Этот узел на самом деле является узлом PROTO, таким как RectangleArena или WoodenBox, которые мы рассматривали ранее.

Упражнение 4 Выберите последний узел WoodenBox в дереве сцены. Нажмите кнопку «Add» вверху дерева сцены. В диалоговом окне выберите **PROTO nodes (Webots Projects) / robots / gctronic / e-puck / E-puck (Robot)**. В центре арены должен появиться робот. Его можно перемещать и вращать так же, как вы это делали с коробками. Сохраните симуляцию и нажмите кнопку «Play». ■

Робот должен начать двигаться, мигать светодиодами и избегать препятствий. Это происходит потому, что у него есть контроллер по умолчанию с таким поведением. Также мы можем наблюдать маленькое черное окно, появившееся в верхнем левом углу. В нем транслируется изображение, снятое камерой робота.

Давайте поэкспериментируем с физикой.

Упражнение 5 Применим силу к роботу, нажав alt + ctrl + левый щелчок + перетаскивание. Чтобы включить физику для наших ящиков, мы должны инициализировать их массу (**mass**) на определенное значение, например, 0.2 кг. Как только это будет сделано, мы сможем применить к ним силу. ■

1.2.3 Контроллер

Теперь мы запрограммируем простой контроллер, который просто заставит робота двигаться вперед.

Контроллер — это программа, которая определяет поведение робота. Поле controller узла Robot указывает, какой контроллер в настоящее время связан с роботом. Обратите внимание, что один и тот же контроллер может использоваться несколькими роботами, но робот может использовать только один контроллер одновременно. Каждый контроллер выполняется в отдельном дочернем процессе, обычно порождаемом Webots.

Упражнение 6 Создайте новый контроллер с именем epuck_go_forward с помощью меню Wizards / New Robot Controller .... Это создаст новый каталог epuck_go_forward в my_first_simulation / controllers. Выберите вариант, предлагающий открыть исходный файл в текстовом редакторе. ■

Новый исходный файл отображается в окне текстового редактора Webots. Теперь мы свяжем новый контроллер epuck_go_forward с узлом E-puck.

Упражнение 7 Давайте выберем наш новый контроллер, как основной для робота. Для этого зайдем в узел робота, в поле controller и нажав Select... выберем из списка контроллер epuck_go_forward. Пока что наш контроллер ничего не делает, давайте вставим следующий код в текстовое окно контроллера и запустим его:

```python
from controller import Robot, Motor

TIME_STEP = 64

# create the Robot instance.
robot = Robot()

# get the motor devices
leftMotor = robot.getDevice('left wheel motor')
rightMotor = robot.getDevice('right wheel motor')
# set the target position of the motors
leftMotor.setPosition(10.0)
rightMotor.setPosition(10.0)

while robot.step(TIME_STEP) != -1:
  pass
```

Сохраните измененный исходный код (File / Save Text File). ■

Если все в порядке, робот должен двигаться вперед. Он некоторое время будет двигаться с максимальной скоростью, а затем остановится, когда колеса повернутся на 10 радиан.

1.3 Практическое занятие

Определение

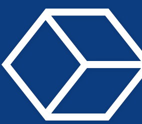
Часть 1:

- Создайте мир first_hw.wbt.
- Добавьте любую понравившуюся арену.
- Добавьте любого робота.
- Исследуйте его возможности.

Часть 2:

- Откройте мир robocubHLJS.
- Запустите симуляцию и проверьте, что игра запустилась.

2. Классическое зрение



2.1 Лекция

- Видишь суслика?
- Нет
- А он есть

ДМБ

Эта глава посвящена компьютерному зрению. В ней рассмотрены особенности канала зрения, представление изображений в компьютере, и детектирование объектов.

2.1.1 Особенности канала зрения

У автономных гуманоидных роботов, как и у людей, как правило именно канал зрения является самым важным и дающим наибольшее количество информации. Понятно, что без других не обойтись, нужно и чувствовать, как ноги давят на пол, и понимать, как расположены в пространстве части тела, но зрение все же выделяется на фоне остальных чувств. Задача компьютерного зрения в робототехнической системе - извлечение небольшого объема осмысленной информации из данных, поступающих с сенсора с самым большим потоком среди всех, которые есть у гуманоидного робота. Гироскоп дает сотни чисел в секунду, датчики силы нажатия ступней дают по порядку столько же, а камера — это миллионы чисел десятки раз в секунду. При этом каждый конкретный пиксель несёт достаточно мало информации, но их много, и объекты видны как группы пикселей, а зрение должно достаточно быстро построить для них короткое описание. Еще одна особенность этого канала восприятия состоит в том, что зрение обладает практически неограниченным рабочим диапазоном. Можно увидеть объект в километре от себя, а вот унюхать или потрогать уже не получится.

Визуальный канал восприятия работает за счет пассивных датчиков, в отличие от эхолокации дельфинов или летучих мышей. Некоторые 3D-камеры, такие как Intel Realsense или Microsoft Kinect, используют инфракрасную подсветку. Это позволяет им работать в темноте и надежно строить трехмерную карту местности, но с другой стороны это работает только в ограниченном диапазоне расстояний. В лиге робофутбола RoboCup использование датчиков с активной подсветкой запрещено, поскольку роботы должны быть похожи на людей, и предполагается, что в конечном итоге они будут играть против человеческой команды. Поэтому было бы справедливо ограничить физические и сенсорные возможности роботов, ориентируясь на людей. Их конструкция ограничивается правилами: у них не может быть больше двух камер, сами камеры должны находиться только на голове, они должны смотреть в одну сторону, не могут иметь активную подсветку или слишком большие углы обзора.

Хотя сенсорика роботов в некоторых отношениях, например в быстродействии, может превосходить человеческую, во многом она от человеческой отстает. В частности, глаз

человека может видеть и очень яркие, и очень темные объекты, а при достаточном времени адаптации способен работать в условиях освещенности, меняющейся в 10^{12} раз. Это не опечатка и не преувеличение, такие числа были бы более привычны в астрономии, но оказывается, что человеческий глаз на такое способен.

2.1.2 Постановка задачи детекции

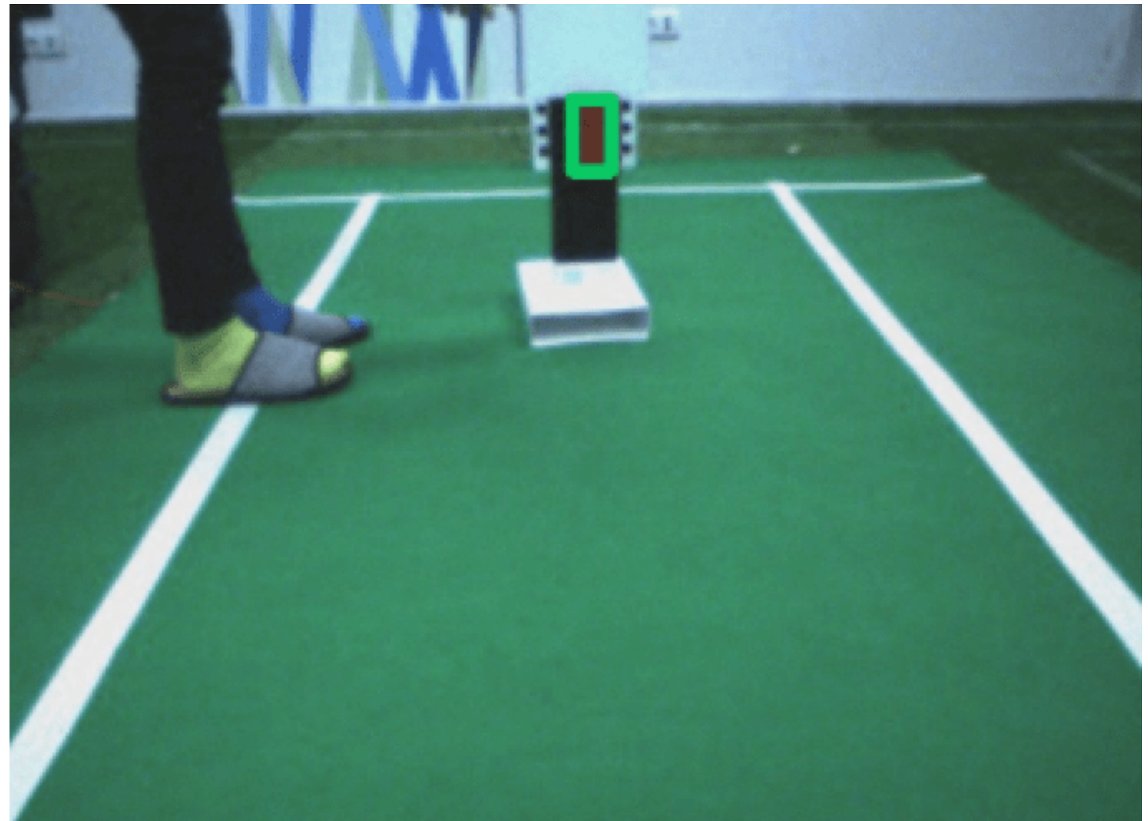


Рис. 2.1: Решенная задача детекции баскетбольного кольца для соревнования в лиге FIRA. Найденное кольцо обведено салатовым прямоугольником.

Самая классическая задача зрения в области — это детекция, то есть нахождение координат объекта в кадре. Мы будем рассматривать по большей части эту задачу в контекст робоспорта, но все подходы и методы запросто переносятся на другие задачи из других областей. Разберем задачу нахождения красного баскетбольного кольца, которое приведено на изображении. Оно тут довольно низко, но и робот, который пытался в это кольцо забить, небольшой. Формально говоря, на вход модулю детекции подается картинка, а на выходе у него пиксельные координаты объектов, то есть пары точек, обычно левый верхний и правый нижний углы прямоугольника, ограничивающие объект. Этот прямоугольник называют bounding box. Есть и другие задачи компьютерного зрения, которые возникают в процессе разработки программного обеспечения робота, например трекинга, (то есть отслеживания объекта от кадра к кадру,) или сегментации,

(разделения всех пикселей изображения на классы).

2.1.3 Представление изображения

Изображение в задачах компьютерного зрения, да и вообще в компьютерах часто бывает представлено в виде набора чисел в трехмерном массиве: две из размерностей соответствуют ширине и высоте изображения, а оставшаяся отвечает за три цветовых канала. Пиксель здесь — это совокупность яркостей по этим каналам: красному, зеленому и синему. В случае использования черно-белой камеры изображение будет одноканальным, то есть пиксель будет состоять из одного числа. Количество каналов в принципе может быть и другим, например, если добавить к обычным цветовым каналам еще инфракрасный или ультрафиолетовый, их станет больше. Для хранения компонент цвета в памяти часто используется тип данных $uint8$, то есть восьмибитные беззнаковые числа, принимающие значения от 0 до 255. Если зрение работает с частотой обновления кадров 10 fps на FullHD камере, то оно должно за секунду преобразовывать десятки миллионов чисел (все пиксели) в сотни чисел (bounding box).

2.1.4 Цветовое пространство RGB

Цветовое пространство RGB — это упомянутое выше представление цвета в виде совокупности трех компонент red, green и blue. Оно простое, интуитивное, прямо соотносится с принципом работы светочувствительной матрицы, но у него есть и слабые стороны. На рисунке 2.2 схематично изображено это цветовое пространство, и три оси x, y, z соответствуют трем базовым цветам. Любой цвет можно представить в виде их комбинации.

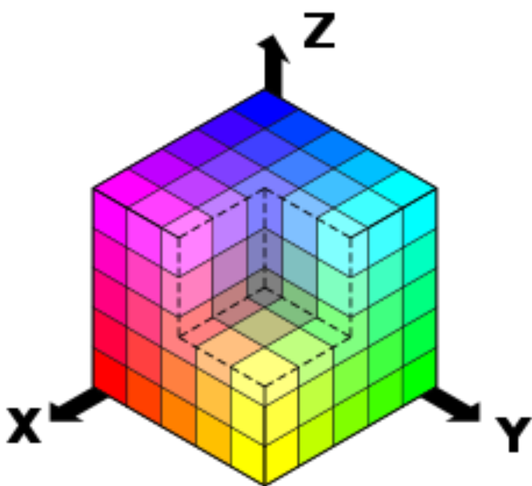


Рис. 2.2: Схематичное изображение цветового пространства RGB. Одна восьмая его часть не показана, но на месте угла, соответствующего максимуму по всем осям, находится белый цвет.

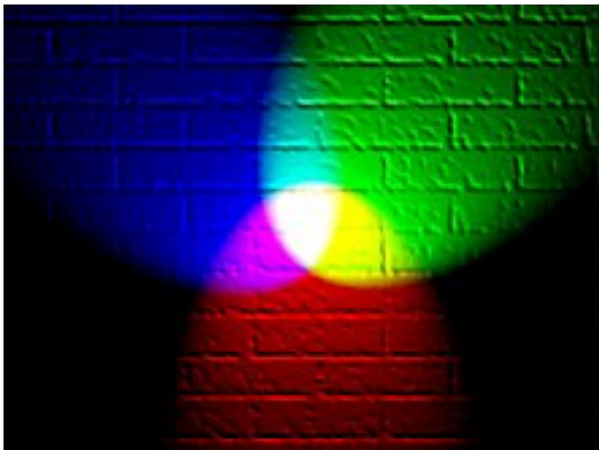


Рис. 2.3: Три базовых цвета в разных комбинациях дают все возможные цвета.

Простейший метод детектирования объекта на кадре — это выделение по цвету. В робофутболе довольно долго использовались цветные мячи как раз потому, что практически любой бортовой компьютер справится с задачей его нахождения. Сейчас в играх университетских команд используются мячи обычной расцветки, а в детских лигах роботы по-прежнему играют контрастными цветными мячами. Объектом будем считать компактно расположенное множество пикселей характерного цвета.

Итак, стандартная процедура нахождения цветного объекта такова. Сначала строится одноканальное изображение пространственными размерами с исходное. В этом изображении, называемом маской, будет всего два цвета: черный с яркостью 0 и белый с яркостью 255. Если пиксель исходного изображения подходит по цвету под объект, то в маске соответствующий пиксель помечается белым, в противном случае — черным. Это и есть простейший цветовой фильтр, это if, который применяется ко всем пикселям изображения сразу.

На рисунке схематично изображено распределение яркостей в канале красного цвета. На исходном изображении есть зеленое поле, занимающее большую его часть и имеющее в среднем низкие значения интенсивности по каналу красного цвета. Полю соответствует глобальный максимум, находящийся в левой части распределения. А ближе к правой его части расположен локальный максимум, соответствующий небольшому скоплению красных пикселей на месте баскетбольного кольца. Желтыми вертикальными чертами обозначены верхняя и нижняя границы яркости красного цвета.

Затем маску нужно очистить от шума и наконец найти в ней наибольшую подходящую область.

2.1.5 Цветовое пространство HSV

При разработке системы зрения для робота как правило стремятся к тому, чтобы она была как можно менее чувствительна к условиям освещения. Но в цветовом пространстве RGB яркость по всем трем каналам возрастает при увеличении яркости источника света, например при переходе в другое помещение. Если в качестве критерия выделения объекта

Рис. 2.4: Бинарная маска, построенная применением цветового фильтра к исходному изображению

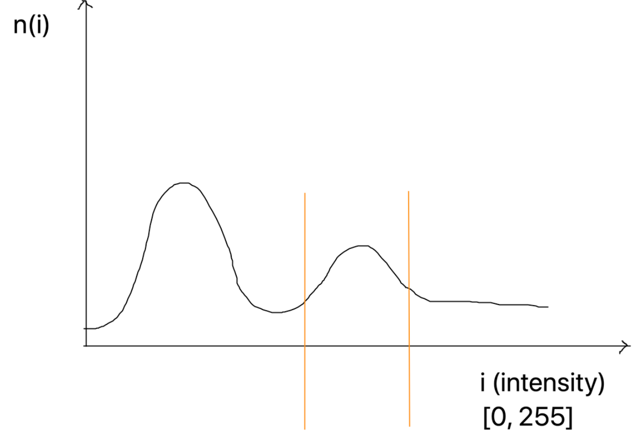


Рис. 2.5: Схематично изображенное распределение яркостей в красном канале

были выбраны конкретные численные значения яркостей цветов, это приведет к тому, что маска объекта изменится, а при сильном изменении освещения вовсе перестанет

соответствовать этому объекту.

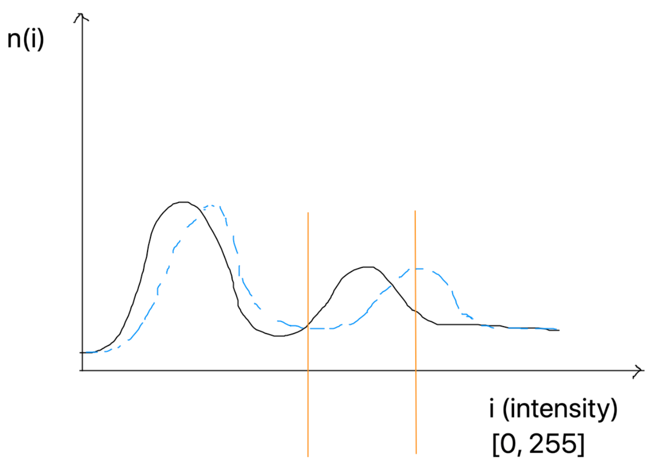


Рис. 2.6: Сдвиг распределения яркостей при изменении условий освещения

Другие цветовые пространства, например HSV, сконструированы таким образом, что позволяют в значительной степени избегать этих эффектов. В нем тоже три канала, но не базовые цвета, а цветовой тон, насыщенность и яркость. Цветовой тон (Hue) — это число, которое можно представить себе в виде координаты в радуге. Насыщенность — это близость цвета к серому: чем он ближе, тем ниже насыщенность. Яркость - это эквивалент масштаба, в котором цвет с заданным тоном и заданной насыщенностью представлен.

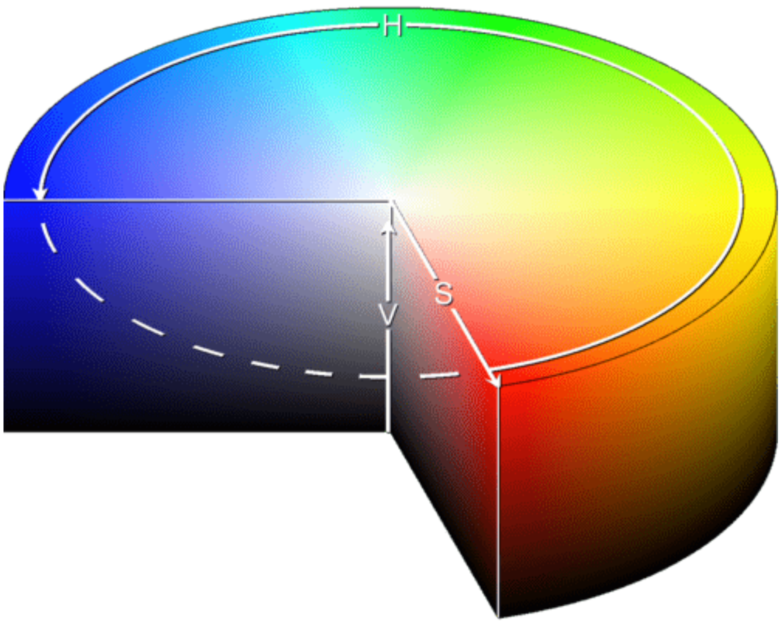


Рис. 2.7: Цветовое пространство HSV

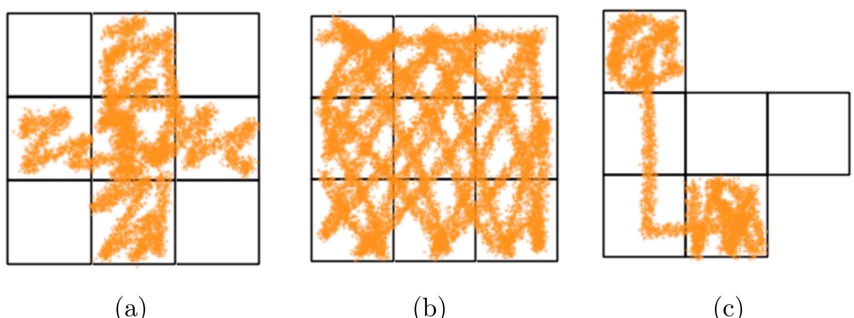


Рис. 2.8: (a) 4-связность (b) 8-связность (c) связное множество

При изменении яркости освещения изменение цвета преимущественно происходит именно в канале яркости. Если установить по яркости достаточно широкие пороги, практически проигнорировав эту компоненту, то ее изменения в некоторых пределах не будут отражаться на получающейся маске.

Перевод изображения из одного цветового пространства в другое как правило является нелинейной операцией. За счет этого и использования чисел конечной точности небольшое изменение значения в одном может вызывать большое изменение в другом. В случае цветовых пространств RGB и HSV это происходит в области низких яркостей.

2.1.6 Компоненты связности

Рассмотрим более подробно маски объектов, получающиеся в результате выделения по цвету. При правильном подборе граничных значений объектам соответствуют «компактные» области белых пикселей, иначе говоря, с белыми пикселями соседствуют белые пиксели.

Подойдем к этому чуть более формально. Назовем связной компонентой (или компонентой связности) такое множество пикселей, для которого верно, что можно попасть из любого в любой, оставаясь внутри множества и делая переходы между соседями. В качестве соседей пикселя обычно рассматриваются или 4 (сверху, снизу, справа, слева), или 8 (добавляя стоящие по диагонали) граничных с ним.

При рассмотрении задачи детектирования объекта в типичной для соревновательной робототехники постановке от модуля зрения часто требуется найти пиксельные координаты нижней точки объекта, например мяча, стойки ворот, другого робота. Таким образом, нужно перейти от точной маски объекта к чему-то, что характеризует местонахождение объекта как целого. Это удобно делать с помощью введения так называемого bounding box, или ограничивающего прямоугольника, т.е. прямоугольника минимальной площади со сторонами, параллельными осям координат, который полностью содержит связную компоненту, соответствующую детектируемому объекту.

За счет шума матрицы, не совсем точного подбора параметров фильтров маски обычно содержат некоторое количество ошибочно классифицированных пикселей. Часто они встречаются на границах объектов, что приводит в частности к некорректной

оценке положения и размера bounding box, а из этого напрямую следуют неточности геометрического позиционирования объекта в системе координат робота. Для того, чтобы сделать области маски более гладкими, чтобы увеличить их или уменьшить, можно использовать так называемые морфологические операции, изменяющие форму компонент связности. Они формально задаются с помощью операции свертки как частного случая фильтра.

Рис. 2.9: Пример применения морфологической эрозии

Для связной компоненты можно найти параметры, описывающие соответствующий ей объект, с точки зрения геометрических свойств. Понятно, что можно найти высоту и ширину, это просто размеры bbox. Но помимо этого можно находить такие вещи, как отношение высоты к ширине, степень округлости, плотность, т.е. отношение количества белых пикселей к площади (в пикселях) bbox. Если нам известно, что детектируемый объект - это стойка ворот, то отношение высоты к ширине должно быть достаточно большим. Если мы знаем, что это мяч, то теоретическая оценка плотности — это $\frac{\pi}{4}$.

Классическое компьютерное зрение хорошо подходит для решения простых задач, в которых можно запросто формализовать признаки детектируемого объекта. Например, если он имеет характерный цвет, характерную форму. Он вытянутый, круглый, синий, монотонно окрашенный. С помощью классического зрения можно найти на кадре руку, колесо, стойки футбольных ворот. Но есть великое множество задач, в которых классическое зрение или бессильно вовсе, или по тем или иным причинам работает заметно хуже нейросетей. Такова в частности детекция объектов, для которых нет адекватного словесного описания, напрямую сводимого к операциям с изображениями. Рассмотрим для примера изображение на слайде.

Тут и цвет похожий, и форма похожая. Но человеку очевидно, в чем дело, а классический алгоритм можно было бы обмануть. Понятно, что можно придумать хитрый критерий, позволяющий находить шерсть, но такой подход потребовал бы постоянной ручной доработки признаков, их настройки и комбинирования. Более того, есть задачи, которые в принципе не решаются классическими алгоритмами. Например, фотореалистичная генерация лиц.

На сайте thispersondoesnotexist.com при каждом его посещении генеративно-состязательная нейросеть создает новое изображение человека, и такие изображения зачастую трудно

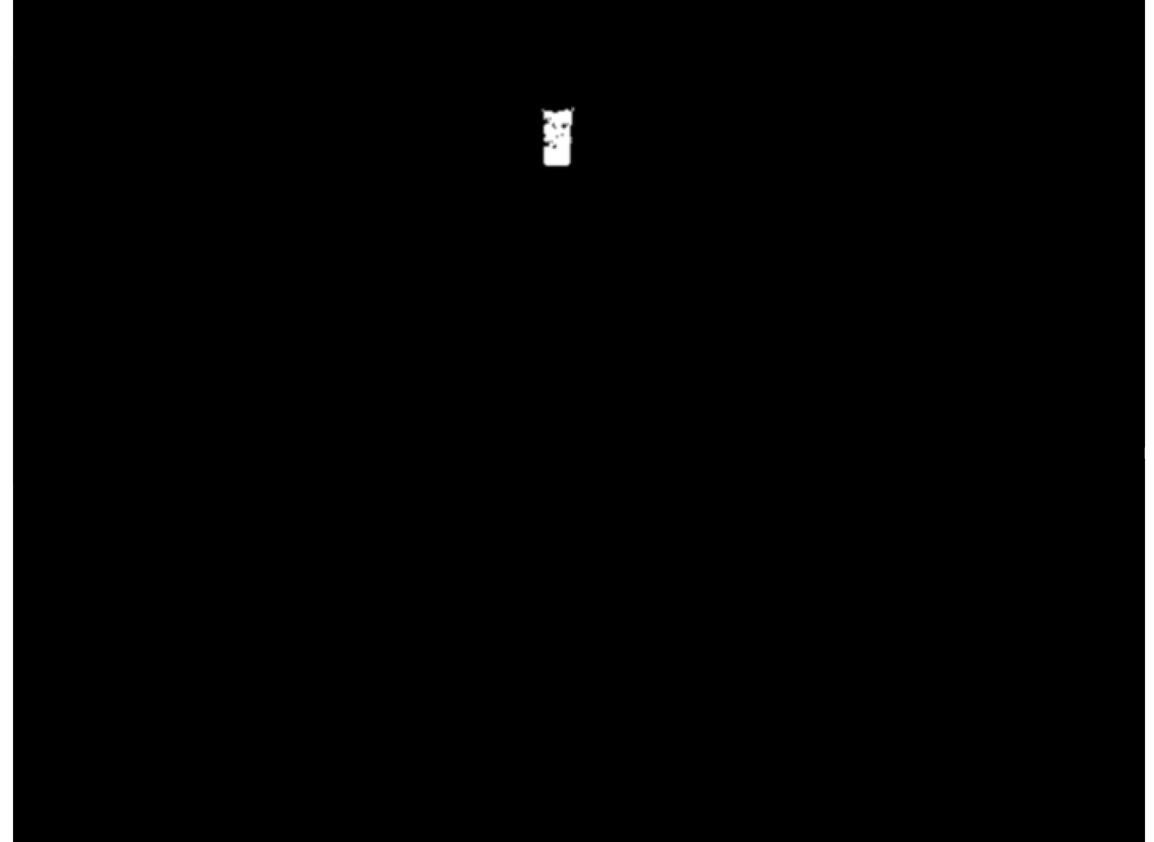


Рис. 2.10: Маска, очищенная от шума

отличить от настоящих. Еще одна такая задача — это автоматизированный поиск заболеваний на снимках магнитно-резонансной томографии. В некоторых случаях врачи учатся годами, чтобы правильно определить и локализовать заболевание, и ясно, что просто так накидать пару фильтров и решить задачу не получится. Классические алгоритмы требуют ручного подбора признаков и настройки, но если задача несложная, ее можно решить довольно быстро. А для того, чтобы нейросеть заработала, ее нужно обучить, и обучение требует размеченных данных, или датасетов, и зачастую довольно больших. Для обучения современных нейросетевых детекторов требуется порядка тысячи картинок на один класс объектов. Но это уже совсем другая история.

2.1.7 Вопросы после лекции

Контрольный вопрос 1 Что произойдет, если вместо трех каналов при построении маски использовать только один? Настроить фильтр на конкретный объект будет проще или сложнее?

Контрольный вопрос 2 Сколько килобайт на диске будет занимать несжатое трехканальное изображение разрешением 640×480?

Контрольный вопрос 3 Как в цветовом пространстве RGB будет представляться

Рис. 2.11: Объекты, для которых трудно подобрать явное описание

Рис. 2.12: Пример лица человека, сгенерированного нейронной сетью

темно-серый цвет? Желтый цвет?

Контрольный вопрос 4 Какой тип данных обычно используется в робототехнике для хранения изображений?

Контрольный вопрос 5 За что отвечают каналы цветового пространства HSV? Какие пороговые значения для функции получения маски нужно задать, чтобы найти только контрастные объекты низкой яркости? Светлые объекты, близкие по цвету к серому?

Контрольный вопрос 6 Какие характеристики компонент связности нужно использовать, чтобы отличать по маскам на виде сбоку толстых бульдогов от худых борзых?

Контрольный вопрос 7 Как с помощью морфологических операций залить небольшие области черного внутри маски?

Контрольный вопрос 8 Что будет происходить с детектированием в цветовом пространстве RGB, если разместить в помещении несколько дополнительных ламп того же спектра, что и уже работающие? Что произойдет в HSV при правильной настройке?

Контрольный вопрос 9 Можно ли поставить камеры на ноги робота, принимающего участие в соревнованиях лиги RoboCup?

Контрольный вопрос 10 Может ли красный объект выглядеть так же, как зеленый, после преобразования, переводящего изображение в оттенки серого?

Контрольный вопрос 11 Сколько есть цветов в RGB, у которых во втором канале значение 120? 130? В HSV?

Контрольный вопрос 12 Как найти контрастный объект на однородном фоне, если не известны их цвета?

Контрольный вопрос 13 Может ли площать bounding box-а объекта превышать его площадь?

Контрольный вопрос 14 В чем причина появления шума при построении маски?

Контрольный вопрос 15 Совпадет ли изображение после перевода из RGB в HSV и обратно с исходным? Почему?

Контрольный вопрос 16 Как с помощью классического зрения обнаружить, что на защитном стекле камеры скопились пылинки?

Контрольный вопрос 17 С помощью каких преобразований можно уменьшить видимый шум на изображении? Какие у этого оборотные стороны?

Контрольный вопрос 18 Как с помощью классического компьютерного зрения посчитать число автомобилей, которые проезжают через ворота?

Контрольный вопрос 19

Вопросы, выходящие за пределы программы.

Контрольный вопрос 20 В какой части спектра чувствительность человеческого глаза максимальна? Что такое кандела?

Контрольный вопрос 21 Что такое слепое пятно, как можно убедиться в его существовании?

Контрольный вопрос 22 У какого датчика угловое разрешение выше, у камеры 8k или у человеческого глаза?

Контрольный вопрос 23 Как определить, есть ли в кадре блики или источники света (например, лампы)?

Контрольный вопрос 24 Как с помощью классического компьютерного зрения найти синий стакан, стоящий на деревянном столе, если в кадре есть и другие синие стаканы?

Контрольный вопрос 25 Что в устройстве камеры соответствует человеческому хрусталику, радужке, сетчатке?

2.2 Семинар

Для выполнения следующих пунктов понадобится компьютер с Jupyter Notebook или интерпретатором питона, а также установленная библиотека OpenCV. Если в коде встречается что-то непонятное, попробуйте спросить у преподавателя или обратиться к документации.

2.2.1 Загрузка изображений с диска

Упражнение 8 Загрузите изображение с диска и выведите его на экран с помощью следующего кода:

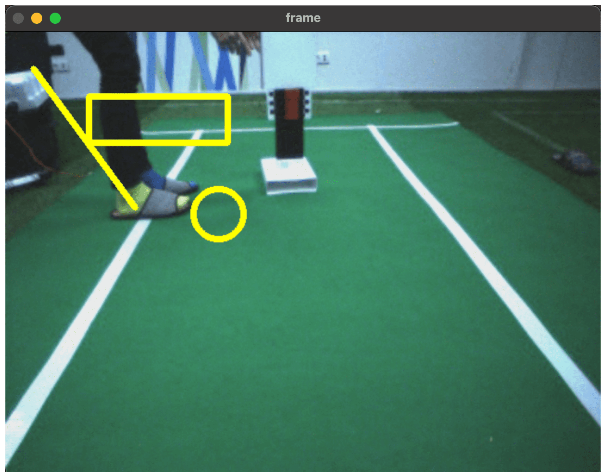
```

import cv2

image = cv2.imread("data/rgb_basket.jpg")
cv2.imshow("frame", image)

cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1)

```

■

Выведите размеры загруженного изображения, распечатав image.shape. Для приведённого изображения это (482, 644, 3) по высоте, ширине и глубине соответственно.

Упражнение 9 Нарисуйте поверх изображения линию, окружность, прямоугольник жёлтого цвета с помощью функций *cv2.line*, *cv2.rectangle* и *cv2.circle*. Откройте документацию к OpenCV, чтобы найти описание и порядок параметров, а также примеры использования.

■



Упражнение 10 Уменьшите изображение в три раза по оси y с помощью функции *cv2.resize*, выведите результат на экран. ■

Упражнение 11 Прибавьте к изображению 50 (пользуясь тем, что изображение - это numpy массив), выведите результат на экран. Прибавьте 100, 200, 256. ■

Упражнение 12 Примените к загруженному изображению метод cv2.inRange, выведите получившуюся маску. ■

2.3 Распознавание паттернов

Эта глава посвящена распознаванию образов, или распознаванию паттернов, или поиску шаблонов. Названий тут можно придумать много, но все они в конечном счете будут отсылать к тому, что для выполнения сложных операций и людям, и роботам, которых они программируют, приходится опираться на те или иные абстракции. Когда мы описываем внешность человека, мы не говорим, что в трех сантиметрах от его переносицы находится сегмент такого-то цвета. Мы можем сказать, что он шатен с голубыми глазами, и это уже дает достаточно много информации, поскольку в библиотеке внешностей содержится такой шаблон.

И для построения компактных описаний, и для создания понятных алгоритмов с доказуемой корректностью нужно вырабатывать механизмы разделения наблюдаемого мира на набор примитивов. В этой главе рассмотрено нахождение углов, а также детектирование геометрических фигур, таких как линии, окружности и эллипсы.

Начнем с того, что такое в принципе этот самый паттерн, образ или шаблон. Это вещь достаточно фундаментальная, но не в строгом смысле слова, а скорее в житейском,

поэтому все определения будут более или менее маханием руками. Для понимания можем договориться, что мы говорим о детектировании объектов некоторого класса, которые отделимы от объектов других классов. В задаче детектирования окружностей таким объектом будет окружность, а параметрами, однозначно задающими ее, будут координаты центра и радиус. Но в принципе паттерном можно назвать и что-то совсем другое, рыбью чешую к примеру. Попробуйте придумать, какие параметры могут быть у рыбьей чешуи.

Не буду долго останавливаться на том, что паттерны встречаются в природе, это и так понятно. Вернее будет сказать, что мы воспринимаем через паттерны все, в том числе и природу. А что касается чешуи, для нее можно ввести что-то похожее на амплитуду, фазу и частоту для электрических сигналов. Они будут описывать направление, в котором расположены чешуйки, их размер и некий начальный сдвиг.

Если говорить о такой задаче, как трекинг объекта, или идентификация его на следующем кадре видео, если известна его позиция на предыдущем, то тут впору вспомнить метод, о котором мы уже немного говорили на лекции по стерео. Он называется Блочным сопоставлением, или сопоставлением с шаблоном, или Template Match.

Использовать этот метод имеет смысл тогда, когда искомый объект слабо меняется между кадрами. Строго говоря, его работоспособность опирается на то, что он не меняется вовсе. В формуле справа T большое — это шаблон, I большое — это изображение, а R большое — это карта того, насколько область изображения похожа на шаблон. В качестве метрики различия используется сумма квадратов по всем пикселям из окрестности. То есть еще раз, происходит следующее. Для каждого пикселя изображения мы прикладываем туда шаблон его центральным пикселем и находим поэлементную разность шаблона и того места картинки, к которому он приложен. Потом возводим все разности в квадрат и складываем. Чем меньше эта сумма, тем больше область изображения похожа на шаблон.

Слева на слайде приведена инвертированная карта различия, и самая яркая точка на ней соответствует как раз лицу футболиста справа. Результатом такой процедуры поиска становится точка с минимальной разностью.

Такой метод детекции прост и работает достаточно быстро, но в целом на этом его плюсы заканчиваются. Человеку вполне очевидно, что если объект повернуть, он останется тем же самым объектом, а вот для алгоритма блочного сопоставления это не так. Помимо этого объект потеряется, если на изображении он станет меньше. Минусов много, но тем не менее у этого алгоритма есть свои применения, в которых они не страшны.

Еще на лекции по стерео мы касались детектирования углов, ну или просто выделяющихся точек. Давайте попробуем посмотреть на эту задачу с точки зрения поиска некоего паттерна. Что такое угол? Ну наверное это точка, в которой пересекаются две границы. А что такое граница? Это множество точек, в котором есть доминирующее направление градиента, т.е. перепада яркости.

2.3.1 Фильтры

Для того, чтобы поговорить об этом чуть более предметно, давайте познакомимся со свертками. Во многом операция свертки похожа на то, о чем мы говорили только что, описывая поиск по шаблону. Только здесь происходит не поэлементное вычитание, а поэлементное умножение. Давайте посмотрим на пример одномерной свертки.

Одномерный сигнал приведен слева, одномерное изображение длины 3, называемое ядром свертки, — в середине, а результат свертки — справа. Для каждой позиции ядра происходит поэлементное умножение и сложение, а затем результат помещается в выходной массив.

С двумерными сигналами происходит в сущности то же самое, только окрестность текущего пикселя теперь двумерная. Посмотрим на то, как можно задать сглаживающий фильтр с помощью свертки. Что произойдет, если произвести свертку изображения с ядром в виде матрицы из единиц 5 на 5, как на слайде слева?

Согласно определению свертки, яркости всех пикселей в окрестности, задаваемой ядром, будут умножены на 1 и сложены. Понятно, что если сглаживанию было подвергнуто монотонное изображение, то оно должно остаться неизменным. Естественное решение этой проблемы — нормировка, то есть деление ядра свертки на сумму элементов в нем, если она не равна нулю.

Давайте обратимся к еще одному примеру фильтра, и он будет не сглаживать изображение, а наоборот, выделять границы. В каком месте изображения есть граница? Там, где значения с одной стороны отличаются от значений с другой стороны. Так давайте же вычтем их друг из друга. На слайде приведены примеры конкретных ядер, выделяющих границы по горизонтали и по вертикали.

Значение градиента вдоль оси x не зависит от значения в точке, но зависит от значений левее и правее. Ну и учитывается, правда с меньшим весом, разница в интенсивностях выше и ниже.

Величину градиента в точке можно определить как корень из суммы квадратов градиентов по осям.

Обратите внимание на границы шляпы на крайнем правом изображении. Они как раз и видны в виде ярких областей на изображении с градиентами, поскольку шляпа резко контрастирует с более темным задником. А на синих перьях градиенты везде большие, поскольку этот объект очень детализирован и фактически весь состоит из резких переходов.

2.3.2 МНК

После того, как мы научились находить такие вот простые паттерны, давайте перейдем к чему-то более комплексному. Поговорим о том, как найти на изображении прямую, а если точнее, то ее модель, имея только зашумленные данные, то есть пары точек x1, y1, x2, y2 и так далее.

Давайте подойдем к этой проблеме аналитически и попробуем найти выражение от исходных данных, задающее прямую, наилучшим образом описывающую имеющиеся данные. Пускай модель прямой — это альфа плюс бета икс. Тогда определим ошибку

на конкретной паре икс итое игрек итое как разность между реальным игрек итым и тем, которое предсказала модель. Ошибка модели для всего набора данных приведена на второй строке.

А теперь давайте возьмем и продифференцируем эту ошибку по альфа и по бета. После некоторых преобразований окажется, что оптимальная по такой метрике ошибки прямая имеет параметры альфа с шапочкой бета с шапочкой. Икс и игрек с чертой здесь — это средние значения.

Хорошо в этом методе то, что он работает быстро и практически не требует дополнительной памяти. Для ситуаций, когда все наблюдения действительно описываются одной-единственной моделью, он подходит отлично. Но такая модель, как видно из формул, зависит от всех данных. И если в них есть какое-то ну совсем уж шумное наблюдение, оно все равно будет учтено. Сломался датчик, в соседней с вашей установкой комнате хлопнули дверью, да мало ли что может случиться, такой метод исправно учтет показания любой степени надежности. И даже один выброс на самом деле может сильно испортить конечный результат.

Можно попытаться учесть возможность появления выбросов, как-то доделать, докрутить, адаптировать алгоритм под такие применения, но мы сейчас перейдем к совершенно другому классу подходов, которые специально созданы для обработки шумных данных.

2.3.3 Преобразование Хафа

Поговорим об одном замечательном методе нахождения объектов, задаваемых набором параметров. Таковы например квадрат, круг, парабола, плоскость, если мы живем в трехмерном мире. Явно выделить параметры для сложного объекта вроде птицы или робота вряд ли получится. Ну или параметров этих будет столько, что преобразование Хафа будет неприменимо.

Предыстория появления этого метода восходит к науке и к лени. В начале 20-го века люди вовсю изучали элементарные частицы с помощью пузырьковых камер, камер Вильсона. Пролетающая по перенасыщенному пару частица оставляла за собой след из капелек сконденсированной воды. Для фиксации траектории нужно было быстро фотографировать частицу, поскольку такие следы недолговечны. А вот после этого лаборанты должны были руками находить параметры траекторий этих частиц. Это занимало много времени и допускало возможность ошибки. И естественно, данные были шумными, так что нужен был очень устойчивый метод.

Ближе к делу. Как можно задать всевозможные прямые с помощью двух параметров? Ну вроде бы коэффициент наклона и константа, а икс плюс б. Но тут проблема с тем, что во-первых, нельзя задать вертикальную прямую, а во-вторых, единичное изменение коэффициента приводит к разным наблюдаемым изменениям прямой для разных значений этого коэффициента.

Поэтому зачастую имеет смысл использовать другую параметризацию прямой — через угол поворота и расстояние от начала координат. Можно представить в таком виде любую прямую, и любой комбинации угла и сдвига соответствует некоторая прямая.

Оригинальный алгоритм преобразования Хафа предполагает полное построение пространства параметров для искомой кривой. Забегая вперед скажем, что размерность,

т.е. число осей в этом пространстве, равна размерности детектируемой кривой: для прямой это два, для окружности три, для эллипса пять. В виде кода пространство параметров для прямых — это двумерный массив, в котором по одной размерности откладываются всевозможные углы с некоторым шагом, а по другой — всевозможные расстояния от начала координат с некоторым шагом. Понятно, что угол тут должен быть от нуля до пи, а расстояние не должно превышать длину диагонали изображения. Использование конечного шага оборачивается тем, что начиная с некоторого момента, например для разницы меньше чем в один градус, прямые считаются совпадающими.

Мы сейчас долго запрягаем, но уже скоро станет ясно, что к чему. Каждый элемент этого массива соответствует некоторой прямой, и каждая прямая может быть отнесена к некоторому элементу аккумулятора, если мы используем конечный шаг.

Через одну точку входных данных может пройти множество прямых. Каждая такая прямая — это элемент аккумулятора. Давайте переберем их все и прибавим по плюс единичке ко всем таким элементам. Одна точка исходных данных отобразится в кривую как на слайде слева. Дальше давайте сделаем так для всех исходных данных. И наконец устроим голосование.

Тогда получится, что те точки, которые лежат на искомой прямой, будут голосовать одинаково, а те, которые не лежат, будут голосовать как попало, и их показания будут расходиться. Справа приведен пример аккумулятора, в котором видны два максимума. Они соответствуют двум прямым в исходных данных.

Нужно сказать и об одной особенности, ограничивающей применение этого метода к сложным объектам. В классической реализации размерность аккумулятора равна размерности объекта, т.е. числу его параметров, и если хочется найти эллипсы к примеру, нужно заводить пятимерный аккумулятор. С сотней разных значений по каждой оси и одним байтом на каждую ячейку, что вообще говоря немного, это будет уже десять гигабайт оперативной памяти.

2.3.4 RANSAC

Может возникнуть вопрос, а для чего нам вообще детектировать эллипсы? Кому придет в голову этим заниматься? Ну вот оказывается, что эллипсы всплывают в неожиданных местах. Например, в задаче локализации робота-футболиста. Круг в центре поля виден роботу именно как эллипс. Другой пример, из жизни кстати, это детектирование концов труб на складе.

Давайте посмотрим на задачу совсем с другой стороны. Попытаемся вместо одной большой модели, как в методе наименьших квадратов, построить много маленьких, и выберем среди них, скажем так, наименее плохую. Метод, основанный на этом принципе, называется ransac или random sample consensus. Он был предложен в восьмидесятых годах и с тех пор нашел много применений в компьютерном зрении.

На самом деле его возможности выходят далеко за пределы нахождение геометрических фигур на изображениях. К примеру, он позволяет оценивать преобразования пространства, переводящие одну камеру в другую, да и вообще любые объекты, для которых можно построить модель.

Каноничной пример — это, конечно, прямые, так что разберемся с рансаком на тех

же данных, а именно на наборе пар икс итое игрек итое. Возьмем из набора данных две произвольные точки. Они однозначно задают единственную прямую. Это будет наша рабочая гипотеза. Найдем, сколько точек из данных находится к построенной линии достаточно близко. Их мы назовем инлаерами, или нормальными точками. А точки за пределами некоей полосы вокруг построенной модели будем называть аутлаерами, или выбросами. Построим еще одну такую модель для двух других случайно выбранных точек. И еще одну, и еще одну. И выберем среди них модель, которая наилучшим образом описывает наблюдения.

Существует вероятностный метод, позволяющий оценить необходимое число итераций для достижения корректного результата с любой наперед заданной вероятностью. Оно дано формулой слева. Здесь дубльве — это доля инлаеров от нуля до единицы, p — желаемая вероятность корректного детектирования, а n — число точек, однозначно задающих модель. Для прямых это будет 2, для окружности 3, а для эллипсов 5, поскольку эллипсы — это пятипараметрические объекты. Они однозначно задаются координатами центра, размерами полуосей и углом наклона.

2.3.5 TinyYOLO

На десерт поговорим о теме, которая относится к распознаванию образов скорее по касательной, а на смысловой карте всего курса в целом лежит где-то между ним и стерео. Я расскажу о статье, которую написала наша лаборатория, посвященной детектированию с помощью нейросетей. Мы не меняли архитектуру сети, только подали в нее чуть-чуть геометрических данных, и качество работы улучшилось.

На момент написания статьи в наших роботах стояло по одной камере, а для мяча использовалась замысловатая двухуровневая система детектирования, которая кроме всего прочего учитывала предположения о геометрических размерах мяча, если бы он лежал в том месте, где он был найден. Мы хотели поменять ее на нейросеть, которая выдавала бы bounding box, получив на вход картинку. Естественно, хотелось дать нейросети возможность использовать те же самые геометрические данные.

На лекции про стерео мы говорили о том, что, зная положения всех сервомоторов и обладая информацией с гироскопа, а также всеми параметрами камеры, можно для конкретного пикселя найти, с какой точки поля в него попадает информация. А можно сделать так для всех пикселей кадра, составив одноканальное изображение, в котором будут храниться данные о том, насколько далеко от точки съемки находился бы конкретный объект, если бы он стоял на полу.

Мы сделали именно так и подали это изображение четвертым каналом вместе с тремя исходными каналами.

Оказалось, что это действительно повышает интересную нам метрику, особенно при высоких требованиях к качеству детектирования. Это стоило нам небольшого замедления в работе сети, но при этом мы например сенсор не меняли, т.е. повышение качества фактически было достигнуто исключительно программно.

Можно соотнести результаты детектирования стандартной нейросетью yolov3-tiny слева и нашей версией справа. Если присмотреться, становится ясно, в чем все дело. Уверенность на мячах выше, робот в левой части сцены задетектирован точнее, а

правый вообще задетектирован только нашей версией сети. Подача дополнительного канала позволяет нейросети принимать во внимание не только цветовые характеристики объектов, но еще и геометрические, потому что фактически канал масштаба, который мы добавили, позволяет соотнести число пикселей и сантиметров в каждой точке поля зрения.

Наш подход время от времени пригождается другим робототехникам, в частности нас процитировала команда, которая занимается разработкой робота для сборки бананов. Ну а у нас с тех пор, как на роботах появилась вторая камера, необходимость в таких фокусах отпала.

2.4 Стерео

Сегодня поговорим о том, как устроены различные цифровые камеры, которые применяются в современных автономных роботах. А также коснемся бинокулярного зрения, обработки и хранения стереоизображений.

2.4.1 Цифровая камера

Начнем с камеры, которую мы будем называть «обычной». Это камера, работающая в оптическом диапазоне с трехканальной картинкой на выходе. Т.е. выдающая массив из чисел скажем 1920 на 1080 на 3, где 3 — это красный, зеленый, синий, 30 раз в секунду. Такие камеры стоят в ноутбуках, веб-камеры тоже обычно такие, и во многих любительских роботах применяются именно они.

Они работают как пассивные датчики, т.е. регистрируют свет, приходящий извне, с помощью массива светочувствительных элементов. Камеры, которые снимают на пленку, тут бы не подошли, а цифровые камеры переводят свет в электрические сигналы, и именно это позволяет использовать их в робототехнике. Светочувствительные элементы для всех трех базовых цветов используются одинаковые, но над ними расположены светофильтры в таком вот шахматном порядке. Элементов, чувствительных к зеленому, тут в два раза больше, и поэтому чувствительность самая высокая именно в этой части спектра.

Мы уже говорили о цветовых пространствах, и тут можно сказать, что используется цветовое пространство RGB. Значение сигнала в каждом из каналов пропорционально мощности приходящего света.

С камерами бывают разные чудеса. Можно этого не замечать, но обычные пользовательские камеры постоянно поднастраиваются, чтобы картинка была сбалансированной по яркости белого в кадре, фокусируются. Лучше всего это заметно, если поднести ладонь близко к веб-камере ноутбука. Для звонков по скайпу это замечательно, а вот если некоторый фильтр однажды был настроен, то после такого срабатывания одному изготовителю известного алгоритма его снова нужно будет настраивать. Но автоматическую настройку всевозможных параметров обычно можно программно отключить, например с помощью opencv или usb драйвера v4l2.

Однажды я купил несколько дешевых камер в переходе, чтобы с ними экспериментировать. И заметил, что при некоторых условиях на картинке появлялись белые пятна, а

если держать камеру рукой, то они пропадали. Оказалось, что поскольку камера была очень низкого качества, пластик корпуса был полупрозрачным, и свет попадал внутрь не только через объектив, но еще и с другой стороны. В итоге я обклеил камеру пластырем, и это помогло.

У камеры есть много важных параметров, включая и баланс белого, и фокусное расстояние, и выдержку, но поговорим сейчас про те, на которые нужно особенно обращать внимание при выборе камеры. Баланс белого можно настроить, фокусное расстояние зачастую в принципе тоже можно, выдержку можно регулировать, а вот три следующих характеристики останутся неизменны. Это разрешение, углы обзора и тип затвора. Поговорим о них отдельно.

С разрешением проще всего. Чем больше разрешение, тем выше детализация объектов на изображении и тем больше данных нужно кодировать, передавать и хранить. Если очень грубо уподобить человеческий глаз камере, то разрешение его будет порядка 100 мегапикселей. Понятно, что тонкостей масса, и плотность расположения колбочек и палочек на сетчатке непостоянна, и сигналы на самом деле передаются не от каждой светочувствительной клетки в отдельности, но порядок величины можно почувствовать. Угловое разрешение глаза, то есть минимальный угол между объектами, при котором они воспринимаются как различные, составляет около трех сотых градуса и близок к таковому у хороших современных камер.

Тут нужно понимать, что чудес не бывает и что можно сделать разрешение камеры хоть пятьсот мегапикселей, но чтобы это имело смысл, нужна соответствующая оптика. В конце концов, есть дифракционный предел, да и коэффициент преломления в материале линзы на самом деле зависит от длины волны.

И о скорости обработки данных тоже не стоит забывать: бывает такое, что робототехники вынуждены ограничивать разрешение потока с камеры, чтобы поддерживать достаточную частоту обработки кадров. В одном из роботов на основе смарт-камеры, то есть компьютера в сущности достаточно маломощного, мы в погоне за скоростью и экономией памяти использовали разрешение в шестнадцать раз меньше максимально возможного.

Еще одна важная характеристика камеры, а если более конкретно, то оптики, — это углы обзора. Чем шире обзор, тем большая часть сцены попадает в поле зрения. С другой стороны, снижается уровень детализации. У широкоугольных линз масса достоинств, в частности и для применения в автономной робототехнике. Когда на наших роботах оптика поменялась с обычной на широкоугольную, робот смог гораздо меньше крутить головой, а видел он при этом больше, чем раньше. Когда он стоит в центре поля, он может одновременно наблюдать все четыре стойки ворот. Есть и оборотная сторона — картинка становится сильно искаженной. Этот эффект рыбьего глаза еще можно увидеть на камерах GoPro и широкоугольных камерах на телефонах. На краю картинки плотность пикселей на квадратный сантиметр сцены сильно ниже, чем в центре.

Если на роботе используются нейросети, при их обучении должна учитываться вариативность в том, как может выглядеть один и тот же объект в разных частях кадра.

Менее очевидная, но тоже очень важная характеристика камеры — это тип затвора, или шаттера, в обиходе он так называется. В обычных камерах используется так на-

зываемый бегущий (или сканирующий) затвор, или rolling shutter. Снятие данных со светочувствительной матрицы происходит строка за строкой, и это занимает какое-то время, обычно порядка миллисекунд. Если за это время объект успевает сдвинуться, то на снимке он будет искажен.

Есть так называемый глобальный затвор, или global shutter. В этом случае информация со всей матрицы считывается одномоментно. Тут есть свои минусы: больший размер каждого пикселя, большее энергопотребление и как следствие более сильный нагрев. Обычно у global shutter камер небольшое разрешение, а стоят они в сравнении с обычными очень много. Но на динамичных роботах, включая беспилотные машины, обычно применяются именно они, поскольку допускать искажения линий разметки за счет особенностей затвора тут нельзя.

2.4.2 Модель камеры

Поговорим о том, как происходит формирование изображения при съемке. Есть сцена, то есть трехмерные объекты в мире, с поверхности которых в сторону камеры излучаются фотоны. Нас интересует, в каких пиксельных координатах, то есть где на изображении, окажется каждый из видимых объектов. Другими словами, хочется построить функцию, которая отображает три координаты в мире в две координаты на изображении. Оказывается, что при построении полноценной модели камеры необходимо учесть следующие факторы.

Во-первых, это внутренние параметры камеры. Давайте разберемся, что они собой представляют. В массивах нумерация обычно начинается с нуля, а на изображении удобно отсчитывать пиксели, откладывая их от оптической оси. Помимо этого, камеры не бывают идеальными, и на самом деле оптическая ось обычно проходит не через середину матрицы. c_x и c_y — это пиксельные координаты оптической оси относительно угла матрицы. а f_x и f_y — это фокусное расстояние, выраженное в пикселях.

Второй фактор при построении изображения — это так называемая дисторсия, возникающая за счет особенностей конкретной оптической системы. Она бывает положительной и отрицательной, наиболее привычной будет скорее всего эффект рыбьего глаза, или fish eye. Соответствующие преобразования описываются вот таким образом. Коэффициенты k_1, k_2 и так далее находят в процессе калибровки камеры, как и перечисленные выше внутренние параметры.

Для калибровки используются несколько десятков фотографий калибровочного шаблона, обычно это шахматная доска, с разных ракурсов. На них сначала находятся углы, а потом запускается итерационный процесс, который находит и коэффициенты дисторсии, и внутренние параметры камеры, и положения точек съемки.

В конечном итоге все эти вещи нужны для того, чтобы из пиксельных координат, которые возвращает модуль зрения, получать информацию о том, где в мире находятся объекты. Из внутренних параметров камеры можно понять, как в мире расположен луч, проходящий через фокус и через конкретный пиксель матрицы. Но для определения расстояния до объекта еще нужно понимать, где на этом луче объект находится, так что по одной камере нельзя выяснить положение объектов без какой-либо дополнительной информации о сцене.

Часто бывает, что это самое дополнительное предположение о сцене состоит в том, что объекты находятся на плоскости. В частности, такое предположение в ходу в робофутболе, и это оправданно. Получается, что луч, проходящий через пиксель матрицы и фокус, в одной точке пересекает плоскость земли, и именно в этой точке по логике вещей находится объект, найденный модулем зрения.

2.4.3 Стерео

Когда сенсоры и вычислительные мощности позволяют, используется представление объектов как групп точек в трехмерном пространстве. Стереозрение позволяет увидеть в объеме настоящие наблюдения, не предсказания, и поэтому у него вообще говоря шире область применения, но разумеется для построения работающей системы стереозрения нужно решить тысячу технических проблем: стереокалибровку, синхронизацию камер, ну и канал для передачи данных должен быть достаточно широким.

Бинокулярное зрение основывается на том, что для разных точек съемки один и тот же объект будет выглядеть чуть-чуть по-разному. И чем ближе объект к камере, тем больше расхождение между изображениями. Например, собственный нос человек видит с двух разных сторон без пересечения. Еще одна забавная вещь та, что пока человеку не напомнишь, что он видит нос, он его не видит.

Расхождение в пиксельных координатах на двух изображениях называется диспаратностью. Для стереопары, то есть двух изображений одного и того же, снятых одновременно с разных точек, можно построить карту диспаратности. В ней для каждого пикселя будет находиться расхождение пиксельных координат между снимками. Для простоты стереопары обычно делают горизонтальными или приводят к горизонтальным. В них диспаратность есть только по оси x, и при этом объект на правой картинке всегда левее.

Казалось бы, для того, чтобы с двумя глазами, стереозрение работало, у них должны пересекаться области видимости. Но вот у голубей например они практически не пересекаются. Однако эти птицы прекрасно ориентируются в пространстве. В полете точка съемки для каждого глаза меняется достаточно быстро и просто за счет линейного перемещения, а на земле они двигают головой вперед-назад, чтобы постоянно обновлять карту диспаратности.

Диспаратность сама по себе не очень интересна, это какие-то пиксели, на которые сдвинут каждый объект. Но ее можно перевести в карту глубины, и тогда каждому пикселю будет соответствовать его глубина в сцене. Для такого преобразования нужны будут описанные выше внутренние параметры камер и информация об их взаимном расположении.

2.4.4 Стереокамеры

Поговорим о том, какими бывают стереокамеры. Они могут быть с активной подсветкой и без. Сначала поговорим о камерах вроде Intel RealSense или Microsoft Kinect, которые подсвечивают сцену в невидимой для человека части спектра, а именно в инфракрасном диапазоне. В RealSense например есть обычная камера оптического диапазона,

то есть с тремя каналами, есть инфракрасный проектор, который накладывает специальный шаблон на окружающие предметы, и две инфракрасные камеры, снимающие этот шаблон с разных точек. При этом вся обработка происходит на устройстве, и для пользователя realsense выглядит как волшебная камера, которая снимает объекты в объеме. Электрическая мощность у нее при этом около 5 ватт она может питаться через type-c кабель, и эта камера — частый выбор для установки на автономного робота. Помимо всего прочего, у нее есть модификации с global shutter.

Такие камеры работают в темноте, в отличие от пассивных датчиков.

В робофутболе сенсоры с активной подсветкой запрещены, поскольку роботы должны быть достаточно похожи на людей. Остается только использовать стерео второго типа, то есть на обычных пассивных камерах. Вопрос: как можно для пикселя с левого кадра найти, где он на правом? Понятно, что нужно построить описание, желательно уникальное для этого пикселя, да еще и такое, которое было бы удобно сравнивать с описаниями пикселей с другого кадра.

Мы рассмотрим две задачи. Первая — это нахождение соответствия между двумя пикселями, относящимися к одному и тому же объекту в общем случае, то есть когда окрестность может быть и повернута, и растянута. Это может пригодиться, если хочется найти сдвиг и поворот камеры между двумя кадрами. А вторая — это построение полной карты глубины для выровненных друг относительно друга картинок.

Достаточно хорошее описание пикселя — это его окрестность. Но проблема здесь в том, что окрестность размера один, то есть только сам пиксель, в принципе не уникальна, а окрестности большего размера не инвариантны к поворотам. Проще говоря, окрестность может выглядеть по-разному на разных снимках. Так что такие описания можно использовать для решения второй задачи — для построения карты глубины по выровненной стереопаре.

Так называемое блочное сопоставление, или block matching, основывается на том, чтобы для окрестности пикселя с одного изображения найти такой сдвиг, при котором на другой снимок эта окрестность накладывается наилучшим образом. Мера несоответствия изображений — это или сумма модулей разностей, или сумма квадратов разностей. То есть происходит следующее. Берется окрестность каждого пикселя с левого снимка. Для нее перебираются разные сдвиги и с такими сдвигами окрестность сравнивается с тем, что находится на другом изображении. Ответом считается сдвиг, на котором достигается минимальное различие.

У блочного сопоставления есть много настроечных параметров, и зачастую к результату бывает нужно применять фильтрацию, сглаживание и другую постобработку.

А вот если нужно построить преобразование, переводящее первую точку съемки во вторую, можно находить соответствия не для всех пикселей, а только для небольшого их числа, зато для очень информативных. Обычно на изображениях бывают не просто россыпи разноцветных пикселей, а пиксели, объединенные в группы по признаку принадлежности к некоторому объекту. И это сопряжено с пространственной близостью на кадре. Нужно искать соответствие между углами и другими примечательными точками на двух кадрах, а не между всеми подряд.

Давайте разберемся, почему так. Если посмотреть на равномерно залитую неоновым

светом однотонную стену, глазу зацепиться будет не за что, потому что все пиксели стены плюс-минус одинаковые. Если на стене нарисована вертикальная контрастная линия, то можно по крайней удерживать внимание на ней. А если их две и они пересекаются, то такая точка становится отличным ориентиром для нахождения на двух изображениях сразу. Примерно из таких соображений точки для сопоставления на двух изображениях и выбираются. Они еще называются ключевыми точками, или key points. Точки на прямой имеют большое значение численно найденного градиента, то есть грубо говоря перепада яркости, а для пересечения прямых это верно в двух направлениях. Хорошая ключевая точка — это к примеру угол.

После того, как были найдены ключевые точки, нужно построить для них описания. Чтобы описания можно было удобно сопоставлять между собой, для них нужно определить меру различия, или метрику. Оказывается, что достаточно хорошо работают описания в виде векторов, для которых можно находить обычные евклидовы расстояния.

Поговорим более подробно о том, как строится одно из самых широко используемых векторных описаний окрестности пикселя — SIFT, или Scale-Invariant Feature Transform. В 16 квадратах 4 на 4 вокруг точки строятся распределения направлений градиентов, и затем они записываются в единый стодвадцативосьмимерный вектор с разными нормализациями, чтобы сделать векторы инвариантными к поворотам и аффинным преобразованиям, насколько это возможно.

Ну и после этого можно за два вложенных цикла найти для каждой точки пару.

У представления мира в трехмерном формате, будь то карта глубины или облако точек, есть много разных применений. Приведем некоторые из них.

Например, использование облаков точек сильно упрощает детектирование противников в робофутболе. Обход противников был важной составляющей нашей тактики в чемпионате 2021 года, где мы заняли первое место. По точкам в трехмерном пространстве, которые видит робот, можно построить приближение плоскости земли. А все то, что в этой плоскости не находится и при этом достаточно высокое — это препятствия, ну или противники.

Одно из занятных применений карты глубины — это интерполяция кадра для двух точек съемки. Предположим, что у нас есть два изображения одной и той же сцены при разных положениях камеры. Проще говоря, есть стереопара. Так вот, если карта диспаратности — это грубо говоря на сколько позиций нужно сдвинуть пиксели левой картинки, чтобы получить правую, то можно домножить ее на 0.5 и получить, на сколько позиций нужно сдвинуть пиксели левой картинки, чтобы получить картинку, снятую с виртуальной камеры, находящейся между двумя реальными.

2.4.5 Работа с видео

Давайте рассмотрим пример работы с видео. Откроем файл и будем покадрово считывать и выводить его на экран, пока что без какой-либо обработки.

```
1
2 import numpy as np
3 import cv2
```

```

video_name = "unicycle.mp4"

cam = cv2.VideoCapture(video_name)

while(True):
    success, frame = cam.read()

    if (success == False):
        print("reading failed")

        ####YOUR CODE BELOW
        cam.release()

        cam = cv2.VideoCapture(video_name)

        continue
        ####YOUR CODE ABOVE

    cv2.imshow("frame", frame)

    key = cv2.waitKey(40) & 0xFF

    if (key == ord('q')):
        break

cam.release()
cv2.destroyAllWindows()
cv2.waitKey(10)

```

2.4.6 Распределения в каналах

Эта и следующие программы предназначены для самостоятельного изучения. В зависимости от формата занятий их можно или писать с нуля, или разбирать уже готовые решения. Для всех этих программ в видеокурсе есть подробный разбор.

```

import numpy as np
import cv2
import matplotlib.pyplot as plt

def print_val(val):
```

```
7        print(val)
8
9    def nothing(val):
10       pass
11
12   #функция построения и рисования распределения
13   def plot_dist(channel):
14       fig, ax = plt.subplots()
15       ax.hist(channel.ravel(), 25, [0,256])
16
17       fig.canvas.draw()
18       dist = np.array(fig.canvas.renderer.buffer_rgba())
19       plt.close()
20
21       return dist
22
23   cv2.namedWindow("frame")
24
25   cv2.createTrackbar("value", "frame", 20, 200, nothing)
26
27   video_name = "unicycle.mp4"
28
29   cam = cv2.VideoCapture(video_name)
30
31   skip_frame_reading = False
32
33   success, frame_ = cam.read()
34
35   while(True):
36       if (skip_frame_reading == False):
37           success, frame_ = cam.read()
38
39       if (success == False):
40           print("reading failed")
41
42           cam.release()
43
44           cam = cv2.VideoCapture(video_name)
45
46           continue
47
48       w, h, _ = frame_.shape
```

```
    frame = cv2.resize(frame_, (h // 2, w // 2))

    val = cv2.getTrackbarPos("value", "frame")

    frame[:, :, 0] += np.uint8(val)

    #########################
    # будет построено распределение яркостей в нулевом канале,
    # то есть frame[:, :, 0]

    dist_0 = plot_dist(frame[:, :, 0])
    dist_1 = plot_dist(frame[:, :, 1])
    dist_2 = plot_dist(frame[:, :, 2])

    dists = np.concatenate((dist_0, dist_1, dist_2), axis=1)

    wd, hd, _ = dists.shape

    dists_resized = cv2.resize(dists, (h // 2, int(w // 2 * h // 2 / hd)))

    #обратите внимание, что число каналов в изображении в графиками
    #равно четырем, и его нажно сократить до 3, чтобы
    #сконкатенировать с кадром
    result = np.concatenate((frame, dists_resized[:, :, :3]), axis=0)

    cv2.imshow("frame", result)

    #########################

    key = cv2.waitKey(240) & 0xFF

    if (key == ord('q')):
        break

    if (key == 32):
        skip_frame_reading = not skip_frame_reading

cam.release()
cv2.destroyAllWindows()
cv2.waitKey(10)
```

2.4.7 Сдвиг распределения в одном из каналов

```

import numpy as np
import cv2
import matplotlib.pyplot as plt

def nothing(val):
    pass

#функция построения и рисования распределения
def plot_dist(channel):
    fig, ax = plt.subplots()
    ax.hist(channel.ravel(), 25, [0,256])

    fig.canvas.draw()
    dist = np.array(fig.canvas.renderer.buffer_rgba())
    plt.close()

    return dist

cv2.namedWindow("frame")

cv2.createTrackbar("value", "frame", 100, 200, nothing)

video_name = "unicycle.mp4"

cam = cv2.VideoCapture(video_name)

skip_frame_reading = False

success, frame_ = cam.read()

while(True):
    if (skip_frame_reading == False):
        success, frame_ = cam.read()

    if (success == False):
        print("reading failed")

        cam.release()

        cam = cv2.VideoCapture(video_name)
```

```
            continue

        w, h, _ = frame_.shape

        frame = cv2.resize(frame_, (h // 2, w // 2))

        val = cv2.getTrackbarPos("value", "frame")

        frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        #frame_hsv[:, :, 1] += np.uint8(val - 100)

        frame_hsv[:, :, 1] = cv2.add(frame_hsv[:, :, 1], val - 100)

        frame = cv2.cvtColor(frame_hsv, cv2.COLOR_HSV2BGR)

        dist_0 = plot_dist(frame[::10, ::10, 0])
        dist_1 = plot_dist(frame[::10, ::10, 1])
        dist_2 = plot_dist(frame[::10, ::10, 2])

        dists = np.concatenate((dist_0, dist_1, dist_2), axis=1)

        wd, hd, _ = dists.shape

        dists_resized = cv2.resize(dists, (h // 2, int(w // 2 * h // 2 / hd)))

        result = np.concatenate((frame, dists_resized[:, :, :3]), axis=0)

        cv2.imshow("frame", result)

        key = cv2.waitKey(240) & 0xFF

        if (key == ord('q')):
            break

        if (key == 32):
            skip_frame_reading = not skip_frame_reading

cam.release()
cv2.destroyAllWindows()
cv2.waitKey(10)
```

```
```

2.4.8 Построение маски объекта

```

import numpy as np
import cv2
import copy
from IPython.display import clear_output

def nothing(val):
    pass

cv2.namedWindow("frame")

cv2.createTrackbar("lb", "frame",   7, 255, nothing)
cv2.createTrackbar("lg", "frame",  14, 255, nothing)
cv2.createTrackbar("lr", "frame",  78, 255, nothing)
cv2.createTrackbar("hb", "frame",  25, 255, nothing)
cv2.createTrackbar("hg", "frame",  80, 255, nothing)
cv2.createTrackbar("hr", "frame", 185, 255, nothing)
cv2.createTrackbar("ksz", "frame", 0, 20, nothing)

img = cv2.imread("objects.jpg")

while(True):
    frame_ = copy.deepcopy(img)

    w, h, _ = frame_.shape

    frame = cv2.resize(frame_, (h // 4, w // 4))

    ksz = cv2.getTrackbarPos("ksz", "frame") + 1
    frame = cv2.blur(frame, (ksz, ksz))

    lb = cv2.getTrackbarPos("lb", "frame")
    lg = cv2.getTrackbarPos("lg", "frame")
    lr = cv2.getTrackbarPos("lr", "frame")
    hb = cv2.getTrackbarPos("hb", "frame")
    hg = cv2.getTrackbarPos("hg", "frame")
    hr = cv2.getTrackbarPos("hr", "frame")

```

```
    frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    mask = cv2.inRange(frame_hsv, (lb, lg, lr), (hb, hg, hr))

    mask_3ch = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    res = np.concatenate((frame, mask_3ch), axis=1)

    cv2.imshow("frame", res)
    #cv2.imshow("mask", mask)

    print(lb, lg, lr, hb, hg, hr)

    key = cv2.waitKey(240) & 0xFF

    clear_output(wait=True)

    if (key == ord('q')):
        print(lb, lg, lr, hb, hg, hr)
        break

cv2.destroyAllWindows()
cv2.waitKey(10)

```

2.4.9 Морфологические операции

```

import numpy as np
import cv2
import matplotlib.pyplot as plt

mask = np.zeros((10, 15), np.uint8)

mask[3:8, 2:6] = 1
mask[5,  4:10] = 0

kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], np.uint8)

```

```
processed = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

plt.imshow(mask, cmap="gray")
plt.show()

plt.imshow(processed, cmap="gray")
plt.show()

import numpy as np
import cv2
import copy
from IPython.display import clear_output

img = cv2.imread("objects.jpg")[::3, ::3, :]

def nothing(v):
    pass

cv2.namedWindow("result")

cv2.createTrackbar("lb", "result", 7, 255, nothing)
cv2.createTrackbar("lg", "result", 11, 255, nothing)
cv2.createTrackbar("lr", "result", 87, 255, nothing)

cv2.createTrackbar("hb", "result", 22, 255, nothing)
cv2.createTrackbar("hg", "result", 74, 255, nothing)
cv2.createTrackbar("hr", "result", 183, 255, nothing)

cv2.createTrackbar("blur_ksz", "result", 0, 55, nothing) #4
cv2.createTrackbar("morph_ksz", "result", 3, 25, nothing)

while (True):
    frame = copy.deepcopy(img)

    lb = cv2.getTrackbarPos("lb", "result")
    lg = cv2.getTrackbarPos("lg", "result")
    lr = cv2.getTrackbarPos("lr", "result")
    hb = cv2.getTrackbarPos("hb", "result")
    hg = cv2.getTrackbarPos("hg", "result")
    hr = cv2.getTrackbarPos("hr", "result")

    blur_ksz  = cv2.getTrackbarPos("blur_ksz", "result") + 1
```

```
    morph_ksz = cv2.getTrackbarPos("morph_ksz", "result") * 2 + 1

    frame_b = cv2.blur(frame, (blur_ksz, blur_ksz))

    frame_hsv = cv2.cvtColor(frame_b, cv2.COLOR_BGR2HSV)

    lth = (lb, lg, lr) #low threshold
    hth = (hb, hg, hr) #high threshold

    clear_output(wait=True)

    print(lth, hth)

    mask = 255 - cv2.inRange(frame_hsv, lth, hth)

    kernel = np.ones((morph_ksz, morph_ksz), np.uint8)

    mask_morph = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask_morph = cv2.morphologyEx(mask_morph, cv2.MORPH_OPEN, kernel)

    mask3c = cv2.cvtColor(mask_morph, cv2.COLOR_GRAY2BGR)

    res = np.concatenate((frame_b, mask3c), axis=1)

    cv2.imshow("result", res)

    key = cv2.waitKey(90) & 0xFF

    if (key == ord('q')):
        break

#cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(100)

```

2.4.10 Обработка связных компонент

```

import numpy as np
import cv2
```

```
import copy
from IPython.display import clear_output

img = cv2.imread("objects.jpg")[::3, ::3, :]

def nothing(v):
    pass

cv2.namedWindow("result")

cv2.createTrackbar("lb", "result", 7, 255, nothing)
cv2.createTrackbar("lg", "result", 11, 255, nothing)
cv2.createTrackbar("lr", "result", 87, 255, nothing)

cv2.createTrackbar("hb", "result", 22, 255, nothing)
cv2.createTrackbar("hg", "result", 74, 255, nothing)
cv2.createTrackbar("hr", "result", 183, 255, nothing)

cv2.createTrackbar("blur_ksz", "result", 0, 55, nothing) #4
cv2.createTrackbar("morph_ksz", "result", 3, 25, nothing)

cv2.createTrackbar("area_th", "result", 0, 1500, nothing)

while (True):
    frame = copy.deepcopy(img)

    lb = cv2.getTrackbarPos("lb", "result")
    lg = cv2.getTrackbarPos("lg", "result")
    lr = cv2.getTrackbarPos("lr", "result")
    hb = cv2.getTrackbarPos("hb", "result")
    hg = cv2.getTrackbarPos("hg", "result")
    hr = cv2.getTrackbarPos("hr", "result")

    blur_ksz  = cv2.getTrackbarPos("blur_ksz", "result") + 1
    morph_ksz = cv2.getTrackbarPos("morph_ksz", "result") * 2 + 1

    area_th = cv2.getTrackbarPos("area_th", "result")

    frame_b = cv2.blur(frame, (blur_ksz, blur_ksz))

    frame_hsv = cv2.cvtColor(frame_b, cv2.COLOR_BGR2HSV)

```

```
    lth = (lb, lg, lr) #low threshold
    hth = (hb, hg, hr) #high threshold

    clear_output(wait=True)

    print(lth, hth)

    mask = 255 - cv2.inRange(frame_hsv, lth, hth)

    #MORPHOLOGY
    kernel = np.ones((morph_ksz, morph_ksz), np.uint8)

    mask_morph = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask_morph = cv2.morphologyEx(mask_morph, cv2.MORPH_OPEN, kernel)

    #CONNECTED COMPONENTS FILTERING
    connectivity = 4
    # Perform the operation
    output = cv2.connectedComponentsWithStats(mask_morph,
                connectivity, cv2.CV_32S)
    # Get the results
    # The first cell is the number of labels
    num_labels = output[0]
    # The second cell is the label matrix
    labels = output[1]
    # The third cell is the stat matrix
    stats = output[2]
    # The fourth cell is the centroid matrix
    centroids = output[3]

    for i in range(1, num_labels):
        a = stats[i, cv2.CC_STAT_AREA]
        t = stats[i, cv2.CC_STAT_TOP]
        l = stats[i, cv2.CC_STAT_LEFT]
        w = stats[i, cv2.CC_STAT_WIDTH]
        h = stats[i, cv2.CC_STAT_HEIGHT]

        comp = np.where(labels == i)

        density = a * 1.0 / (w * h)

        if (t < 300):
```

```
            mask_morph[comp] = 0

    print("test", num_labels)

    mask3c = cv2.cvtColor(mask_morph, cv2.COLOR_GRAY2BGR)

    res = np.concatenate((frame_b, mask3c), axis=1)

    cv2.imshow("result", res)

    key = cv2.waitKey(90) & 0xFF

    if (key == ord('q')):
        break

#cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(100)

```

2.4.11 Работа с контурами

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import copy

img = cv2.imread("rails.jpg")

blur_k_sz = 15
morp_k_sz = 19
area_th = 2000
contour_eps = 0.02

blurred = cv2.blur(img, (blur_k_sz, blur_k_sz))

hsv = cv2.cvtColor(blurred, cv2.COLOR_RGB2HSV)

single_channel = hsv[:, :, 2]

ret, otsu = cv2.threshold(single_channel, 0, 255,
```

```
            cv2.THRESH_BINARY + cv2.THRESH_OTSU)

ker = np.ones((morp_k_sz, morp_k_sz), np.uint8)

opened = cv2.morphologyEx(otsu, cv2.MORPH_OPEN, ker)

closing_ker = np.ones((morp_k_sz + 10, morp_k_sz + 10), np.uint8)

closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE, closing_ker)

output = cv2.connectedComponentsWithStats(closed, 4, cv2.CV_32S)
num_labels = output[0]
labels = output[1]
stats = output[2]

filtered = copy.deepcopy(closed)

for i in range(num_labels):
    a = stats[i, cv2.CC_STAT_AREA]

    if (a < area_th):
        filtered[np.where(labels == i)] = 0

contours, _ = cv2.findContours(filtered, cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)

for cont in contours:
    perim = cv2.arcLength(cont, True)
    approx = cv2.approxPolyDP(cont, contour_eps * perim, True)

    cv2.drawContours(img, [approx], -1, (0, 255, 0), 3)

plt.imshow(blurred)
plt.show()
plt.imshow(single_channel)
plt.show()
plt.imshow(otsu)
plt.show()
plt.imshow(opened)
plt.show()
plt.imshow(filtered)
plt.show()
```

```
plt.imshow(img)
plt.show()
```

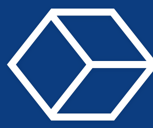
2.4.12 Вычитание фона

```
import numpy as np
import cv2
import copy

video_name = "unicycle.mp4"

cam = cv2.VideoCapture(video_name)

_, frame = cam.read()

background = copy.deepcopy(frame)

morp_k_sz = 3
rho = 0.04
area_th = 10000

i = 0

while(True):
    success, frame = cam.read()

    if (success == False):
        print("reading failed")

        cam.release()

        cam = cv2.VideoCapture(video_name)

        continue

    background = cv2.addWeighted(background, 1 - rho, frame, rho, 0)

    #cv2.imshow("background", background)

    frame_hsv  = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)
    backgr_hsv = cv2.cvtColor(background, cv2.COLOR_RGB2HSV)
```

```
    absdiff = cv2.absdiff(cv2.blur(frame_hsv[:, :, 2], (5, 5)),
                backgr_hsv[:, :, 2])

    #cv2.imshow("absdiff", absdiff)

    _, foreground_mask = cv2.threshold(absdiff, 20, 255,
                            cv2.THRESH_BINARY)

    #cv2.imshow("foreground", foreground_mask)

    ker = np.ones((morp_k_sz, morp_k_sz), np.uint8)
    opened = cv2.morphologyEx(foreground_mask, cv2.MORPH_OPEN, ker)
    closing_ker = np.ones((morp_k_sz + 20, morp_k_sz + 20), np.uint8)
    closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE, closing_ker)

    #cv2.imshow("closed", closed)

    output = cv2.connectedComponentsWithStats(closed, 4, cv2.CV_32S)
    num_labels = output[0]
    labels = output[1]
    stats = output[2]

    filtered = copy.deepcopy(closed)

    for i in range(1, num_labels):
        a = stats[i, cv2.CC_STAT_AREA]

        t = stats[i, cv2.CC_STAT_TOP]
        l = stats[i, cv2.CC_STAT_LEFT]
        w = stats[i, cv2.CC_STAT_WIDTH]
        h = stats[i, cv2.CC_STAT_HEIGHT]

        if (a < area_th):
            filtered[np.where(labels == i)] = 0

        else:
            cv2.rectangle(frame, (l, t), (l + w, t + h), (255, 0, 255), 3)

    cv2.imshow("frame", frame)
    cv2.imshow("closed", closed)

```

```
79      key = cv2.waitKey(140) & 0xFF
80
81      if (key == ord('q')):
82          break
83
84      i += 1
85
86  cam.release()
87  cv2.destroyAllWindows()
88  cv2.waitKey(10)
```

3. Нейронные сети



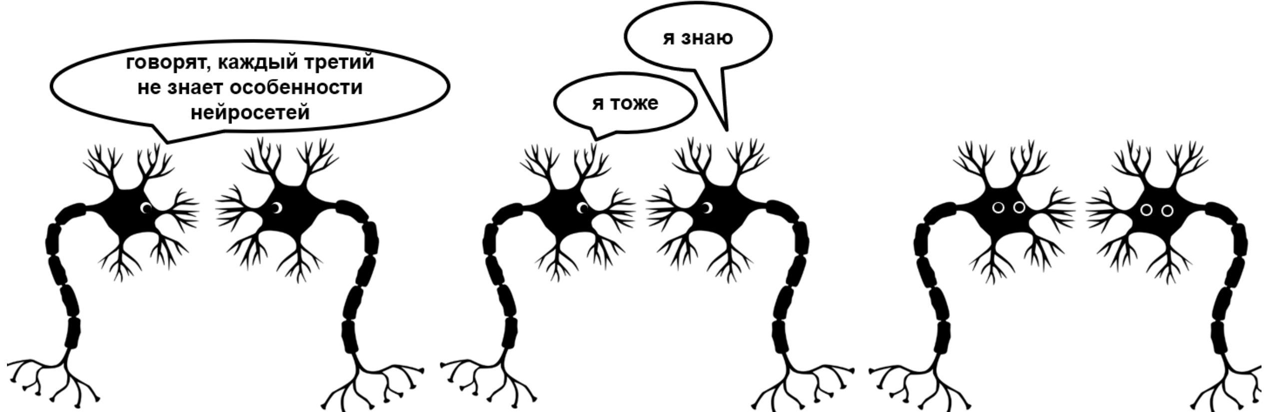


Рис. 3.1: https://temofeev.ru/info/articles/7-let-khaypa-neyrosetey-v-grafikakh-i-vdokhnovlyayushchie-perspektivy-deep-learning-2020-kh/

3.1 Лекция

3.1.1 Задача обучения

Определение

Машинное обучение имеет свои подвиды, чаще всего его делят:

- на обучение с учителем (Supervised learning)
- обучение без учителя (Unsupervised learning)
- обучение с подкреплением (Reinforcement learning)

В рамках этого курса мы будем рассматривать только обучение с учителем.

Обучение с учителем

Определение

Дано:

- X — множество объектов
- Y — множество ответов
- $y : X \to Y$ — неизвестная зависимость

Найти:

$a : X \to Y$, такую, что a будет приближать y на всем множестве X

В первую очередь нам дано X — это множество объектов, Это может быть любой набор данных, который идеологически обладает какими-то общими чертами. Дальше, Y — это множество ответов. И есть такое вот , мы подразумеваем что она есть, это отображение из в Y, это какая-то неизвестная зависимость, которую мы хотели бы узнать. Найти же нам нужно a — это какое-то еще отображение из X в Y, такое что a будет приближать y

на всем множестве X. Почему именно на всем множестве? Потому что для выявления этой зависимости мы не можем использовать абсолютно все множество объектов. Если мы хотим найти мяч на картинке, мы не будем использовать все мячи в мире. Лучше мы возьмем какую-то их часть, желательно наиболее разнообразную, и будем считать, что эта выборка позволит нам определить все признаки, которые отличают мяч от других объектов.

Из обучения с учителем можно выделить некоторые наиболее распространенные задачи.

Первая задача — это *регрессия.* Регрессия фактически аппроксимирует конкретную функцию, и в качестве ответа мы ожидаем значение из какого-то отрезка, например $[0, 1]$.

Вторая задача — это *классификация.* Это определение к какому классу принадлежит тот или иной объект. То есть, если роботу нужно определить, какой конкретно из ваших домашних питомцев, например кошка или собака, перед ним находится, он будет решать именно эту задачу. В классификации ответы модели являются дискретными значениями, в частности ответ 0 может интерпретироваться как «кошка», а 1, как ответ «собака».

Обобщая, обучение с учителем сводится к тому что мы должны определить и описать пары объект-ответ, а также как оценивается качество ответа и какое конкретно качество нас удовлетворяет.

3.1.2 История

История началась в 1943 году с нейрофизиолога Уоррена Маккалоха и математика Уолтера Питтса.

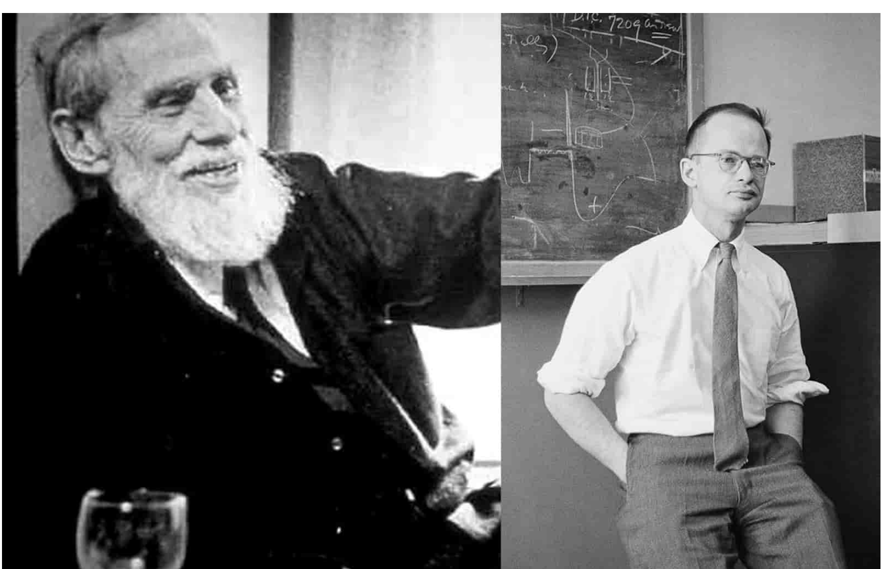


Рис. 3.2: https://labelyourdata.com/articles/history-of-machine-learning-how-did-it-all-start/

Они вместе написали статью о том, как могут работать биологические нейроны, в которой описали математически модель.

Биологический нейрон

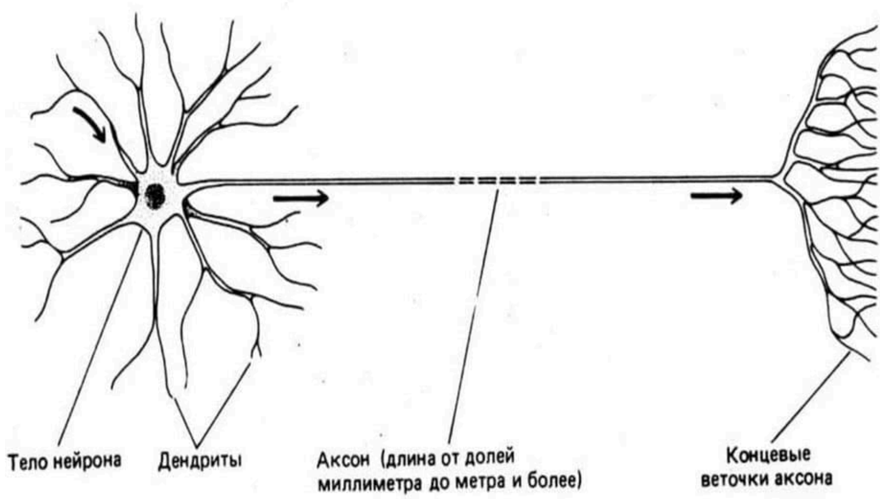


Рис. 3.3: Источник: https://labelyourdata.com/articles/history-of-machine-learning-how-did-it-all-start/

Давайте рассмотрим биологический нейрон. Мы видим, что у нейрона есть большое множество входов, у одной клетки их может быть сотни. Эти входы называют дендриты. И есть выход из этого нейрона — аксон. Естественно, аксон прикрепляется ко множеству нейронов сразу. Особенностью нейрона является то, что из этого множества входов принимаются какие-то сигналы, нейрон их аккумулирует и если суммарный сигнал выше определённого порога, то по аксону идет выходной сигнал. Если сигнал не преодолевает порог, то соответственно он дальше не идет. Еще была замечена важная особенность биологических нейронов: если для какого-то действия или восприятия связи между какими-то конкретными нейронами дают лучший результат, то эти связи укрепляются, и через них проходит больший сигнал. На основе этих данных Маккалох и Питтс предложили следующую модель.

Искусственный нейрон

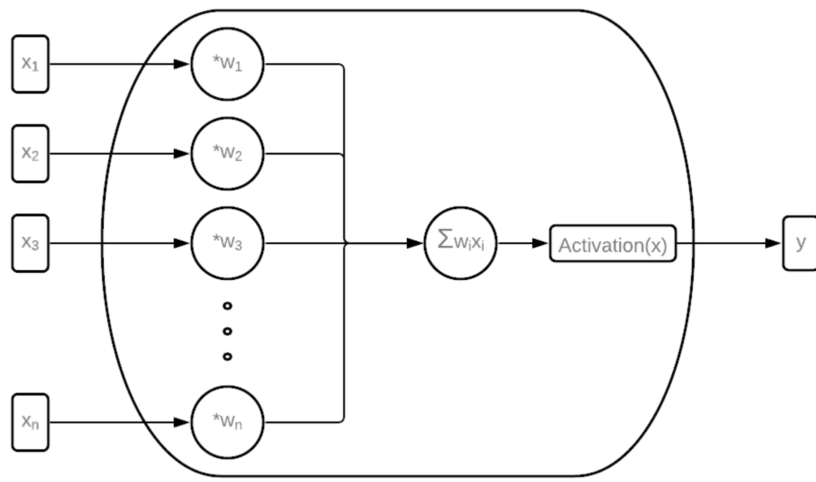


Рис. 3.4: Схематическое представление модели искусственного нейрона

Перед вами первая математическая модель. Здесь иксами помечаются входы в этот искусственный нейрон, у каждого входа есть свой вес, обозначается w. Веса здесь моделируют качество связи между нейронами, позволяют регулировать поток сигнала. В последующем входные сигналы, перемноженные с весами, суммируются внутри нейрона и единым потоком проходят через функцию активации. В данном примере ею может служить пороговая или по-другому ступенчатая функция, где если сигнал преодолел пороговое значение, то генерируется новый сигнал и распространяется дальше.

Персептрон

В 1957 году Фрэнк Розенблатт, реализовав несколько нейронов и связав их между собой, предложил свою реализацию модели искусственной нейронной сети, названную персептроном.

Рис. 3.5: https://blog.neu.ro/blog/deep-learning-and-agi-part-i-computer-vision/

Это была двухслойная сеть, то есть нейроны были попарно связаны между слоями. Можно сказать, это было величайшей разработкой на тот момент. Компьютерное моделирование мозга, которое хоть и могло решать пока что простые задачи, тем не менее потрясало умы! Это развило мощнейший интерес к термину «искусственный интеллект», люди прониклись идеей, что вот-вот и компьютерный мозг, подобно человеческому, будет за нас решать все задачи.

Критика. Минский и Папперт

Рис. 3.6: https://www.datasciencearth.com/bilgisayarli-goru-computer-vision/

Через какое-то время после этого бума в 1969 году Марвин Минский и Сеймур Папперт написали книгу «Персептроны», где критиковали персептрон, потому что та модель, которую реализовал Розенблатт, была сильно ограничена. Модель была линейная и не могла аппроксимировать нелинейности многих задач. Также они писали, что если увеличивать число нейронов или слоев, то мы очень быстро упремся в вычислительный порог, на который компьютеры еще долго не будут способны. Такая критика оттолкнула ученых и финансирование исследований персептрона.

Метод обратного распространения ошибки

Но в 1974 году появился «Метод обратного распространения ошибки». Он был впервые описан Александром Галушкиным и одновременно и независимо Полом Веробосом. Этот алгоритм основан на методе градиентного спуска (SGD), и его целью является минимизация ошибок ответов многослойного персептрона. До этого персептрон мог решать задачи только, если его веса были подстроены вручную. Введение этого метода позволило автоматически настраивать веса, и сделало персептрон гораздо более универсальным. Метод обратного распространения ошибки используют до сих пор, но чаще мы слышим английское его название **Back Propagation**. Он зарекомендовал себя как очень эффективный метод обучения и был распространен и на другие модели нейронных сетей. Интерес к данному направлению снова вырос.

Второй упадок

Тем не менее к 1988 году интерес к нейронным сетям упал. Это связывают с недостаточным развитием вычислительных машин. На тот момент компьютеры не позволяли выполнять такое большое количество операций, может быть, и не очень сложных, таких как суммирование и перемножение. Просто делать их нужно было в огромнейшем количестве. Второй фактор был больше психологическим. Он связан с тем, что в 1957 году появился персептрон и все вдохновились идеей искусственного интеллекта, но по прошествии 30 лет, практически ничего не изменилось. Ученые были очень разочарованы. Это сильно отталкивало, люди думали что, видимо, нейронные сети не стоят внимания.

Перелом во взглядах все же произошел. И на это повлияли следующие факторы.

Закон Мура

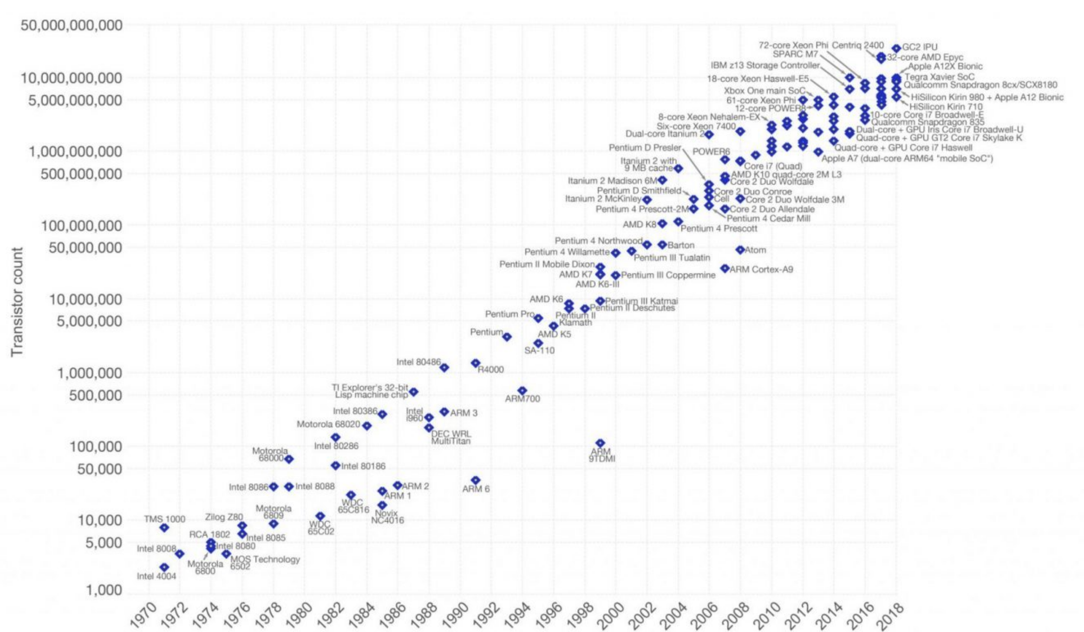


Рис. 3.7: https://lifehacker.ru/texnologicheskaya-singulyarnost/?amp&

Эти мысли крутились в обществе ровно до тех пор, пока вычислительные машины не развились до нужной степени. Все из-за закона Мура, который гласит, что каждые два года число транзисторов в процессорах увеличивается в два раза, т.е. производительность растет экспоненциально. И в принципе это видно по графику. Каждая точка — это новый процессор. Здесь шкала логарифмическая и видно, как точки выстраиваются в прямую.

Данные

Также стоит сказать что параллельно с 1988 года наращивался объем данных. Появилась возможность их всех хранить. Например, операторы сотовой связи получали огромное количество информации, которую хотели потом использовать, по всему миру

люди фотографировали разные объекты, а затем сохраняли изображения в цифровом виде и делились ими в интернете. Но все-таки не было каких-то конкретных методов этот объем данных исследовать и возможности применить.

GPU

В 2010 году впервые нейронная сеть была запущена на графическом процессоре. Этот вычислитель по своей универсальности был немного хуже центрального процессора, но базовые операции выполнял хорошо. Ключевой же особенностью являлась его архитектура. В графическом ускорителе сотни или даже тысячи ядер, каждое из которых существенно слабее такового в центральном процессоре. Тем не менее операции, которые требовалось выполнять во время работы нейросети, были довольно простыми. Параллельные вычисления на огромном количестве ядер значительно ускорили как работу, так и обучение нейронных сетей. Это действительно был прорыв, который дал начало новому направлению под названием «глубокое обучение». Это позволило обуздать нейронные сети с большим количеством слоев, которые тем не менее вычислялись за разумное время, а из-за своей универсальности этот инструмент позволил решать те задачи, где классические методы не справлялись.

Сейчас нейронные сети применяются и в наших компьютерах, и в наших смартфонах. Они оценивают возможную прибыль, дают рекомендательные предложения, заменяют лица на видео и многое другое.

3.1.3 Функции активации нейрона

Поговорим про конкретные примеры, которые довольно часто используются на практике.

Пороговая функция

Давайте рассмотрим пороговую или ступенчатую функцию.

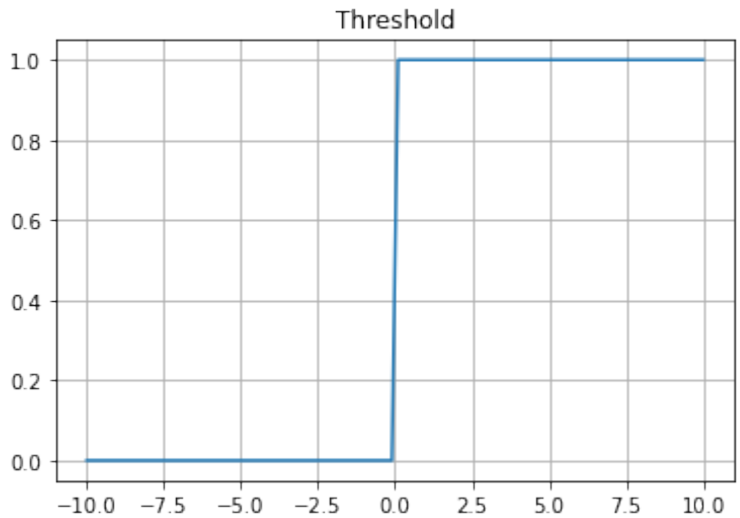


Рис. 3.8: Пороговая функция активации

Ее формула выглядит так:

$$f(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

Она подходит только для задачи классификации. Задачи регрессии требуют ответ из некоторого промежутка, который для обобщения можно свести к $[0, 1]$. В случае если итоговый ответ должен быть, например, в промежутке $[-1, 1]$, выход нейронной сети из промежутка $[0, 1]$ можно преобразовать по формуле $x_{result} = 2x_{out} - 1$. В целом она несет именно непрерывный характер. Естественно с помощью пороговой функции мы что-то непрерывное получить не сможем и для регрессии она не подходит. К тому же в дальнейшем мы расскажем вам про процесс обучения нейронной сети, но сейчас вкратце поясню что нужно, чтобы обучать нейронную сеть. Мы должны прогнать от входов эти значения до выхода и в обучении с учителем должны сравнить этот результат с тем что мы хотим получить и это изменение вернуть обратно. То есть эти коэффициенты внутри надо каким-то образом подстроить и для того, чтобы посмотреть эти изменения нам нужно будет считать производные, в том числе на функциях активации и, к сожалению, для такой пороговой функции производную посчитать очень проблематично. Поэтому для обучения она совсем не подходит. Мы можем вручную подбирать эти значения, но для автоматического обучения она не годится. Давайте рассмотрим другую функцию.

Линейная функция

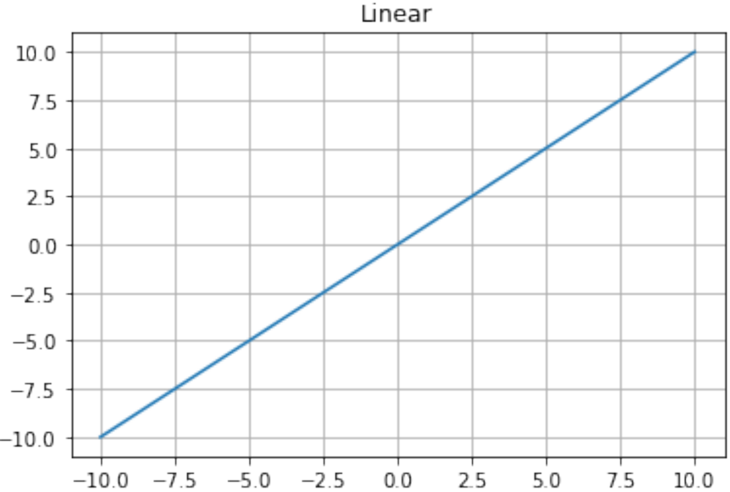


Рис. 3.9: Линейная функция активации

Линейная функция Ее формула выглядит так:

$$f(x) = kx,$$

с точностью до коэффициента. Такая функция отлично подходит и для классификации и для регрессии. Также ее хорошей особенностью является то, что ее производную легко

считать. Но так как мы пытаемся аппроксимировать функцию создавая суперпозиции нейронов, т.е. мы эти нейроны объединяем между собой получая какие-то новые функции. Из-за того, что получаются естественно суперпозиции линейных функций, мы не можем получить какую-то нелинейность. В этом плане линейная функция активации сильно ограничивает наши возможности. Еще нужно отметить, что производная функции активации очень сильно влияет на то, как происходит процесс обучения. Здесь производная не зависит от уровня сигнала, т.е. будет сигнал больше или меньше, производная всегда константа. Это такая особенность линейной функции, которая может сыграть как в плюс, так и в минус. И повторюсь, главная проблема линейной функции в качестве функции активации — мы не сможем аппроксимировать нелинейную.

Сигмоида и гиперболический тангенс

Для этого придумали сигмоиду.

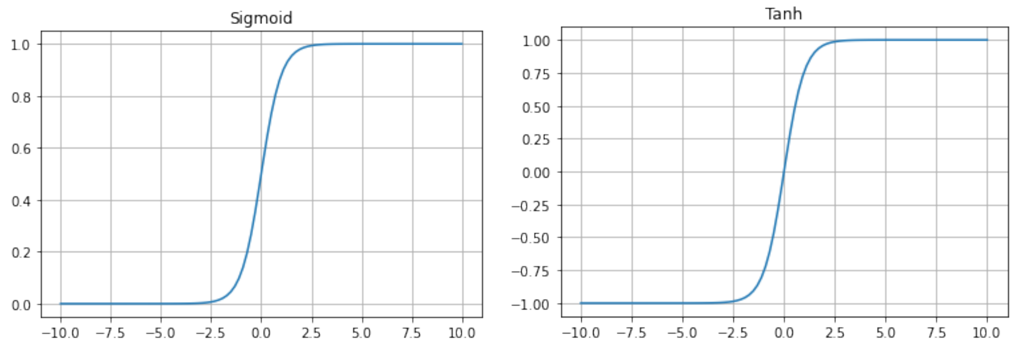


Рис. 3.10: Сигмоида (слева) и гиперболический тангенс (справа)

$$(f(x) = \frac{1}{1 + e^{-x}}$$

Она имеет вид: единица деленная на единицу плюс e в степени минус x. Такая функция активации очень хорошо подходит для классификации и регрессии и довольно часто используется. Вычисление производной для этой функции достаточно дорогой процесс. К счастью производную можно выразить через саму функцию. Это значительно ускоряет вычисления, что делает использование такой функции возможной. Также стоит сказать, что производная здесь будет зависеть от сигнала, что решает одну из проблем линейной функции. Еще один большой плюс такая функция уменьшает сильный сигнал и увеличивает слабый. Так как экспонента строго положительна, то функция активации ограничена 0 снизу и 1 сверху. Не всегда бывает удобно пользоваться функцией у которой область значений строго положительна. Поэтому придумали использовать гиперболический тангенс.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Все свойства и даже график очень похожи с сигмоидой, кроме границ. В данном случае функция ограничена от -1 до 1.

ReLu

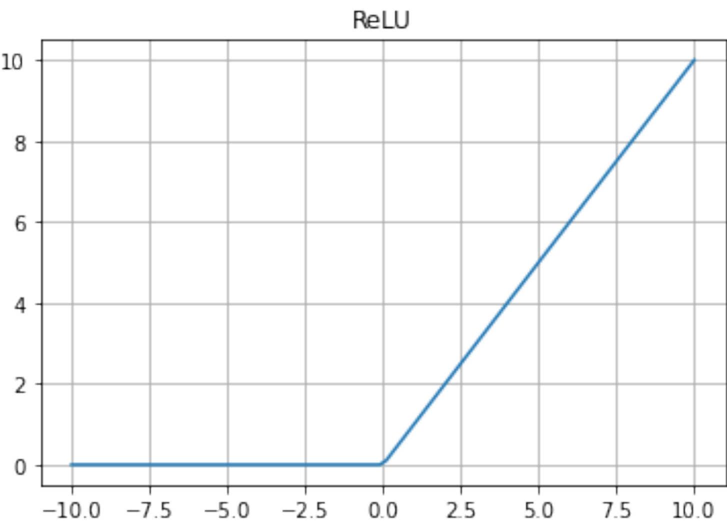


Рис. 3.11: Функция активации ReLU

Была придумана функция под названием ReLU. Она строится таким образом:

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

Видно, что эта функция уже не линейна. Она точно так же универсальна, как предыдущие: может использоваться для классификации, для регрессии, и производная и легко считается: она в одном случае константа, а другом просто 0. Другими словами таких функций можно напихать много и это особенно не добавит вычислительной сложности. Комбинацией таких функций можно аппроксимировать в принципе любую непрерывную функцию, и это очень приятный бонус, но естественно не без проблем. У ReLU есть особенность: она называется dying relu. При неположительном сигнале видим, что производная равна нулю, то есть нейрон просто отключается, умирает. Он не умирает, конечно в прямом смысле, но не участвует в обучении, потому что при нулевой производной изменение весов не распространяется дальше к началу сети. Нейроны не должны влиять на эту связь и поэтому некоторые нейроны остаются пассивны к обучению, что в итоге не очень хорошо влияет на обучение. Из-за того что функция такая легкая и в целом очень универсальна, функции этого семейства часто используются для решения задач.

Leaky ReLu

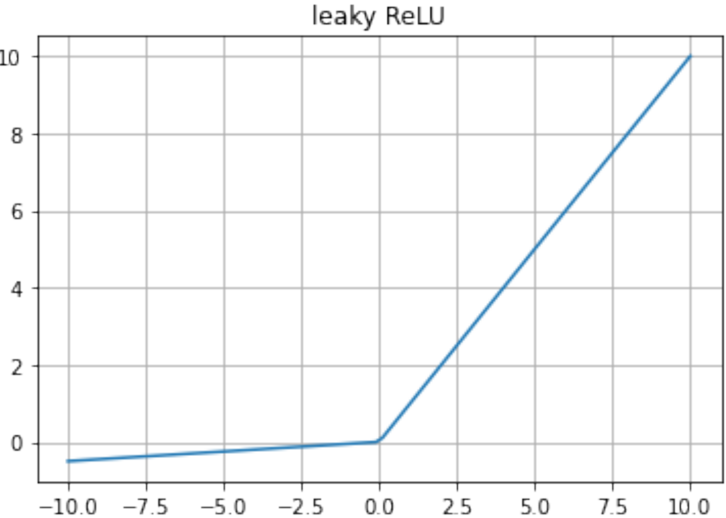


Рис. 3.12: Функция активации Leaky ReLU

Решение проблемы умирающей ReLu тоже было найдено с помощью формирования новой функции под названием Leaky ReLu.

$$f(x) = \begin{cases} x & (x > 0) \\ 0,01x & (x \leq 0) \end{cases}$$

Мы всего лишь добавляем небольшой наклон в отрицательную часть функции. Теперь функция будет задаваться двумя коэффициентами, для левой и правой ветви отдельно. Коэффициент отрицательной части выбирают небольшим, он может быть 0,1 или 0,01 в зависимости от того, как сильно мы хотим влиять на веса нейрона при отрицательных сигналах, что предотвращает застаивание.

3.1.4 Связи нейронов

Нужно сказать, что мы будем формировать из нейронов целые слои, из слоев — сеть. Давайте уточним важный пункт про веса и связи. Как вы помните, концепцию весов придумали, опираясь на биологические нейроны. Те связи, которые больше участвовали в формировании правильного ответа на конкретное действие, укреплялись. В математической модели укрепление связи достигается путем увеличения коэффициента связи, то есть весом w. Когда мы говорим про веса какого-то слоя, то подразумеваем веса между нейронами текущего и предыдущего слоев.

Стоит сказать, что есть и обратные связи. То есть мы можем добавить к входным данным дополнительный вход, который будет формироваться из выхода сети, с каким-то коэффициентом. Обратные связи работают в задачах, где нам необходимо помнить о том, что было в предыдущей момент времени, чтобы сформировать ответ. Примеры

использования можно найти в анализе звука, а также распознавании текстов. В этих задачах применяются рекуррентные нейронные сети, которые не рассматриваются в рамках этого курса.

3.1.5 Слои. Выходы

Полносвязный слой

Давайте теперь поговорим про слои. Перед вами полносвязный слой. Он так называется, потому что нейроны, идущие позади, связываются с каждым нейроном этого слоя, т.е. каждый из них влияет на каждый. Почему же они так хороши? Известно, что один нейрон линейно разделяет какую-то область, это наглядно продемонстрировано в рамках этой недели. Двухслойная сеть (с одним внутренним слоем) ограничивает выпуклую многогранную область, что также продемонстрировано на примере задачи XOR. Трехслойная сеть (с двумя внутренними слоями), может ограничивать многогранную область, причем необязательно выпуклую и необязательно связную, т.е. в принципе такими сетями мы можем разграничивать любые области. Поэтому нейронные сети нам так интересны. Настолько они универсальный инструмент.

Многоканальный выход

Стоит также сказать, что на самом деле мы можем иметь не один выход в нейронной сети. Количество выходов зависит только от количества нейронов в выходном слое. Здесь вы можете видеть пример с двумя выходами. Зачем это нужно? Если мы хотим решить задачу не бинарной классификации, когда у нас всего два класса, а задачу, где классов много, или аппроксимировать вектор-функцию. Применение таких моделей вы также увидите в рамках нашего курса.

3.1.6 Теорема Цыбенко

Возвращаясь к универсальности нейронных сетей, в 1989 году была доказана теорема Цыбенко, которая звучит так.

Теорема Цыбенко (1989)

Если $\sigma(\vec{x})$ - непрерывная сигмоида, то для любой непрерывной на $[0,1]^n$ функции $f(\vec{x})$ существуют такие значения параметров $\vec{w_j} \in \mathbb{R}^n, bias_j \in \mathbb{R}$, что двухслойная сеть

$$a(x) = \sum_{j=1}^{n} \alpha_j \sigma(\vec{w_j}^T \vec{x} + bias_j)$$

равномерно приближает $f(\vec{x})$ с любой точностью ε или

$$|a(\vec{x}) - f(\vec{x})| < \varepsilon, \text{ для всех } \vec{x} \in [0,1]^n$$

Нам теоретически достаточно всего двух слоев и одной нелинейной функции активации для того, чтобы приблизить **любую** непрерывную функцию на отрезке [0, 1] с **любой** точностью… Это очень сильное утверждение, которое расценивается так, что с помощью нейронных сетей можно решить очень большой спектр задач.

3.1.7 Набор данных

Итак, задача обучения нейронной сети состоит в том, чтобы построить параметрическую функцию, отображающую множество входов во множество выходов.

Справа приведены изображения цифр из датасета для классификации, которым мы займемся позже на семинаре. Надо по изображению понять, какая на нем цифра.

В случае задачи детекции, как слева на слайде, задача сети состоит в том, чтобы отобразить картинки в множество прямоугольников, которые ограничивают объекты, находящиеся на изображении. Каждому такому прямоугольнику, или bounding box, сопоставляется его класс, т.е. велосипед, собака, машина.

Если данных для обучения недостаточно или если хочется сделать сеть устойчивой к некоторым преобразованиям, можно применить к элементам датасета эти самые преобразования. Например, повороты, как здесь, или изменение размера, чтобы нейросеть научилась игнорировать эти преобразования. Такая предобработка называется аугментацией. В Гарри Поттере было такое заклинание — «Агуаменти», просто добавь воды, т.е. можно рассматривать аугментацию как своего рода долив воды. Мы не добавляем никаких новых данных, не снимаем новые написанные цифры, не размечаем новых собак, но при этом мы делаем датасет таким, что цифры в нем представлены под разными углами, таким образом обучая сеть игнорировать повороты.

3.1.8 Функция потерь

Предположим, что нейросеть получила на вход картинку, ну или другой объект, который она принимает на вход, обработала его и получила некоторый выход. Чтобы запустить процесс обучения, нужно узнать, насколько правильный этот ответ. Чтобы это узнать, мы должны сравнить его с правильным ответом, который нас уже есть. Для задач детекции правильные ответы — это прямоугольники вокруг объектов, которые на изображениях разметил человек.

В зависимости от того, какая задача решается, так называемые функции потерь или лосс-функции могут быть разными. Если решается задача восстановления изображения или детекции, можно вводить на картинках или на точках bounding box первый или второй лосс из тех, что представлены на слайде, т.е. линейный или квадратичный. Квадратичный лосс хорош тем, что при увеличении отличия ответа, который дает модель, от правильного ответа в два раза, его значение для соответствующего элемента увеличивается в четыре раза. Здесь и дальше игрек с шапочкой - это правильный ответ, а игрек просто - это ответ, который дает модель.

Для задач классификации, то есть сопоставления изображению цифры 0, 1, 2, 3 и т.д., может использоваться так называемая кросс-энтропия. Это лосс, формула которого написана снизу. Он равен сумме по всем классам от минус вот такого выражения в скобках. Таким образом сети, которые занимаются классификацией, отдают на выход вектор вероятностей того, что объект, поданный на вход, принадлежит к какому-то конкретному классу. Правильный ответ выглядит как вектор, где у правильного класса стоит единичка, а все остальные нули.

!!!!!!!!Стоит добавить пример

В принципе обучение нейросети состоит из последовательной модификации весов, с которыми значения, пришедшие на вход, складываются. Об этом было в предыдущей лекции, сейчас мы чуть подробнее на это посмотрим. Веса обучаются за счет того, что ошибка, вычисленная в виде значения функции потерь, распространяется от конца сети к началу. Функция потерь — это на самом деле просто многопараметрическая функция от всех весов и значений на входе.

Слайд «Функция потерь2» Рассмотрим пример вычисления функции потерь для такой вот простой модели. Допустим, что будучи обученной правильно, сеть берет вход, умножает его на два и получает выход. А инициализированная случайными весами сеть умножает на три. Игрек с шапочкой — это правильный ответ, который мы знаем, он лежит в разметке, а игрек — это тот ответ, который на данный момент дает сеть. В таком случае лосс будет численно равен 16

Посмотрим на пример MSE. Она через y, который вывела модель, включает в себя все веса модели. И нам нужно взять и получить производную функции потерь по всем параметрам модели для текущего входа.

3.1.9 Обратное распространение ошибки

Можно сделать это, используя правило дифференцирования сложных функций. То есть мы сначала продифференцируем z — значение лосса — по всем параметрам последнего слоя, а потом с их помощью получим значения градиентов для слоя перед ним. Иначе говоря, мы сначала узнаём лосс, потом вычисляем градиенты на последнем слое, потом на предпоследнем и так далее с использованием так называемого chain rule, которое в сущности есть просто правило дифференцирования сложной функции. dz по dx0 это dx1 по dx0 на dz по dx1. Здесь веса — это матрицы, а вход — это вектор.

И после того, как мы получили значения градиентов для всех весов, мы можем сделать шаг против градиента, чтобы лосс-функция приблизилась к своему минимуму. На каждом конкретном объекте из набора данных нужно делать не слишком большой шаг, в противном случае модель рискует слишком сильно подстроиться под какой-то конкретный объект. Таких объектов много, и нужно учитывать не особенности каждого, а статистические закономерности, и шаги, получается, должны быть сравнительно маленькими.

3.1.10 Оптимизатор

Предположим, что пользуясь этим методом мы получили градиенты весов для конкретного объекта из dataset. Может встретиться случай, в котором градиенты направлены практически в обратную сторону по сравнению с предыдущей итерацией. Таким образом, в пространстве параметров наша модель будет осциллировать туда-обратно, лосс уменьшаться не будет и сходиться к минимуму тоже не будем. Можно для того, чтобы решить эту проблему, принять во внимание историю градиентов. Можно поддерживать скользящие среднее или взвешенную сумму градиентов за все время до этого.

И в этом состоит роль так называемого оптимизатора. Оптимизатор занимается тем, что смотрит на историю того, как изменялись градиенты и того, как модель перемещалось в пространстве параметров, и выбирает, в какую сторону нужно подвинуть веса. К

примеру, там может решаться задача одномерной оптимизации вдоль градиента, т.е. на сколько нужно подвинуть веса, чтобы лосс упал на наибольшее значение. Есть разные оптимизаторы, самый простой из них состоит в отсутствии оптимизатора — градиент с минусом домножается на некоторую небольшую константу, и веса модифицируются таким образом.

Есть достаточно сложные оптимизаторы, которые в свое время очень сильно повлияли на обучение, сделали его гораздо быстрее. Один из самых популярных, оптимизатор Адам, был изобретен совсем недавно — в 2015 году, и он по скорости сходимости значительно превосходит обычный градиентный спуск.

3.1.11 Эпоха

Эпоха — это полный проход по обучающей выборке, когда для каждого объекта из него производится прямое и обратное распространение. Для того, чтобы внутри батча не оказывались все время одни и те же объекты, батчи на каждой эпохе формируются случайным образом. Количество эпох, требуемое для обучения, зависит от конкретной задачи, в жизни обычно это величины порядка сотни. Но можно найти случаи, в которых сходимость достигается за считанные эпохи, и можно представить себе систему, в которой сеть постоянно дообучается, так что формально говоря количество эпох для нее не ограничено

3.1.12 Learning rate

Learning rate — это параметр, который уже упоминался ранее, это ограничение на максимальный шаг изменения весов. Оно бывает как правило порядка одной 0.1 — 0.001, т.е. он не очень большой. Это нужно для того, чтобы за одно обновление весов сеть не изменилась очень сильно. Learning rate тоже зависит от задач, и если он очень маленький, то вероятность того, что модель куда-то случайно ускачет, низкая, но при этом и сходимость медленная. А если большой, то можно достаточно быстро дойти до окрестности минимума, а потом вокруг нее ходить и в неё совсем не спускаться. Довольно часто при обучении используется изменение learning rate после некоторой эпохи или при выполнении некоторого условия, например если loss на обучающей выборке не уменьшался уже пять эпох, learning rate можно уменьшить. В таком случае сеть начнет сходиться медленнее, но к более оптимальному состоянию, что и требуется, если мы считаем, что с большим learning rate уже оказались в окрестности оптимума.

3.1.13 Цикл обучения

Просуммируем, то что мы до этого рассмотрели в train loop, или цикле обучения. Сначала мы получаем из набора данных пару «вход и ответ». Потом подаем вход в модель, получаем из нее выход, потом считаем значение функции потерь от того, что вернула модель и правильных ответов. Делаем проход обратного распространения и модифицируем веса сети. Повторим это в цикле. Одна эпоха — это один проход по всему набору данных, и внешний цикл реализует такой проход. Соответствие практически по строчкам. Input и target — это формирование пар «икс игрек». Дальше есть строка

optimizer.zero_grad. Это зануление градиентов, которые хранятся в оптимизаторе. Если этого не сделать, то оптимизатор будет думать, что у нас продолжается длинный-длинный батч и мы продолжаем накапливать градиенты. Output = net(input) — это получение ответа модели, дальше подсчет лосса, обратный проход и шаг оптимизатора. В этом случае размер батча равен единице.

3.2 Семинар

Для работы с семинаром, а также для выполнения практической части понадобится Python3 с библиотеками torch (PyTorch), matplotlib.

3.2.1 Работа с тензорами в torch

Здесь мы с вами познакомимся с фреймворком PyTorch, а также поговорим про тензоры, как их создавать и какие операции можно к ним применять. Для начала требуется установить библиотеку torch на свое устройство. Импортировать ее для дальнейшего использования внутри кода нужно через следующую команду.

```
1 import torch
```

Создание тензоров

В PyTorch тензор можно создать несколькими способами. Первый — это метод **tensor()**, на вход которого мы подаем полный список из нужных нам элементов.

```
1 a = torch.tensor([[1., -1.], [1., -1.]])
```

Далее можно создать тензор из нулевых элементов. Для этого применим метод **zeros()**. На вход они принимают список из размеров будущего тензора по осям. В этом примере видно, что по нулевой оси тензор будет иметь размер 2, а по первой — размер 4.

```
1 b = torch.zeros([2,4])
```

Функция **ones()** работает аналогичным образом за исключением того, что итоговый тензор состоит из единиц. Размер можно указывать не только через список, используя [], но и напрямую.

```
1 c = torch.ones(1,2,3)
```

Операции с тензорами можно производить на разных устройствах: на Центральном процессоре (CPU), на видеокартах (GPU) или на специальных тензорных процессорах (TPU). Здесь вы можете увидеть, каким образом можно указать конкретное устройство для вычислений. Метод **torch.cuda.is_available()** позволяет узнать, есть ли в системе хотя бы одна cuda видеокарта. Если есть, то в качестве вычислителя выбирается, как

в примере, первая видеокарта, если же она недоступна, то выбирается центральный процессор.

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

d = torch.ones([2, 4], dtype=torch.float64, device=device)
```

В примере создается тензор из единиц, который хранится в памяти (оперативной или видеопамяти) устройства, инициализированного выше, со специализированным типом. Здесь конкретно указано, что элементы хранятся в виде float64. Можно таким же образом определять и другие типы хранящихся данных.

Операции над тензорами в torch

Инициализируем 2 тензора, на примере которых мы будем все рассматривать.

```
x = torch.tensor([[1., 2.], [3., 4.]])
y = torch.tensor([[1., -1.], [-1., 1.]])
```

Сначала вызовем метод sum, который позволяет найти сумму элементов тензора. Как видно, результат возвращается тоже в виде тензора. Здесь и во всех следующих примерах метод print() и текст будут приведены для справки.

```
print('Sum of elements in x:\n', x.sum())
```

Метод — item, дает возможность получить данные, содержащиеся в этом тензоре. Вот пример:

```
print('Sum of elements in y:\n', y.sum().item())
```

Функция torch.add позволяет реализовать поэлементное сложение тензоров. Естественно, тензоры должны быть совместимы по размеру. Иначе будет ошибка.

```
print('Adding one tensor to another:\n', torch.add(x,y))
```

mul() аналогично производит поэлементное умножение двух тензоров.

```
print('Multiplying one tensor with the other one:\n', torch.mul(x,y))
```

В случае когда мы хотим реализовать матричное умножение в PyTorch, это можно сделать с помощью метода **matmul()**.

```
input_shape = 3
weights = torch.rand([input_shape])
```

```
print('weights', weights)

inp = torch.tensor([1., -1., 1.]).t()
print('input', inp)

torch.matmul(weights, inp)
```

Естественно, при перемножении двух векторов, один из них требуется транспонировать, и для этого подходит метод **t()**, который есть у каждого тензора. Здесь также приведен еще один пример создания тензора **rand()**,который позволяет заполнить тензор случайными значениями из отрезка [0, 1]

Метод **cat()**, что есть сокращение от английского concatenate, объединяет два тензора по заданной оси.

```
torch.cat((weights, torch.tensor([0.5])), axis=0)
```

3.2.2 Реализация модели нейрона

Перейдем к модели нейрона.

В данную модель входят: входные сигналы, веса. Мы видим, что, например, один сигнал проходя через нейрон перемножается на определенный вес, дальше все сигналы суммируются, проходят активацию, и мы получаем результат.

Функция активации нейрона

Давайте определим функцию активации. В биологических нейронах сигнал должен пройти через какой-то порог (threshold) активировации, чтобы мы могли передать сигнал далее. Давайте зададим функцию активации таким образом, что при наличии сигнала результат будет единицей, если его нет, то ноль. Это функция Хевисайда, которая записывается такой вот системой.

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

Напишем ее на Python, но расширим так, чтобы порог мы могли задавать самостоятельно.

```
def threshold_function(x, threshold=0):
    return 0 if x < threshold else 1
```

Давайте визуализируем наш результат. Для отрисовки графиков будем использовать библиотеку matplotlib, которая должна быть заранее установлена на ваше устройство. Метод plot() требует на вход 2 списка данных, по-сути это значения по оси X и оси Y. Сгенерируем данные по оси X и запишем их в inp. Для генерации данных по оси Y применим к каждому значению из inp нашу пороговую функцию.

```
import matplotlib.pyplot as plt

inp = [x*0.01 - 0.5 for x in range(100)]
plt.plot(inp, [threshold_function(x, threshold=0) for x in inp])
```

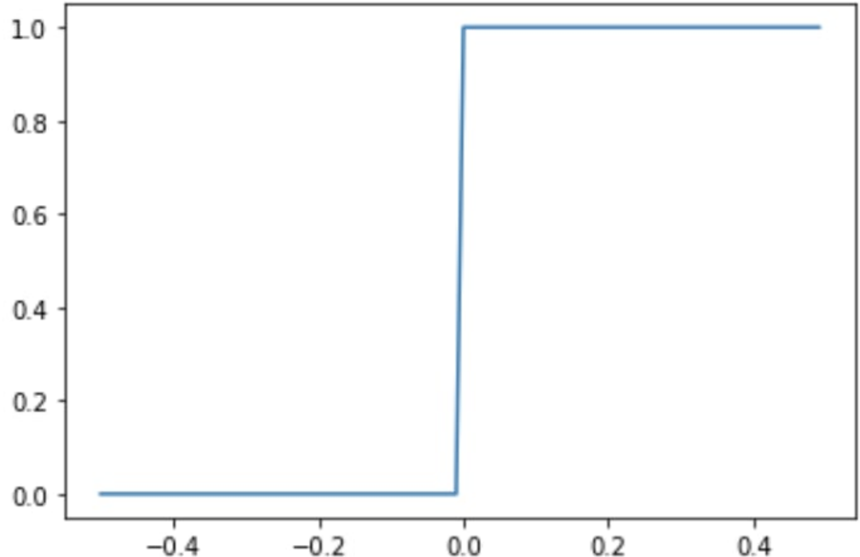


Рис. 3.13: Пороговая функция (функция Хевисайда)

На графике видим, что, как только сигнал перешел ноль, значение функции стало единицей, получилась ступенька.

Класс Neuron

Давайте рассмотрим модель нейрона, реализованную с помощью PyTorch в виде класса.

```
class Neuron(object):
    def __init__(self,
                 input_shape=None,
                 weights: torch.Tensor=None,
                 activation_function=None,
                 debug=False):

        assert input_shape is not None or weights is not None
        if weights is not None:
            self.weights = weights
        else:
            self.weights = torch.rand([input_shape, 1])
```

```
        self.activation_function = activation_function
        self.debug = debug

    def agregate_signal(self, input_tensor):
        if self.debug:
            print('Input_tensor:\n', input_tensor)
            print(f'Multiplying:{self.weights} * {input_tensor}')
        m = torch.mul(self.weights, input_tensor)
        if self.debug:
            print(f'Result:\n{m}')
        s = m.sum()
        if self.debug:
            print(f'Sum:\n{s}')
        output = s.item()
        if self.debug:
            print('Output without activation:\n', output)
        return output

    def activation(self, input_value):
        if self.activation_function is not None:
            output = self.activation_function(input_value)
        else:
            output = input_value
        if self.debug:
            print('Output after activation:\n', output)
        return output

    def forward(self, input_tensor):
        output = self.agregate_signal(input_tensor)
        output = self.activation(output)
        return output
```

При его инициализации (создании) выполняется команда assert, с помощью которой мы можем проверить, задан ли нам входной размер векторов или нам уже даны готовые веса. Если веса у нас уже есть и они обучены, то класс помещает их в данный нейрон, и мы получаем готовую модель, способную что-то считать. Поэтому сначала мы требуем, чтобы входной размер или веса были не пустыми, иначе это выдаст ошибку.

Если веса не пустые, то мы загружаем их в нейрон, т.е. во внутреннюю переменную self.weights, иначе эту внутреннюю переменную мы заполняем случайно, как в примере выше (input_shape). Далее задаем внутреннюю функцию активации нейрона, путем передачи этой функции активации извне через параметр activation_function.

Параметр self.debug нужен, чтобы регулировать выводы на экране результатов внут-

ренних процессов.

Далее реализуем 2 основных метода: суммирование (перемножение) и активация. Первый метод — agregate_signal. В результате этого метода мы выводим не тензор, а какое-то конкретное число. Далее метод activation, который получает конкретное значение и применяет функцию активации, которую она получила извне, к этому значению. Если функция активации не задана, то все вернется к исходному значению.

Далее следует метод forward, который заключается в прогонке сигнала от начала нейрона вперед, через него, для получения какого-то результата. Другими словами, данный метод принимает входной тензор, агрегирует сигнал, затем применяет метод активации и возвращает результат.

3.2.3 Ручное обучение нейрона

Давайте решим задачу «Классификации линейно-разделимых классов». Создаем набор данных и видим, что мы можем построить такую прямую, которая отделит данные классы друг от друга.

```
N = 10

C0_x0 = np.random.random(N)
C0_x1 = (C0_x0 + [np.random.randint(10)/10 for i in range(N)] + 0.1)
C0 = np.array([(C0_x0[i], C0_x1[i]) for i in range(len(C0_x0))])

C1_x0 = np.random.random(N)
C1_x1 = (C1_x0 - [np.random.randint(10)/10 for i in range(N)] - 0.1)
C1 = np.array([(C1_x0[i], C1_x1[i]) for i in range(len(C1_x0))])

# plotting
plt.scatter(C0_x0, C0_x1, s=10, c='red')
plt.scatter(C1_x0, C1_x1, s=10, c='blue')
plt.xlim([-0.1,1.1])
plt.ylim([-1.1,2.1])
plt.grid(True)
```

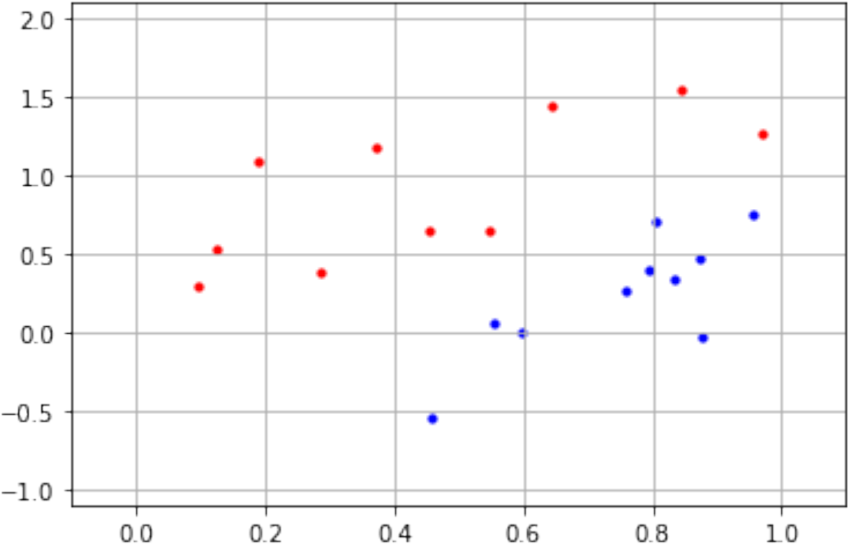


Рис. 3.14: Датасет для задачи линейно-разделимых классов

Решаем эту задачу таким образом, что все значения выше прямой должны оказаться красными, а ниже — синими. Все решение основывается на построении прямой, которая будет разделять эти классы.

$$\begin{cases} w_0x_0 + w_1x_1 \geq 0 & \text{C0 (red)} \\ w_0x_0 + w_1x_1 < 0 & \text{C1 (blue)} \end{cases} \tag{3.1}$$

Мы точно знаем, что она проходит через 0 (частный случай), поэтому в данном случае — это упрощенный вид. Нам необходимо подобрать коэффициент угла наклона. Сейчас мы делаем это вручную.

$$w_0x_0 + w_1x_1 = 0 \tag{3.2}$$

$$x_1 = -w_0/w_1 * x_0 \tag{3.3}$$

$$k = -w_0/w_1 \tag{3.4}$$

В коде опишем то же самое и построим прямую, подобрав коэффициенты так, чтобы она разделила наши классы.

```
w0 = -1
w1 = 1

wf_x0 = [-1, 0 , 1, 2]
wf_x1 = [x0*(-w0/w1) for x0 in wf_x0]

# plotting
```

```
plt.plot(wf_x0, wf_x1)
plt.grid(True)
plt.xlim([-0.1,1.1])
plt.ylim([-1.1,2.1])
plt.scatter(C0_x0, C0_x1, s=10, c='red')
plt.scatter(C1_x0, C1_x1, s=10, c='blue')
```

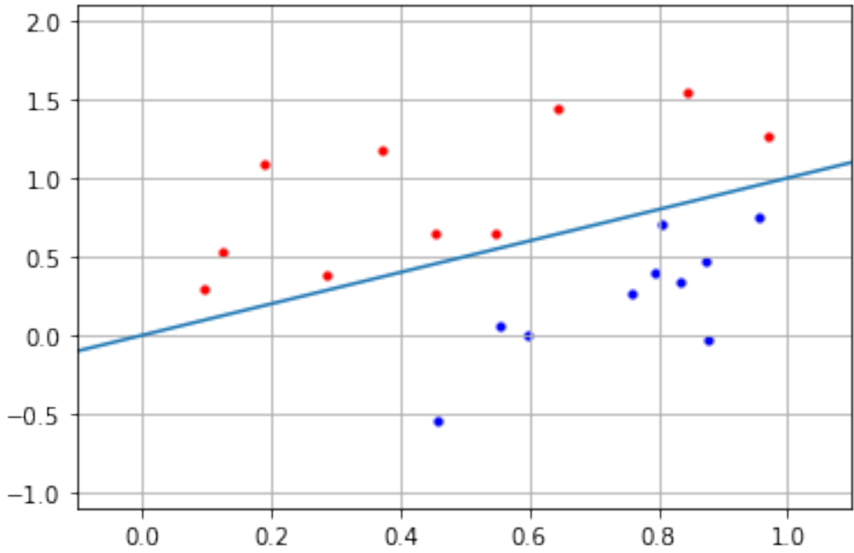


Рис. 3.15: Прямая, подобранная для разделения двух классов

Коэффициенты фактически и являются весами, а x0 и x1 — это входные сигналы и видно, что данная прямая фактически является нейроном с 2 входами. Таким образом, нейрон решает нашу задачу классификации линейно-разделимых классов построением прямой. В общем случае нейрон строит гиперплоскости. Посмотрим, как нейрон будет оценивать точки (относительно этой прямой), которых не было в первоначальной выборке.

```
weights = torch.tensor([w0, w1])
nn = Neuron(weights=weights, activation_function=threshold_function, debug=True)

x = [1, -1]

out = nn.forward(torch.tensor(x))

# plotting
if out >= 0.5:
    print("Class C0 (Red)")
```

```
11        plt.scatter(x[0], x[1], s=10, c='red')
12
13    else:
14        print("Class C1 (Blue)")
15        plt.scatter(x[0], x[1], s=10, c='blue')
16
17    plt.plot(wf_x0, wf_x1)
18    plt.grid(True)
19    plt.xlim([-0.1,1.1])
20    plt.ylim([-1.1,2.1])
```

Подставив в качестве весов наши подобранные коэффициенты, мы фактически провели обучение нейрона. Теперь он умеет определять, к какому классу принадлежит та или иная точка.

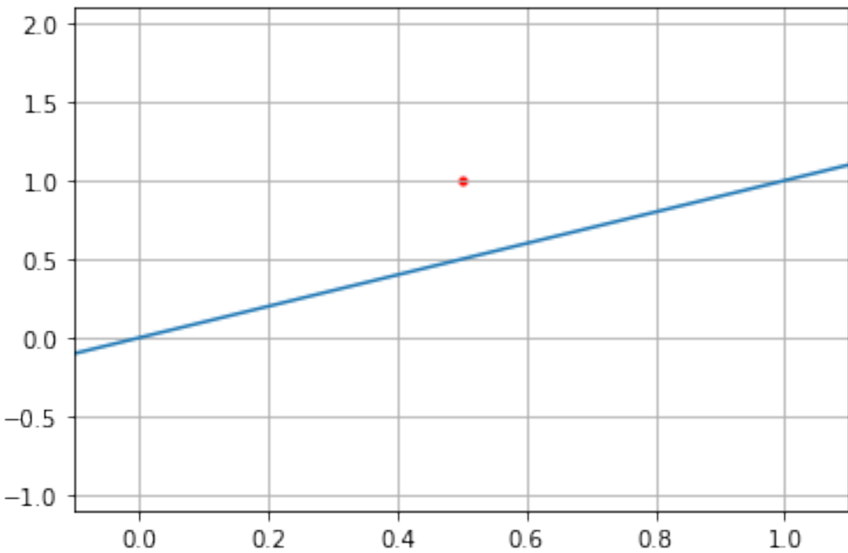


Рис. 3.16: Проверка работоспособности нейрона на точке, не принадлежащей изначальной выборке

3.2.4 Обучение нейрона с байесом

В предыдущей задаче датасет был сгенерирован так, что его можно было разделить прямой, проходящей через начало координат. Конечно, это было упрощение. Реальные данные устроены гораздо сложнее. Давайте усложним задачу, сгенерировав датасет, точки которого заведомо подняты над началом системы координат.

```
1    N = 10
2    b = 2
3
```

```
C0_x0 = np.random.random(N)
C0_x1 = (C0_x0 + [np.random.randint(10)/10 for i in range(N)] + 0.1 + b)
C0 = np.array([(C0_x0[i], C0_x1[i]) for i in range(len(C0_x0))])

C1_x0 = np.random.random(N)
C1_x1 = (C1_x0 - [np.random.randint(10)/10 for i in range(N)] - 0.1 + b)
C1 = np.array([(C1_x0[i], C1_x1[i]) for i in range(len(C1_x0))])

# plotting
plt.scatter(C0_x0, C0_x1, s=10, c='red')
plt.scatter(C1_x0, C1_x1, s=10, c='blue')
plt.xlim([-0.1,1.1])
plt.ylim([-1.1+b,2.1+b])
plt.grid(True)
```

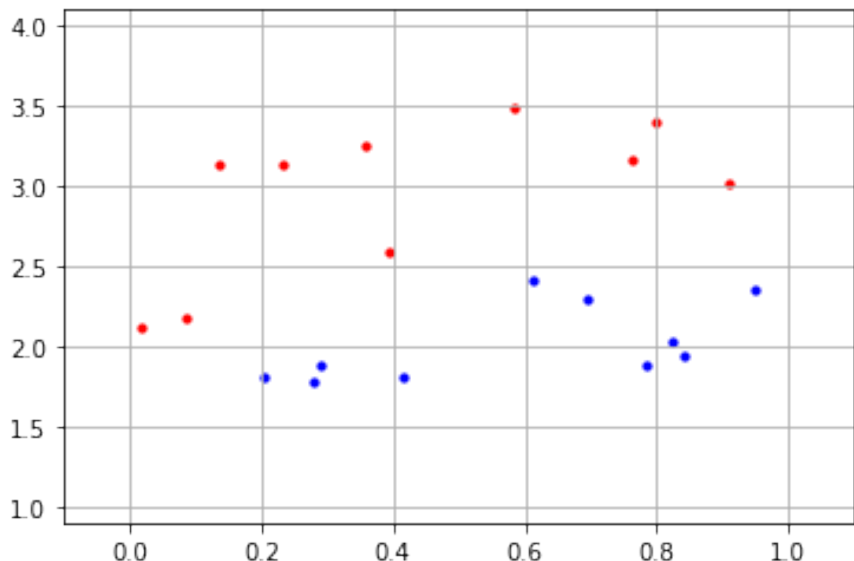


Рис. 3.17: Усложненная выборка линейно-разделимых классов (точки подняты над началом отсчета)

Эти два класса мы никогда не сможем разделить при помощи подбора коэффициентов угла наклона.

```
w0 = -1
w1 = 0.3

wf_x0 = [-1, 0 , 1, 2]
```

```
wf_x1 = [x0*(-w0/w1) for x0 in wf_x0]

# plotting
plt.plot(wf_x0, wf_x1)
plt.grid(True)
plt.xlim([-0.1,1.1])
plt.ylim([-1.1,2.1+b])
plt.scatter(C0_x0, C0_x1, s=10, c='red')
plt.scatter(C1_x0, C1_x1, s=10, c='blue')
```

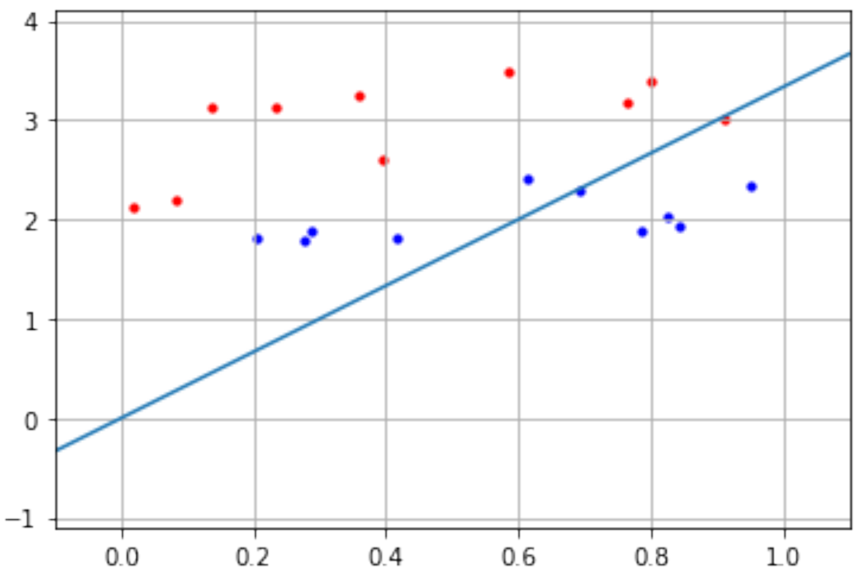


Рис. 3.18: Подбор угла наклона прямой не позволяет решить задачу

Какое есть решение? Нам необходимо поднять данную прямую. Введем свободный член в наше уравнение прямой.

$$\begin{cases} w_0x_0 + w_1x_1 + bias \geq 0 & \text{C0 (red)} \\ w_0x_0 + w_1x_1 + bias < 0 & \text{C1 (blue)} \end{cases} \tag{3.5}$$

$$w_0x_0 + w_1x_1 + bias = 0 \tag{3.6}$$

$$x_1 = -w_0/w_1 * x_0 - bias/w_1 \tag{3.7}$$

$$k = -w_0/w_1, b = -bias/w_1 \tag{3.8}$$

Модель нейрона стала более универсальной. Теперь мы сможем не только менять угол наклона, но и перемещать прямую вверх и вниз. Реализация такой модели будет базироваться на предыдущей и в коде выглядеть так:

```
class BiasNeuron(Neuron):
    def __init__(self, input_shape=None, weights: torch.Tensor=None, bias=None, activati
                     debug=False):
        super().__init__(input_shape, weights, activation_function, debug)

        if bias is not None:
            self.bias = bias


    def forward(self, input_tensor):
        output = self.agregate_signal(input_tensor)

        output = output + self.bias

        output = self.activation(output)
        return output
```

Здесь, по сути, нам требуется уточнить, как использовать новый параметр bias во время инициализации модели и во время прогонки сигнала вперед. В нашем случае мы просто прибавляем смещение к результату суммы входов.

Подберем коэффициенты, которые позволят нам решить эту задачу:

```
w0 = -0.5
w1 = 0.5
b = 2.

bias = -b * w1

wf_x0 = [-1, 0 , 1, 2]
wf_x1 = [x0*(-w0/w1) + b for x0 in wf_x0]

# plotting
plt.plot(wf_x0, wf_x1)
plt.grid(True)
plt.xlim([-0.1,1.1])
plt.ylim([-1.1,2.1+b])
plt.scatter(C0_x0, C0_x1, s=10, c='red')
plt.scatter(C1_x0, C1_x1, s=10, c='blue')
```

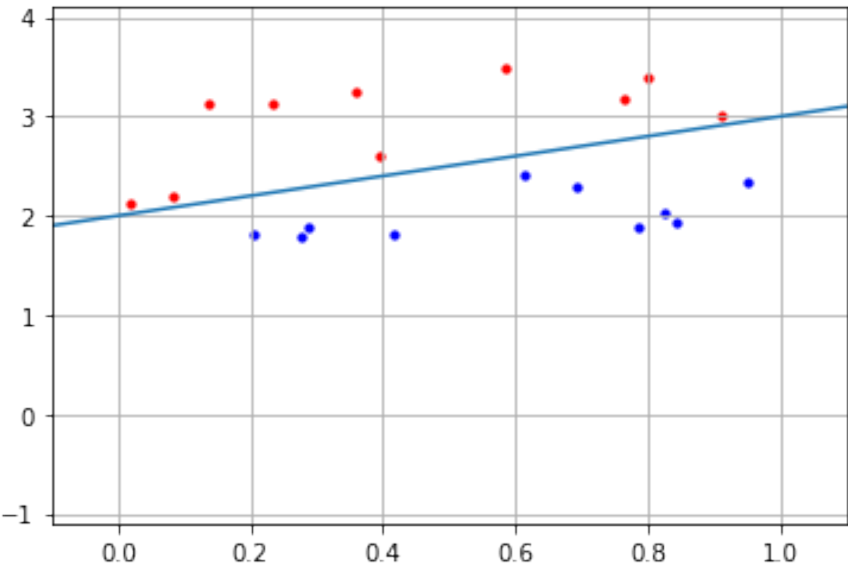


Рис. 3.19: Добавка нового параметра bias (смещение) позволяет решить задачу

Давайте «обучим» наш новый нейрон и посмотрим, как он будет решать задачу на тех данных, на которых не обучался.

```
weights = torch.tensor([w0, w1])
nn = BiasNeuron(weights=weights, bias=bias, debug=True)

x = [1, 3]

out = nn.forward(torch.tensor(x))

# plotting
if out >= 0.5:
    print("Class C0 (Red)")
    plt.scatter(x[0], x[1], s=10, c='red')

else:
    print("Class C1(Blue)")
    plt.scatter(x[0], x[1], s=10, c='blue')

plt.plot(wf_x0, wf_x1)
plt.grid(True)
plt.xlim([-0.1,1.1])
plt.ylim([-1.1,2.1+b])
```

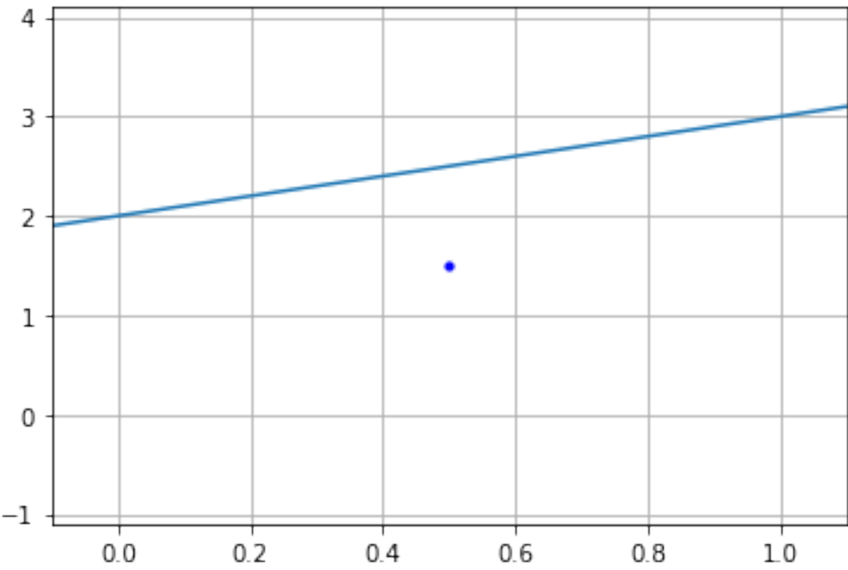


Рис. 3.20: Проверка работоспособности нейрона на точке, не принадлежащей изначальной выборке

3.2.5 Задача «XOR»

Следующая задача, XOR (или исключающее ИЛИ), выглядит так, что две по диагонали противолежащие точки принадлежат одному классу, две другому.

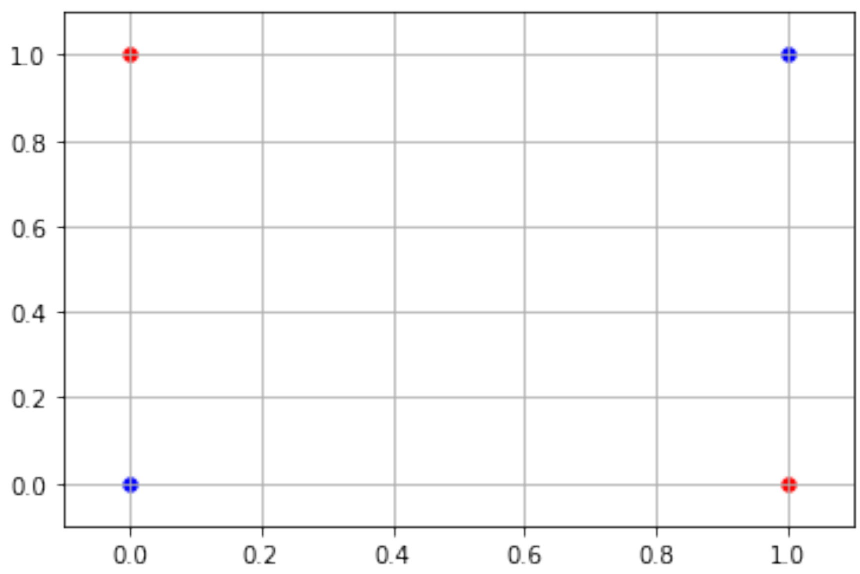


Рис. 3.21: Датасет для задачи «XOR»

Нам точно также необходимо их линейно разделить. Соответственно также все, что выше прямой, будет единичкой, ниже 0.

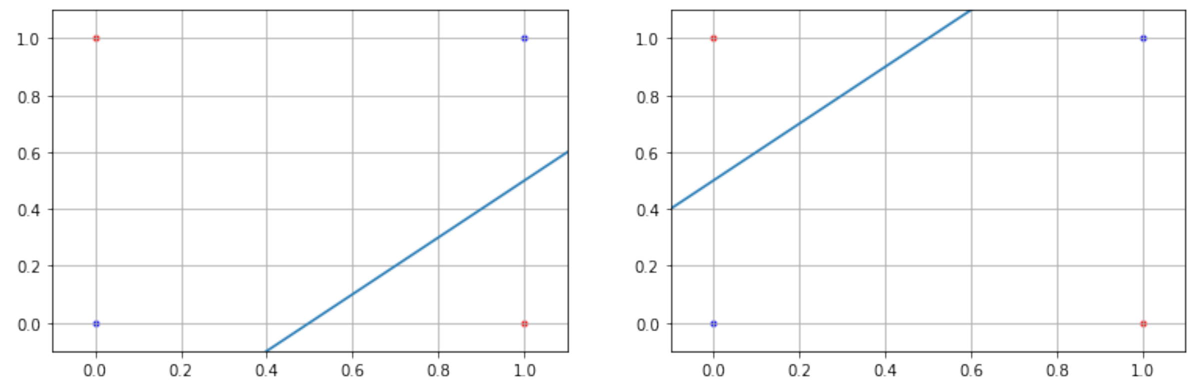


Рис. 3.22: С помощью одной прямой разделить классы невозможно

У нас две картинки с разными результатами. Если мы вычтем из второй картинки первую, то получим нужный нам результат. Придется использовать 2 прямые.

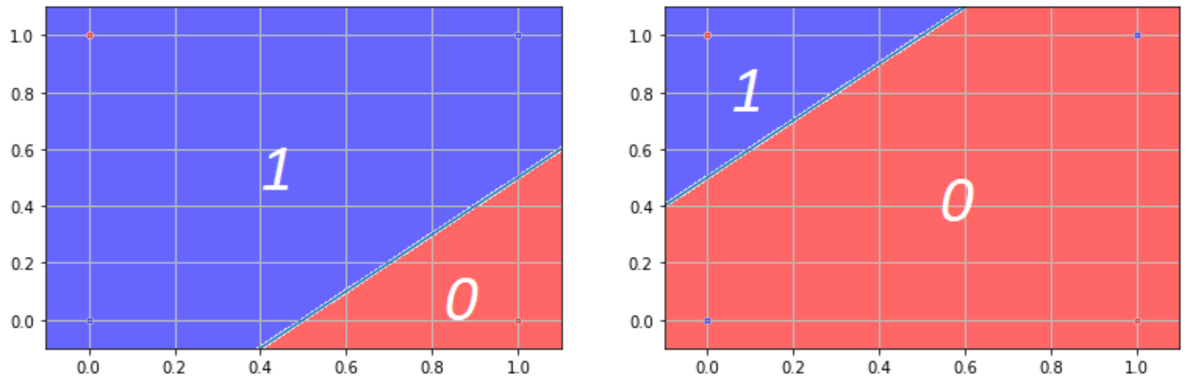


Рис. 3.23: С помощью одной прямой разделить классы невозможно

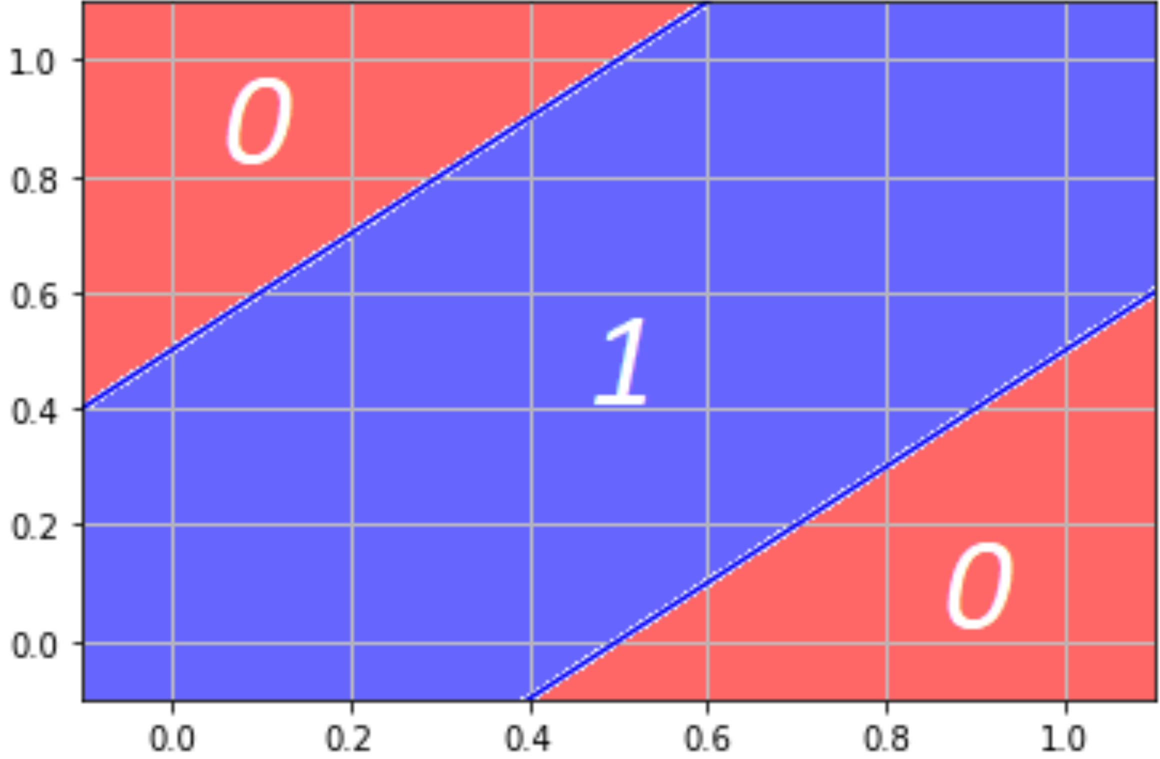


Рис. 3.24: С помощью объединения разделенных областей мы можем получить нужный нам результат

Как с помощью нейрона это реализовать: в первом слое будет 2 нейрона, мы из значений второго нейрона будем вычитать значения первого и получим нужный нам результат. Теперь мы уже работаем с целой сетью (2 нейрона в первом слое, во втором 1), которую также называют персептроном.

Реализуем ее с помощью описания слоев (списка нейронов, которые будут лежать внутри) и функции forward, которая в каждый нейрон помещает входной тензор. Нейроны первого все считают, дальше формулируется ответ, преобразовываем его в тензор и передаем в каждый нейрон из 2-го слоя.

Если во втором слое для первого нейрона сделать коэффициент -1, а для второго 1, то сигналы, пришедшие с первого, будут вычитаться из второго, так как сигнал будет умножаться на -1(будет отрицательным), а затем складываться с положительным сигналом из 2-го нейрона, т.е. проводим ту же самую операцию сложения, только с такими коэффициентами.

Теперь мы можем посмотреть на результат. Изначально у нас было 4 точки для обучения, но сразу видно, что внутренняя область принадлежит одному классу, а внешние — другому.

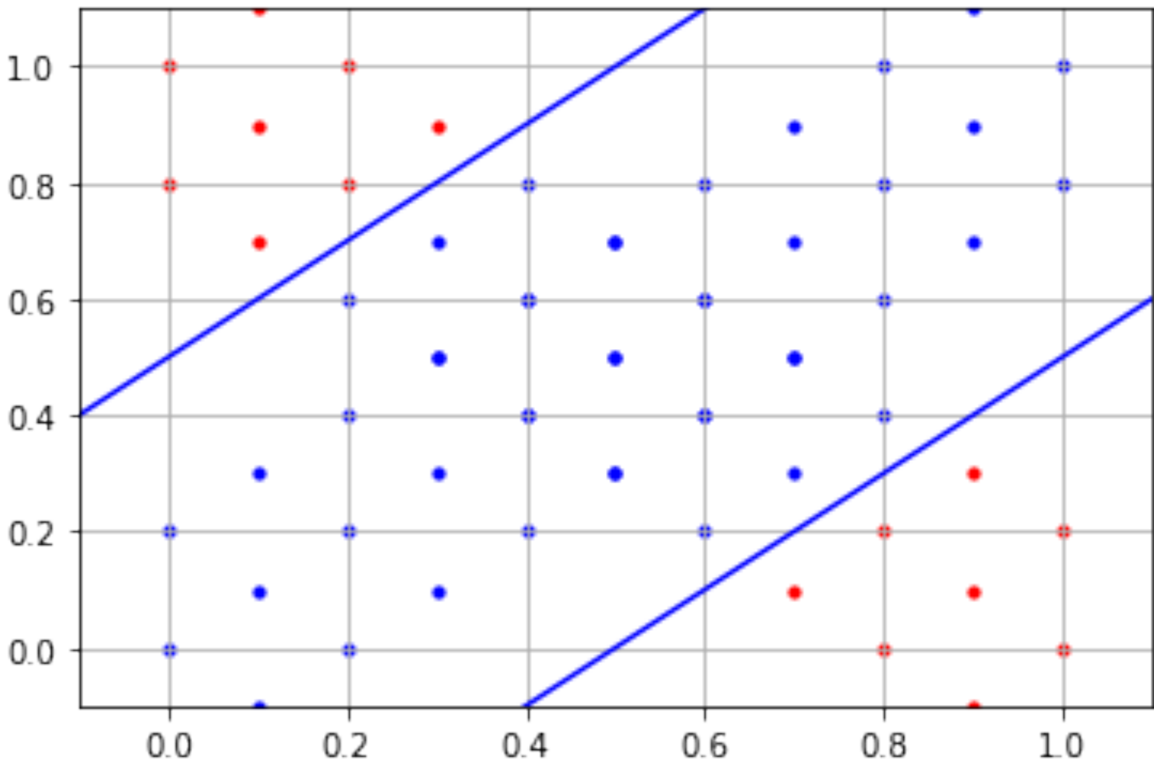


Рис. 3.25: С помощью одной прямой разделить классы невозможно

3.2.6 Задача «Домик»

```

```

Давайте рассмотрим, как происходит автоматическое обучение нейронных сетей. Как изменение параметров влияет на итоговый результат? Делать мы это будем на довольно наглядной задаче под кодовым названием «Домик». Прежде чем мы начнем, давайте импортируем нужные библиотеки:

```
import math
import random

import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
from torch.utils.data import Dataset, DataLoader
```

И выберем устройство, на котором будем производить вычисления:

```
device=torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

На данном компьютере есть видеокарта, поэтому мы будем использовать ее в качестве устройства. Напомню, что метод torch.cuda.is_available() показывает, доступна ли cuda видеокарта. Запись "cuda:0"означает, что мы будем выбирать первую видеокарту: число после двоеточия указывает на id видеокарты внутри операционной системы, и обычно числа идут по порядку от 0. Для этой конкретно задачи отсутствие или наличие видеокарт не оказывает никакого влияния ввиду ее простоты. Но для обучения, например, сверточных сетей — это будет иметь весомое значение, так как станет влиять в первую очередь на скорость обучения: вы затратите меньшее количество времени на ожидание.

Мы с вами хотим научить нейронную сеть распознавать домик. Как он формируется?

```
class DomikDataset(Dataset):
    def __init__(self, dataset_size, noise=0.0):
        super().__init__()
        self.house_width = 0.6
        self.house_height = 0.7
        self.roof_height = 0.3

        self.points, self.targets = self._generate(dataset_size, noise)

    def _left_roof(self, point):
        b = (0.5 * self.house_height - (self.house_height + self.roof_height) * (1 - sel
            (0.5 - (1 - self.house_width) / 2)
        k =  2 * (self.house_height - b) / (1 - self.house_width)
        return k * point[0] + b >= point[1]

    def _right_roof(self, point):
        b = (0.5 * self.house_height - (self.house_height + self.roof_height) * (1 + sel
            (0.5 - (1 + self.house_width) / 2)
        k = 2 * (self.house_height - b) / (1 + self.house_width)
        return k * point[0] + b >= point[1]

    def _generate(self, size, noise):
        points = [[random.random(), random.random()] for i in range(size)]

        targets = []
        for point in points:
            if (1 - self.house_width) / 2 <= point[0] <= (1 + self.house_width) / 2 and
```

```
                self._left_roof(point) and self._right_roof(point):
                inside = 1.
            else:
                inside = 0.

            if random.random() < noise:
                inside = 1. - inside

            targets.append(torch.tensor([inside], device=device))

        return points, targets

    def show(self, dividing_criterion=lambda s: 1 if s > 0.5 else 0):
        domik_x = []
        domik_y = []
        nedomik_x = []
        nedomik_y = []

        for sample in self:
            point, target = sample
            if dividing_criterion(target):
                domik_x.append(point[0])
                domik_y.append(point[1])
            else:
                nedomik_x.append(point[0])
                nedomik_y.append(point[1])

        plt.plot(domik_x, domik_y, '.', nedomik_x, nedomik_y,'.')

    def __len__(self):
        return len(self.points)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        sample = (torch.tensor(self.points[idx], device=device), self.targets[idx])

        return sample
```

У «Домика» есть 3 основных параметра: его ширина, высота и высота крыши. Здесь есть проверки на то, каким образом можно указать границу левой или правой крыши,

как сгенерировать данные точки. Также есть метод **show()**, который будет отображать этот набор данных. И исходные два параметра необходимо перегрузить для работы с torch.DataLoader. Подробнее о нем вы узнаете в следующей главе, сейчас не будем заострять на этом внимание. Визуально этот датасет выглядит следующим образом:

вы видите домик, состоящий из точек. Он достаточно существенный относительно всей остальной картинки. Точки внутри домика (синие) — это один класс объектов, соответственно оранжевые точки — другой класс объектов. Фактически здесь мы будем решать задачу бинарной классификации.

Давайте рассмотрим, как будет выглядеть функция обучения.

```
def train(network, train_dataset, epochs, criterion, optimizer):
    loss_epochs = []
    for idx in range(epochs):
        loss_samples = []
        for sample in train_dataset:
            optimizer.zero_grad()   # zero the gradient buffers
            point, target = sample
            output = network(point)
            loss = criterion(output, target)
            loss.backward()
            loss_samples.append(loss.data.numpy())
            optimizer.step()    # Does the update

        loss_samples_mean = float(sum(loss_samples)) / len (loss_samples)
        print(f"Epoch {idx: >8} Loss: {loss_samples_mean}")
        loss_epochs.append(loss_samples_mean)

    plt.plot(loss_epochs)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.show()
```

Она на вход принимает сеть (network), обучающую выборку (train_dataset), количество эпох (epochs), критерий (criterion) (каким критерием мы оцениваем то, насколько мы близки к итоговым значениям, так как у нас обучающая выборка состоит из объектов и ответов, и наша нейронная сеть, соответственно, выдает какое-то значение в результате, т.е. принимает что-то на вход и выдает конечное решение, и мы хотим этот результат сравнить с тем, что мы предполагаем получить. Метрика оценивания, насколько мы далеки от того, что должны иметь — это и есть критерий.) и оптимизатор (optimizer).

В данном семинаре мы будем везде использовать SGD (метод градиентного спуска).

Мы итерируемся по эпохам.

Дальше поэтапно просматриваем элементы в наборе данных, обнуляем градиенты (мы можем эти градиенты в какой-то момент аккумулировать и потом обновлять веса,

либо, как в данной ситуации, будем постоянно обнулять градиент для того, чтобы он не оказывал влияние на последующий элемент).

Дальше у sample (объекта обучающего выборке, который состоит из собственно объекта и ответа) мы выделяем объект — точку, которая отвечает за положение (x;y) и ответ — это принадлежность класса (0;1); в зависимости от того, внутри мы домика или вне.

Далее используем функцию network, т.е. помещаем нашу точку в сеть и получаем какой-то выход (output). Сеть что-то прогнозирует, и мы сравниваем по критериям, которые зададим в дальнейшем, ответ нейронной сети с ответом, который мы хотим получить.

Получаем ошибку.

Дальше, используя метод обратного распространения (backward propagation), мы прогоняем полученный результат обратно, чтобы обновить все нужные веса, для этого используем optimizer step, который их обновляет. Все, что написано ниже нам надо только для красивого отображения того, каким образом меняется loss в зависимости от эпохи. Тестовая функция выглядит почти таким же образом. Мы из sample выбираем входные и выходные данные, наш target, и после этого прогоняем входные данные через сеть, которая уже на этот момент обучена. Получаем ответ и дальше сравниваем, насколько хорошо он оценим, т.е. понятно, что нейронная сеть не выдаёт сразу 1 или 0, она выдает какое-нибудь число, типа 0,6/0,7/0,3. Мы должны этот ответ причислить к какому-то из двух классов. Так как у нас выборка практически равномерная (т.е. приблизительно одинаковое количество точек с 2-ух сторон), мы можем сказать, что те точки, которые получим со значением больше, чем 0,5, фактически являются единицами, а со значениями меньше, чем 0,5, соответственно, скорее всего принадлежат к нулевому классу. Dividing_criterion-это функция, которую вы передаете для того, чтобы оценить соответственно к какому из классов выборка принадлежит, а дальше сравниваем наш output(выход) (1 и 0) с таргетом (1и 0). В зависимости от этого мы можем разделить dataset для того, чтобы потом его отобразить красиво и сравнить с теми данными, которые мы хотим получить.

Давайте начнем рассматривать dataset.

```
DATASET_SIZE = 5000
domik = DomikDataset(DATASET_SIZE)

data = DataLoader(domik, batch_size=1)

# plotting
domik.show()
```

Зададим размер датасета в 5000 точек. Это достаточное количество, чтобы обучение шло довольно хорошо. В этой задаче мы не будем менять параметр batch_size. Выставим его равным 1. О его назначении мы также поговорим в следующей главе. Соответственно, это мы создали сам dataset domik, дальше мы создаем DataLoader — часть библиотечки

pyTorch: она разделяет набор данных на серии с возможностью сделать shuffle, т.е. может изменить порядок подачи данных из датасета. Дальше функция show, показывает итоговый результат.

Давайте для начала создадим совсем простую сеть. Как это сделать в PyTorch?

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(2, 1)

    def forward(self, x):
        x = F.sigmoid(self.fc1(x))
        return x

net = Net().float().to(device)
```

Создаем класс, где называем нашу сеть (Net), которая должна быть наследником класса Module (torch.nn.Module).

С помощью метода __init__ записываем, какие слои мы используем. В данном случае 1 слой fc1 (полносвязный слой) задается в фреймворке PyTorch, как nn.Linear (изначально мы импортировали torch.nn как nn). Используем аффинное преобразование, про которое мы много раз говорили (перемножение, сложение, добавка свободного члена). Что внутри? В качестве параметров принимается количество входов и выходов слоя. В данном случае 2 входа и 1 выход. Фактически мы таким образом задали 1 единственный нейрон.

Дальше функция forward нужна для того, чтобы показать, каким образом мы прогоняем сигнал от входа к выходу. Здесь **x** — входной сигнал, изначально помещаем его в наш слой self.fc1, а затем применяем функцию активации (сигмоиду) F.sigmoid(self.fc1(x)) и возвращаем результат работы метода: x, который был преобразован.

Дальше создаем переменную net (с маленькой буквы), которая будет хранить в себе экземпляр класса Net; float-формат нужен для показания того, что мы работаем не с целыми числами, а с типом float, т.к. точки лежат от 0 до 1, и to.device — для отображения того, что мы используем нашу сеть на видеокартах. Однако если мы будем использовать cpu-процессор, то to.device можно не писать; или изначально же мы написали функцию выбора (device/cpu), поэтому можно смело использовать этот код, он будет работать и там, и там. Загружаем сеть, у нее в данной ситуации in_features=2, out=1, bias=True.

Давайте обучим модель.

```
EPOCHS_TO_TRAIN = 10
train(network=net,
      train_dataset=domik,
```

```
        epochs=EPOCHS_TO_TRAIN,
        criterion=nn.MSELoss(),
        optimizer=optim.SGD(net.parameters(), lr=0.1))
```

Все основные составляющие уже описаны, поэтому нам просто надо подобрать подходящие параметры обучения. 1. Указываем количество эпох (10) (epochs_to_train=10) 2. В параметры network вставляем net(нашу сеть) 3. В train dataset записываем domik, а еще лучше, запишем DataLoader, т.к. это позволит нам работать с сериями (стандартный dataset их не поддерживает, там просто хранятся данные). 4. В качестве функции ошибки указываем, что будем использовать среднеквадратичное отклонение. 5. И optimizer. Мы будем использовать везде SGD и будем менять его скорость обучения, посмотрим, на что это влияет. Для начала сделаем скорость обучения совсем большой, чтобы посмотреть, как функция будет обучаться.

Обучение происходит какое-то время, так как данная задача очень простая, эпоха довольно быстро идет. Есть задачи, например по распознаванию текста, они обучаются сутками и, в целом, если у вас есть набор данных с достаточно сложной задачей, вычисление потребует достаточно большого количества времени, а время вычислительных ресурсов стоит больших денег, это тоже стоит учитывать. Поэтому, прежде чем проводить эксперименты, нужно подумать о том, какие параметры стоит поменять для того, чтобы это всё улучшилось. Несмотря на то, что все равно очень непредсказуемо, базируясь на знаниях об обучающей выборке, можно сделать предположение о том, как изменится итоговый результат, и о том, какой природы данные и какую задачу мы решаем.

По графику видно, что какое-то время Loss еще пытался меняться, но на самом деле, судя по значениям, можно сказать, что он прогнал все за один раз и больше не модифицировался.

Посмотрим на результат.

Фактически, сейчас мы создали 1 нейрон с 2 входами и одним выходом, который может только линейно разделить область и вообще весь dataset на 1 и 2. Как мы видим, он так и сделал и больше ничего не может. Но в итоге получилось 55% правильных ответов, не так уж и плохо, но для данной задачи это очень мало.

В целом обучение нейронных сетей построено на некоторой интуиции. Абсолютно верных решений для всех задач нет, поэтому если вы знаете данные, то можете улучшить процесс обучения. Давайте попробуем другую сеть.

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 4)
        self.fc2 = nn.Linear(4, 1)

    def forward(self, x):
```

```
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        return x

net = Net().float().to(device)
```

Например, мы видим, что Loss довольно низкий, кривая обучения хорошая, а в результате 96%. Радует то, что стало больше, но домик стал более закругленным. Это объясняется видом функции sigmoid (непрерывно дифференцируема, гладкая), но то количество нейронов и времени, которое мы использовали не позволили sigmoid стать достаточно узкой, чтобы эти углы хоть как-то выделить, однако, результат радует уже больше.

Если мы посмотрим на данные, то увидим, что здесь довольно ровненькое все, домик угловатый, поэтому давайте попробуем угловатую функцию relu использовать для угловатого домика. В данном случае используем leaky_relu для избежания возможных каких-то конфузов, связанных с умиранием весов. Давайте создадим leaky_relu, здесь уже два слоя соответственно, так как один нейрон уже сделать ничего не сможет, поэтому давайте использовать двухслойную сеть, но в ней будет два входных нейрона и один выходной.

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(2, 4)
        self.fc2 = nn.Linear(4, 1)

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x))
        x = F.leaky_relu(self.fc2(x))
        return x

net = Net().float().to(device)
```

Запустим обучение. Loss уже стал больше, но я предполагаю, что это не даст хорошего результата, функция обучения очень странная. Результат по-прежнему нас не удовлетворяет, поэтому давайте увеличим количество нейронов хотя бы до 10 штук и уменьшим скорость обучения. Как можно увидеть, мы стали идти гораздо точнее и график стал лучше: Loss падает, но в какой-то момент выходит на ровную прямую. В идеале мы стремимся к тому, чтобы в какой-то момент он стал плоским. Если Loss перестает меняться — значит обучение завершено, но стоит опасаться переобучения. Домик стал больше похож на домик, правильных ответов уже 94%. Давайте перезапустим

dataset (все это время было показано, каким образом видит сеть данные, на которых она обучалась) на данные, на которых она ещё не обучалась. Конечно, они все очень похожи, однако это все равно что-то новое для нее, поэтому мы видим, что домик, хоть и немного кривой, но уже гораздо больше похож на истинную картинку, чем было до этого.

Как же быть, если получается, что классов больше, чем 2? Далее на этом же примере хочу показать, как можно работать уже с несколькими классами. В принципе, данную задачу можно рассматривать с бинарной стороны, т.е. у нас либо единичка, либо нолик. Однако для мультиклассовой задачи будут использоваться другие метрики. Уже написанный нами универсальный код был сделан для того, чтобы показать, насколько хорошо он применим в большинстве случаев. Так как основной задачей является классификация, давайте поменяем метрику и, соответственно, будем использовать не MSE, а более полезную для классификации метрику-BCE (бинарная кросс-энтропия). Также рассмотрим один нюанс, что будет, если заменить, leaky_relu на sigmoid, также увеличить количество эпох и уменьшить скорость обучения в 2 раза.

Давайте рассмотрим мультиклассовую задачу, т.е. используем не бинарную классификацию, а категориальную. Внесем некоторые изменения, чтобы он работал с мультиклоссовостью. Давайте создадим dataset и новую выборку, которая будет отличаться от первой только ответами. Если в первой они задавались как тензоры, то для категориальных важно, чтобы был лейбл (именно целым числом), а не тензор. Поэтому данные изменения были внесены в набор данных, для его работы. Давайте будем использовать leaky_relu, так как данные у нас все-таки угловатые, и посмотрим, как это может нам помочь. Возникла ошибка, поэтому давайте torch. device заменим на cpu, это избавит нас от ошибки и для данной задачи. Благодаря хорошему процессору, скорость обучения только увеличится.

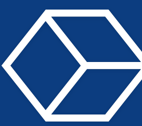
Далее давайте рассмотрим задачу (домик с окошком), которая гораздо сложнее для нейронных сетей, так как у нас область находится внутри другой области и разделить ее довольно непросто, но мы попробуем это сделать. Давайте значительно увеличим количество нейронов и рассмотрим это на leaky_relu. Получается очень много весов.

Мы можем еще улучшить наши результаты, например добавив еще один слой. Добавляем одну строчку и обязательно согласовываем между собой входы и выходы слоев. У вас может возникнуть вопрос по поводу теорем, в которых говорится о двухслойных сетях, которые могут предсказывать все, что угодно: почему тогда мы не видим этого? На самом деле, теоретически они могут, но вопрос скорее заключается в том, что необходимо подобрать определенные параметры и затратить на это достаточное количество времени. Найти такие параметры сложно, поэтому мы решим эту проблему, просто взяв сеть побольше и посложнее, которая гораздо быстрее придет к тому же самому. Я добавил весов, что повлекло за собой активное падение Loss, т.е. ошибку, которая вышла практически напрямую. Давайте оценим результат полученного (больше похоже, 96%).

3.3 Практическое занятие

Обучение домика с окнами и дверями. Обобщение полученных знаний.

4. Классификация изображений



4.1 Лекция

4.1.1 Проблема анализа изображений.

Как мы можем применить нейронные сети к задачам компьютерного зрения? Особенность в том, что даже при наличии теоремы, доказывающей возможность данного процесса, встречаются определенные трудности.

Рассмотрим выделенную область данного изображения. Для компьютера каждый такой квадрат — пиксель — обозначает какое-то число от 0 до 255. Это интенсивность свечения конкретного пикселя: 0 — черный, 255 — белый. Такая матрица хранится как информация об изображении. В чем проблема? На большой картинке очевидно, что это грань горы. Если вы рассмотрите приближенную часть изображения, то не поймёте, что на ней нарисовано. Данные не информативны. Наш пример — картинка 9 на 9 пикселей. Если мы будем закладывать в каждый входной нейрон один пиксель, для аппроксимации изображения, мы будем закладывать интенсивность каждого отдельного пикселя в свой нейрон. Для понимания изображения на входе нам нужно иметь 81 нейрон. Если изображение 10 на 10 пикселей, то это целых 100 входов. Размеры современных изображений минимум 300 на 300 пикселей. Невероятное количество ни о чем не говорящих данных для обработки. Из этого можно сделать вывод, что двух-трехслойные сети теоретически позволяют решить задачу анализа изображений, но подобрать веса и размер сети (какое количество нейронов должно быть в 1-м, во 2-м слое и т.д.) - проблематично. Четкого решения задачи нет — для каждой отдельной выборки оно подбирается отдельно. Даже в стандартной нейронной сети мы не можем применять полносвязные для анализа изображений. В поиске ответов, ученые посмотрели на устройство сетчатки человеческого глаза. Обнаружилось, что нейрон подходит не к каждому конкретному выходу, не к каждому конкретному сенсору (в данных случаях палочки и колбочки сетчатки глаза), он подходит к нескольким выходам сразу и потом формирует общий сигнал. Тем самым, прежде чем опускать информацию в классификатор или, соответственно, регрессор, мы должны агрегировать информацию с нескольких источников и скомпоновать ее в один конкретный объект. Мы должны выделить более информативные источники.

4.1.2 Сверточный слой

Для данной задачи придумали сверточный слой. Рассмотрим, как он работает математически. Попытаемся смоделировать сетчатку глаза человека. Перед вами точка приложения свертки и изображение. В каждой ячейке хранится конкретная интенсивность пикселя. Мы прикладываем операцию свертки к конкретной ячейке. Прогоняемся по всему изображению окошком — ядром свертки. Производим операцию свертки и результат записываем в следующий наш выход. Математически эта формула выглядит вот таким образом. Есть выход ячейки — k-той, m-ной ячейки. На самом деле, это просто двойная сумма, где Wi, j — это ячейка ядра свёртки, Xi + k, j + m - пиксель изображения. Мы фактически перемножаем ядро, суммируем все в одно, прибавляем

смещение и результат записываем в ответ. Здесь видно: 4 умножили на 0, 0 на 0 и т.д. и результат записали в точку приложения свертки.

Операция свертки

Свертка — это операция, которая применяется к 1-канальному изображению X на Y. У неё есть размер, ширина и высота. Параметрами свёртки является ядро — матрица разных размеров, обычно квадратная, которая прикладывается к разным местам исходного изображения. Элемент исходной картинки — это коды. Элемент матрицы, который называется ядром — это фича (англ. feature). Мы домножаем элементы одной картинки на элементы ядра и складываем все, что получилось. Получившуюся взвешенную сумму помещаем на следующий слой, т.е. в следующую карту признаков, в следующую feature map.

Ядра бывают разными. Рассмотрим на примере. Синее изображение — это исходное изображение, зеленое изображение — это то, что получается после свертки, т. е. следующая feature map. Тёмно-синее — это ядро свертки, которое "прикладывается" к исходному изображению. Получается, что веса, которые есть в этом ядре, домножаются на элементы исходного изображения. Для получения следующей feature map, следующей карты признаков, нам необходимо переместить ядро свертки и приложить к всевозможным местам на исходном изображении. Обратите внимание, что если приложить еще один такой-же темно-синий квадрат уже к левому углу исходного изображения, то получится еще некое значение, соответствующее левому пикселю следующей карты признаков. Из размера следующей карты признаков следует, что единственное возможное положение для ядра — следующее слева — через два пикселя от того положения, в котором оно нарисовано сейчас. Это подводит нас к тому, что такое stride.

Параметры свертки

Stride — это шаг, с которым сдвигается ядро свертки между соседними его прикладываниями, между соседним вычислением взвешенной суммы. Стандартная свёртка производит движение ядра с шагом 1 — сначала по X на 1, потом по Y на 1. Мы проходим всевозможные положения. Если сделать stride не равным 1, — например равным 2, — это ядро будет сдвигаться с шагом 2. При этом по X и по Y может быть разный stride.

Следующее дополнение к обычной свертке, которое можно использовать — **padding**. Если мы хотим преобразовать одну карту признаков в другую, но при этом не уменьшить ее размер, то можем дополнить нашу исходную карту признаков нулями по краям. Если бы stride был один, стандартный, мы бы смогли приложить ядро свертки вот в таком положении, потом в положении, когда она занимала бы середину левого нижнего ряда и, наконец, крайнем. Следующая карта признаков была бы размером 3 x 3, а исходная карта признаков размером 5 x 5, т.е. при использовании свертки без padding — без того, что изображено на средней картинке, — размер feature map уменьшился. Во избежании, мы можем дополнить карту признаков нулями и тогда размер изображения не изменится. Есть подход, когда значение за границами карты признаков дополняется не нулями, а отраженной копией того, что внутри, но это, скорее, экзотика.

Третий метод построения сверток — **dilation**. Веса, которые содержатся в ядре,

прикладываются к исходному изображению не подряд друг за другом, а через 1 (рисунок справа). При размере ядра 3 x 3, в исходном изображении захватывается область размером 5 x 5. Мы пропускаем некоторые пиксели и таким образом, за счет игнорирования некоторых элементов исходного изображения, увеличиваем охват. Если изображение имеет сравнительно низкую детализацию, при которой соседние пиксели очень близки по цвету, мы можем использовать dilatation. Однако метод используется не только в подобных случаях. Также можно было бы просто уменьшить изображение перед подачей в нейросеть.

Описанные методы можно комбинировать. Наиболее подходящий подбирается с помощью практики. Обратите внимание на то, что каждая комбинация параметров дает свой собственный размер следующего feature map на один и тот же размер предыдущего feature map. И stride, и padding, и dilation изменяют размеры следующей карты признаков.

Рассмотрим простейший пример одномерного фильтра. Нижние белые квадраты — это некий одномерный ряд. Изображенная справа матрица 3 x 1 — это ядро свертки. Сначала посмотрите на левую половину. Мы берем ядро свертки, прикладываем его к самой левой части, домножаем 1 на 0 - получаем 0, 0 на 1 - получаем 0, -1 на -2 — получаем -2, складываем 0 + 0 + -2 — получаем -2. В результате мы вычислили значение поэлементного произведения вот этого кусочка ряда — входного вектора, — и ядра. Далее мы делаем то же самое со сдвигом на 1 — также перемножаем, складываем и получаем другое значение. Делаем так для всех мест, в которых мы можем приложить ядро и получаем следующую feature map. Справа приведён пример применения свертки с таким же ядром, только с шагом равным 2.

Почему это фильтр, выделяющий границы? Посмотрим, как бы выглядели значения этого фильтра, если бы у нас была резкая граница темных и белых пикселей. Там, где пиксели только темные, мы бы взяли тёмный пиксель, домножили его на единицу, после взяли темный пиксель, помноженный на 0, после взяли такой же темный пиксель, помноженный на -1, и за счет того, что цвета пикселей одинаковые, в результате получился бы 0, и цвет стал бы черным, т.е. нулем. Если какая-то граница есть, то значения, соответствующие элементам ядра свертки, совпадать не будут и в результате на следующей feature map, мы получим значение разности между значениями пикселей с двух сторон границы.

В процессе обучения нейронной сети те самые веса, т. е. элементы ядра свертки, тоже обучаются. Также, как и все остальные параметры, которые обучаются в другие слоях. К ним применяется метод обратного распространения ошибки — backpropagation. После обучения они могут выглядеть, к примеру, подобным образом. Для решения задач есть фильтр, который обходит всё исходное изображение и через домножение на него пикселей этого исходного изображения происходит получение следующей карты признаков, с более высокоуровневыми feature maps.

Посмотрите на пример применения сверточной нейронной сети к изображению с машиной. Нужно отметить, что сверток может быть достаточно много — 32 или 64 штуки. Если на входе мы имеем одно изображение с 3 каналами, то с помощью сверток мы можем перевести это изображение, например, в 32 канала. У фильтров в этих свертках добавляется еще одна размерность и мы прикладываем к трехмерному tensore

с числами, к исходному изображению ядра свертки, располагая его в неких координатах по X и Y, при этом занимая всю его глубину, все три цветовых канала, тремя разными двумерными ядрами. Если мы возьмем 32 трехмерных ядра, то трёхканальное входное изображения преобразуется в 32-канальное выходное изображение. Иначе, — 3 одноканальных изображения преобразуются в 32 одноканальных изображения. Обратите внимание на изображение — при движении вправо с каждым следующим фильтром размер исходного изображения уменьшается. Слева — исходная картинка с машиной, посередине — что-то похожее на машину, а справа — низкополигональное изображение из маленького количества пикселей, совершенное не напоминающее исходный объект. Как это происходит? На примере стоит свёрточный слой — активация — свёрточный слой — активация — pooling, который мы рассмотрим далее.

Операция Pooling

Pooling — это уменьшение размера изображения, производимое за счет замены группы пикселей на функцию от этой группы пикселей. Простейший случай — это maxpooling. Из группы пикселей мы вытаскиваем максимум по ним и помещаем его на следующую карту признаков. Простейший случай maxpooling — это maxpooling 2 x 2. Предположим, что у нас есть некоторое изображения 4 x 4, мы делим его на 4 квадрата 2 x 2. Далее в каждом квадрате выбираем максимальное значение и перемещаем его на следующий слой. Во-первых, это позволяет уменьшить (в данном случае в 4 раза) количество пикселей, которые у нас есть в feature maps. Исходное изображение в full hd формате — это порядка нескольких миллионов значений. В результате мы хотим получить, например в случае классификации, всего несколько чисел, т. е. вектор из вероятностей и вряд ли их будет несколько миллионов. Мы должны сильно уменьшать размерность без потери информации. Сохранение информации означает выделение из всей информации наиболее для нас важную. Как правило, это яркие или граничные элементы. Если мы применим maxpooling к изображению с окружностью, то белая окружность на черном фоне по-прежнему останется окружностью. Если вместо maxpooling задействовать, например, average pooling, т.е. перенос на следующий feature map среднего значения по группе пикселей, то эта окружность стала бы более блеклой. Если мы вытаскиваем на следующий слой максимум, то окружность остается окружностью. Таким образом, за счет сохранения или увеличения интенсивности границ, мы можем сохранять значительную часть содержащейся в них полезной информации при уменьшении суммарного размера векторов.

4.1.3 Батч. Нормализация батча

Батч

При стандартной процедуре обучения модификация весов происходит сразу после прямого прохода и обратного прохода с вычислением градиентов. Но на самом деле модифицировать веса после каждого прохода необязательно, можно сначала посчитать несколько градиентов для всех весов, их сложить, а потом один раз поменять веса.

Размер батча — это как раз количество объектов, для которых мы получаем градиенты перед изменением весов сети. Если это число достаточно большое, то можно

практически в два раза уменьшить количество вычислений. Помимо этого, батчи помогают сделать обучение менее чувствительным к выбросам. Если есть шум — выброс, который не ложится в принципе в логику разделения данных на группы, то если модифицировать веса по нему, можно ухудшить производительность сети. Использование батчей предполагает усреднение, и таким образом позволяет в какой-то степени нивелировать наличие выбросов.

С другой стороны, большой размер батча потребует много видеопамяти. Особенно это актуально для картинок, при некотором batch size они перестают помещаться в память. Это является одним из принципиальных ограничений на размер батча.

Нормализация

Это приведение всех изображений в батче к нулевому среднему и к единичной дисперсии. Для того, что у нас пришло на вход — для вектора X картинки на Y картинки, на глубину картинки, на размер батча — мы считаем среднее и после делаем так, чтобы у всего batch среднее было равно 0, а дисперсия была равна 1. Это позволяет, при условии, что нормализация батча остается также и после обучения, т.е. на inference (когда мы применяем модель, она тоже применяется), — достичь того, что распределение на входе всегда плюс-минус одинаковое. Можно сказать, что данные становятся более однородными и за счет этого сеть учится быстрее и работает стабильнее.

4.1.4 Метрики для классификации

Поговорим про метрики, т.е. функции, позволяющие оценить итоговую производительность сети. Они отличаются от лоссов: лосс минимизируется в процессе обучения, он должен быть непрерывным (в смысле функций), а метрика не обязана. И, помимо этого метрики как правило проще интерпретировать, потому что лосс в 0.378 — это хорошо или плохо? А вот 89% правильных ответов — это вполне понятная вещь.

Немного отойдем в сторону и посмотрим на практическую постановку задачи бинарной классификации в приложении к робофутболу. В системе зрения гуманоидного робота, которого использует наша команда, есть быстро работающая часть, предоставляющая кандидатов в мячи. Грубо говоря, это округлые белые объекты на зеленом фоне. Эти кандидаты, которых уже мало, порядка десятка, подаются в нейросеть-классификатор, которая собственно решает, мяч это или нет. И вот в такой постановке метрика precision для задачи бинарной классификации — это количество правильных ответов «мяч» к полному количеству ответов «мяч». Recall — это количество правильных мячей к сумме количества правильных мячей и того, что мячом является, но сетью как мяч классифицировано не было. Во многих задачах важен и precision, и recall, и поэтому введена еще одна метрика, вычисляемая через них: это их удвоенное произведение, деленное на сумму. Удвоенное оно для того, чтобы F1 был равен единице, когда и precision, и recall равны единице.

4.1.5 Метрики для изображений

Во многих задачах, в частности в задачах компрессии изображений, может возникнуть необходимость численно оценить различие между двумя картинками. В качестве метрик

здесь вполне могут использоваться те же самые функции, которые были описаны выше в качестве функций потерь: если картинки похожи, то попиксельная разница будет маленькой, и MSE тоже. Но помимо этих функций есть и другие, например PSNR — пиковое отношение сигнала к шуму — это логарифм квадрата максимального значения по изображению, деленного на среднеквадратичную ошибку. Здесь MAX_original — это самый яркий пиксель исходного изображения, а MSE считается для исходного изображения и восстановленного.

SSIM, или структурное сходство, определяется через средние, мю, дисперсии и ковариацию двух изображений. Коэффициенты c1 и c2 определяюся через максимальные значения для используемого типа данных и в частности помогают избежать деления на ноль.

Многие простые метрики сходства изображений страдают от того, что специально подобранные или случайно попавшиеся картинки могут давать совершенно контринтуитивные меры близости. Например, практически идентичные картинки могут согласно метрике очень сильно отличаться или, наоборот, разные изображения по метрике могут быть близки. Есть даже статьи, где для разных метрик из компьютерного зрения подбирают такие изображения.

В задачах сегментации, т.е. сопоставления каждому пикселю изображения его класса, часто используется так называемый intersection over union — площадь пересечения правильного ответа и полученного, делённая на объединение.

4.1.6 Обучение. Валидация. Тест

Мы можем разделить весь набор данных, который у нас есть, случайным образом (это важно) на три части: первый называется train и бывает он обычно порядка 80 процентов. Две другие, по 10 процентов, называются валидация и тест. Тест — это те 10 процентов, которые мы откладываем в сторону до самого конца обучения чтобы потом получить честную и непредвзятую оценку того, как работает модель. Train и валидация используются при обучении следующим образом: мы производим обратное распространение по данным из трейна, т.е. нейросеть учится обобщать закономерности, которые в нем есть, а потом проверяем, чему равен лосс на валидации.

На картинке лосс на валидации выше, чем на трейне, но они равномерно падают. А в тот момент, когда лосс на валидации начинает расти, пора останавливать обучение. Почему это так? Потому что для данных, которые сеть еще не видела, она начинает давать результаты хуже чем до этого, т.е. она начинает учить набор данных которые ей доступны, учить train. Мы можем это обнаружить как раз с помощью валидационной выборки. До момента, когда лосс на валидации начал расти, сеть всё лучше и лучше работала на данных оттуда, т.е. она обобщала закономерности в данных, позволяющие работать с другими данными той же природы, а после этого она уже переобучается под конкретный набор тренировочных данных.

А зачем нужен тест? На самом деле валидация не влияет на то, как учится сеть: она влияет через критерий остановки. Мы останавливаем обучение, когда лосс на валидации начинает расти, т.е. в конечном счете модель, которая у нас получится, зависит от валидационной выборки тоже. И в результате независимое значение производительности

можно получить на отложенной выборке (на тесте), которая в обучении не участвовала вообще никак.

4.1.7 Confusion matrix

Confusion matrix — это матрица, в которой для каждого класса содержится, сколько элементов этого класса было отнесено к каждому из классов. В этом случае — сколько котов были приняты за котов, сколько котов за собак, сколько собак за котов и сколько собак за собак. Если в ошибках есть явный дисбаланс, например если для одного из классов задача решается очень плохо, можно подумать над тем, почему это так. Возможно, обратить внимание на разметку, перепроверить ее, а может разметить ещё объектов, если это возможно и если кажется, что этот конкретный класс по каким-то причинам трудно отделить от других. Естественно, чем больше числа на диагонали, тем производительность сети выше, потому что она принимает котов за котов, собак за собак и так далее.

4.1.8 Переобучение

Есть такой эффект, который называется переобучение, или overfit. Это ситуация, когда модель слишком хорошо выучивает те данные, которые мы ей предоставили, т.е. сначала она учит закономерности в данных, а потом начинает учить особенности конкретного набора данных. Разделение классов красных и синих точек, нарисованное черной кривой, кажется достаточно разумным, потому что оно игнорирует выбросы — элементы, находящиеся не в своей группе. Зеленая кривая на картинке гораздо более точно повторяет разделение синих и красных точек, но при этом такое разделение специфично конкретно для этого набора. Функция потерь для зеленой кривой будет меньше, чем для черной, потому что конкретно на этом наборе зелёная кривая гораздо лучше осуществляет разделение точек. Но если такие наборы генерируется случайным образом, то следующий набор будет отличаться от этого, и модель, которая выдала ответ, нарисованный салатовой кривой, получит достаточно плохой результат. Она будет переобучаться под закономерности, которых на самом деле в данных нет. Для того, чтобы бороться с переобучением, нужно знать, в какой момент стоит останавливать обучение модели.

Классификация объектов с помощью нейронной сети Постановка задачи классификации. Сверточные слои. Обработка данных. Обучение

Локальность признаков, биологические предпосылки к введению сверточных слоев. Параметры сверточных слоев, размерности данных и ядер.

4.2 Семинар

Для работы с семинаром, а также для выполнения практической части понадобится Python3 с библиотеками torch (pytorch), torchvision, matplotlib.

4.2.1 Работа с датасетами в PyTorch

Сейчас мы разберем работу с данными в PyTorch. Мы начнем с определения того, что такое датасет и рассмотрим две реализации, далее узнаем про подготовку данных и аугментацию и закончим на разбиении датасета на обучающую, валидационную и тестовую выборки. Работа с данными — очень важная часть любого исследования в глубоком обучении. От реализации загрузчика и обработчика данных будет зависеть как качество, так и скорость обучения. Более того хорошо написанный датасет делает проведение экспериментов более удобным. Для начала давайте поймем, что такое датасет и какие их виды бывают.

Определение

Выборка данных (датасет) — набор элементов и их характеристик.

- Задача классификации — самый простой вариант. Каждой картинке соответствует класс, определяющий объект, который на этой картинке изображен.
- Задача детекции — уже чуть сложнее, т.к. на одной картинке может быть расположено уже несколько объектов. Теперь для каждой картинки хранится не только класс, но и ограничивающий прямоугольник (bounding box на англ.).
- Все другие виды датасетов. На самом деле их вариантов очень много, но в рамках этого курса мы рассмотрим только эти два варианта, т.к. они чаще всего используются в робототехнике.

А теперь давайте перейдем к конкретной реализации в PyTorch:

Pytorch: простой датасет

```
1  from torch.utils.data import Dataset
2
3  class NumbersDataset(Dataset):
4    def __init__(self, low, high):
5        self.samples = list(range(low, high))
6
7    def __len__(self):
8        return len(self.samples)
9
10   def __getitem__(self, idx):
11       return self.samples[idx]
```

Работа с любой нейронной сетью начинается с загрузки данных. Для этого в PyTorch есть базовый класс Dataset от которого мы и будем наследовать наши загрузчики данных. Для дальнейшей работы нам нужно обязательно реализовать 3 функции: init — она обычно служит для указания, какие данные мы будем загружать, может выполнять загрузку картинок из указанной папки или, как в данном случае, просто генерировать последовательность числел от low до high; len, которая должна возвращать размер нашего датасета, И getitem, принимающую номер элемента из нашей последовательности

и по этому номеру возвращающую элемент.

Загрузчик данных

```
from torch.utils.data import DataLoader

dataloader = DataLoader(dataset, batch_size=40, shuffle=True)
for i, batch in enumerate(dataloader):
  print(i, batch)
```

Теперь у нас есть прекрасный датасет с числами, но его как-то нужно подавать в нейронную сеть. Для этого существует такой инструмент как Dataloader. Он принимает в себя размер датасета, размер пакета данных и уточняет, нужно ли их перемешивать. Dataloader сделает из нашего датасета генератор, переведет все данные в тензорный формат, потому что в pytorch нейронные сети принимают только его и сформирует батчи данных указанного размера. Давайте чуть остановимся на том, что такое батч. Обучение нейронной сети состоит их двух этапов — прямое и обратное распространение ошибок. Так вот, обратное распространение считается достаточно долго, поэтому принято сначала прогнать несколько элементов датасета через сеть, запомнить ошибки, а потом обновить веса нейронной сети по среднему от всех ошибок.

Pytorch: датасет с изображениями

```
import matplotlib.pyplot as plt
import cv2
import os
from google.colab import drive
drive.mount("/content/drive")
path_to_data = "drive/MyDrive/Colab Notebooks/data"
dt = cv2.imread(os.path.join(path_to_data, "10.jpg"))
dt = cv2.cvtColor(dt, cv2.COLOR_RGB2BGR)
plt.axis("off")
plt.imshow(dt)
```

Давайте рассмотрим датасет с изображениями мяча. Для начала загрузим картинки себе на диск, и посмотрим на пару картинок из датасета. Считываем нулевую картинку и выводим ее на экран. Мы видим футбольный мяч размера 64 на 64 пикселя.

```
from torch.utils.data import Dataset

class ImgDataset(Dataset):
  def __init__(self, path = "data", transform = None):
      self.path = path
      self.samples = [el for el in os.listdir(self.path) \
```

```
                    if el.endswith(".jpg")]
        print(self.samples)
        if transform is None:
          self.should_transform = False
        else:
          self.transform = transform
          self.should_transform = True

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path = os.path.join(self.path, self.samples[idx])
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        if self.should_transform:
          img = self.transform(img)

        return {"image": img, "type": "ball"}
```

Далее нам опять же требуется реализовать 3 функции init len getitem. Теперь мы готовы перейти к созданию датасета с картинками. В функции init мы запоминаем пути до всех существующие картинок из папки data. Также тут появляется переменная трансформ, но ее мы рассмотрим чуть позже. Функция len выглядит идентично, а вот getitem уже порядком сложнее. Тут мы по индексу получаем путь до картинки, считываем ее и, если нужно, применяем функцию трансформ. Результат работы этой функции — словарь с полями image, — тут лежит картинка и type — тут лежит название класса, к которому относится картинка.

```
dataset = ImgDataset(path_to_data)
plt.axis("off")
plt.imshow(dataset[10]["image"])
```

Проверим работу нашего датасета. Для начала инициализируем, а затем отобразим нулевой элемент.

Подготовка данных

```
from torchvision import transforms, utils


scale_factor = 0.5
img_size = dataset[0]["image"].shape[:2]
```

```
6
7 scaled_size = (int(img_size[0]*scale_factor), int(img_size[1]*scale_factor))
8 crop_size = (30, 30)
```

Зачастую перед обучением нейронной сети требуется предобработать данные. Давайте рассмотрим возможные случаи на примере. Пользоваться будем методом transforms из torchvision. Объявим необходимы константы. Это масштаб, и размер обрезанной картинки.

```
1 data_transform_list = [
2         transforms.ToPILImage(),
3         transforms.Resize(scaled_size),
4         transforms.CenterCrop(crop_size),
5         transforms.ToTensor(),
6         transforms.Normalize(mean=[0.485, 0.456, 0.406],
7                              std=[0.229, 0.224, 0.225])
8     ]
```

Для начала нужно перевести картинку в формат PIL, PyTorch работает именно с таким форматом картинок. Обычно нейронная сеть принимает картинки определенного разрешения, поэтому нам просто необходима функция Resize. Следующая — вырезание из центра (CenterCrop). Это преобразование может понадобиться на практике, если информативен только центр картинки или по краям наблюдаются сильные искажения, например когда изображение широкоугольное. Далее мы приводим картинку в тензорный формат (ToTensor), так как уже знаем, что Pytorch работает только с тензорами. Последним шагом мы применяем нормализацию (Normalize). Это позволит нашей сети учиться эффективнее. Данные параметры нормализации являются стандартом. Они посчитаны на самой большой базе картинок — ImageNet, где содержится более миллиона картинок.

Аугментация

Мы пришли к самой интересной и нетривиальной задаче в обработке данных перед обучением — увеличение объема данных без новых данных. Давайте внимательно посмотрим на картинку с мячом. Кажется, что если мы повернем эту картинку на 90 градусов, то мяч так и останется мячом. Давайте попробуем ее поворачивать. Выходит, что из одной картинки с мячом мы смогли сделать сразу 4, т.е. увеличили размер датасета в 4 раза! А что если вращать с меньшим периодом? А что если вращать картинку на случайный угол каждый раз, когда мы пытаемся достать ее из датасета? Таким образом у нас получится возможность каждую эпоху обучения получать разные картинки и, тем самым, увеличивать количество информации в данных. Мы с вами понимаем, что повернутый мяч все еще мяч. И он с таким же успехом сможет встретиться нам в игре. Нейронная сеть так сразу это понять не может, ей требуется это «объяснить». Абсолютно такую же логическую цепочку можно провести для масштаба и отражения.

```
from torchvision import transforms

data_augentation_list = [
      transforms.RandomRotation(360),
      transforms.RandomCrop((60,60)),
      transforms.RandomHorizontalFlip(),
    ]
```

Теперь давайте посмотрим, как это работает. Еще раз воспользуемся модулем transforms из torchvision. С его помощью зададим набор преобразований, о которых мы говорили выше. Случайный поворот от -20 до 20 градусов. Случайное изменение масштаба до 300 на 300 пикселей, случайное отображение с вероятностью 50 процентов.

Разбиение датасета

```
import torch
train_set, val_set, test_set = torch.utils.data.random_split(data,
                                                              [80, 10, 10])
print(len(train_set), len(val_set), len(test_set))
```

В заключение мы рассмотрим разбиение датасета на обучающую, валидационную и тестовую выборки. В фреймворке torch есть функция random split. Первым аргументом мы передаем датасет, а вторым длинны обучающей, валидационной и тестовой выборок. Сумма длин должна быть равна длине датасета.

4.2.2 Реализация нейросетевого классификатора изображений

Рекомендуем выйти в runtime, нажать change runtime type и поменять hardware accelerator на GPU, т.е. на видеокарту. Работать с графическим процессором быстрее.

Мы рассмотрим загрузку одного из стандартных dataset, что это за dataset, какие данные в нем содержатся, как можно было бы решать задачу классификации для цифр классическими методами — классическим компьютерным зрением, — определим простую нейросеть, посмотрим на итерации TrainTest и на train_loop, на цикл обучения, а также на графики лосса, которые будут рисоваться в процессе обучения. Потом попробуем поменять некоторые параметры сети и посмотреть, как это влияет на обучение.

Для тренировки мы используем Dataset MNIST, состоящий из TrainTest на 60 тысяч изображений 28 x 28 одноканальных для трейна и 10 тысяч таких же для теста.

Создание сети и пайплайна обучения

В PyTorch есть доступный способ загрузить необходимый нам Dataset MNIST. Нам нужно передать train_loader или test_loader. Дополнительно нужно передать batch_size. Выше мы его определили как 64. Напомним, что batch_size — это количество изображений, по которым будут собираться градиенты для сети перед тем, как будут использованы

для изменения весов сети. Сначала мы посмотрим на 64 картинки, вычислим то, насколько нейросеть для них «не права», а после обновим веса — один раз на 64 картинки.

```
batch_size = 64
no_cuda         = False
use_cuda = not no_cuda and torch.cuda.is_available()
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ])),
    batch_size=batch_size, shuffle=True, **kwargs)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ])),
    batch_size=batch_size, shuffle=True, **kwargs)
```

Загружаем картинки для трейна и для теста — разница только в том, что в загрузке для теста train=False. Изучим, что в нём лежит. Посмотрим на первый batch из того, что выдает train_loader. После первого цикла выходим из него. Далее выведем первые 4 картинки из 64 и распечатаем их лейблы — что мы хотим получить из этой картинки.

```
import matplotlib.pyplot as plt
from IPython.display import display, clear_output

for data, target in train_loader:
    for i in range (4):
        print (target [i])
        plt.imshow (data [i, 0, :, :])
        plt.show ()

    break
```

Если мы хотим получить 5, то и выводим 5 как tensor. Разрешение изображения низкое, цифры написаны неровно, по-разному, и на глаз трудно определить шаблон, по которому можно отделить одно изображение от другого. Наши изображения одноканальные, а цветовая гамма от фиолетового к желтому — это формат представления одноканальных

картинок. Здесь задействован uint8 — от 0 до 255.

Как можно подойти к этой задаче со стороны классического компьютерного зрения? Как вариант, — воспользоваться анализом связанных компонентов и получить маску всех ненулевых пикселей. После найти количество концов у цифр: — у цифры 5 их 2, у восьмерки — 0, у шестерки — 1. Затем выяснить, сколько у цифры есть внутри областей, по цвету совпадающих с фоном, т.е. какова характеристика нашей области с точки зрения связанности. У восьмерки есть два кольца, у шестерки одно, и таким образом мы отделяем две цифры друг от друга. С другой стороны, если пятерка написана неаккуратно, то может получиться замыкание. Методы классического компьютерного зрения классического могут плохо работать с разнообразными наборами данных. Нейросети справляются с этим значительно лучше, потому что при правильном подходе к обучению они находят наилучшие критерии из всех возможных для классификации цифр от 0 до 9. За счет механизма обратного распространения ошибки — backpropagation, грамотной инициализации, правильном выборе момента остановки мы получим лучший способ разделения данных на классы, который следует из этих данных.

Посмотрим, какую сеть мы определили для того, чтобы решить задачу классификации — отнести одноканальную картинку 28 x 28 из пикселей типа np.uint8 к одному из 10 классов. Она называется Simple_net благодаря своей простоте.

```
class Simple_net(nn.Module):
    def __init__(self, hidden, out_sz):
        super(Simple_net, self).__init__()

        self.fc1 = nn.Linear(28**2, hidden)
        self.fc2 = nn.Linear(hidden, hidden)
        self.fc3 = nn.Linear(hidden, out_sz)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.leaky_relu (x)

        x = self.fc2(x)
        x = F.leaky_relu (x)

        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)

        return output
```

У нее сразу есть два параметра, чтобы их можно было менять снаружи. Отметим, что out_sz меняться не будет -— это просто 10 по числу классов. Hidden — это количество нейронов в скрытом слое. Мы не будем учитывать, что соседние пиксели изображения

как-то связаны между собой. Сразу после того, как в функцию forward пришел X, мы возьмем его и вытянем вектор, т.е. мы не будем учитывать геометрическую связь между соседствующими пикселями. Будем рассматривать изображение просто как вектор длины 28 в квадрате. После применим к этому вектору первый полносвязный слой. Он устроен таким образом: на входе вектор размера 28 в квадрате, на выходе вектор размера в hidden, т.е. в размер скрытого слоя. Позже попробуем определить его совсем маленьким, просто маленьким и средним и посмотрим, как это повлияет на результат. Далее мы применим функцию активации, применим ещё один слой и переведем вектор размера hidden в вектор размера out_sz, т.е. в вектор с вероятностями, которые пойдут на выход для классификации. После с помощью log_softmax мы получаем то, что нам нужно.

Функции Train/Test устроены практически эквивалентно, за исключением того, что в одном месте модель находится в режиме train, а в другом в режиме eval.

```
1  def train(model, device, train_loader, optimizer, epoch, log_interval, loss_archive):
2      train_loss = 0
3      loss_epochs = []
4      model.train()
5      for batch_idx, (data, target) in enumerate(train_loader):
6          loss_samples = []
7          data, target = data.to(device), target.to(device)
8          optimizer.zero_grad()
9          output = model(data)
10         loss = F.nll_loss(output, target)
11         loss.backward()
12         optimizer.step()
13 
14         train_loss += loss.item () * batch_size
15 
16         if batch_idx % log_interval == 0:
17             print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
18                 epoch, batch_idx * len(data), len(train_loader.dataset),
19                 100. * batch_idx / len(train_loader), loss.item()))
20 
21     train_loss /= len(train_loader.dataset)
22     loss_archive.append (train_loss)
```

```
1  def test(model, device, test_loader, loss_archive):
2      model.eval()
3      test_loss = 0
4      correct = 0
5      with torch.no_grad():
6          for data, target in test_loader:
```

```
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target).item() * batch_size  # sum up batch
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-pr
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    loss_archive.append (test_loss)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

Каждый batch log_interval (log_interval = 100) происходит распечатка текущего значения лосса. В конце прохода по dataset — одной итерации трейна, одной эпохи — записываем в архив лоссов, переданный снаружи, усредненный лосс по данному проходу.

Давайте соберем все воедино, чтобы создать пайплайн обучения.

Определим epochs — количество эпох, learning rate (lr), gamma, seed — число случаев инициализации, log_interval — то, во сколько batch будет происходить распечатка среднего лосса. При возможности учимся с помощью cuda, т.е. на видеокарте при ее доступности.

```
epochs          = 10
lr              = 0.1
gamma           = 0.7
seed            = 1
log_interval    = 100

torch.manual_seed(seed)
device = torch.device("cuda" if use_cuda else "cpu")
```

Определяем модель и приводим ее к устройству. В нашем примере 3 — это количество скрытых нейронов, 10 — сколько значений мы хотим на выход.

```
model = Simple_net(100, 10).to(device)
```

Определяем оптимизатор - это Adadelta.

```
optimizer = optim.Adadelta(model.parameters(), lr=lr)

scheduler = StepLR(optimizer, step_size=1, gamma=gamma)
```

Рассмотрим цикл:

```
for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval, train_loss)
    test(model, device, test_loader, test_loss)
    scheduler.step()

    # plotting part
    plt.figure (figsize=(24, 15))

    plt.plot (train_loss, 'r')
    plt.plot (test_loss, 'g')
    plt.ylim(ymin=0)

    plt.show ()

    plt.pause (0.01)
```

В train передаётся device. Данная функция отличается тем, что здесь data и target приводятся к устройству. Приведение к устройству означает, что данные копируются из обычной памяти в видеопамять, если это требуется. Определим model, optimizer, scheduler, лоссы. Здесь и происходит train loop. Вызываем train_test и рисуем то, что получилось.

```

```

Обучение классификатора. Анализ полученной модели

Запускаем и смотрим на результат. На каждую новую распечатываемую строчку происходит обработка 100 batch. После завершения первой эпохи Accuracy 54% — это значит, что правильная цифра была выяснена по картинке в 54% случаев. На следующей эпохе сети удалось сделать уже 59%. Лосс падает, а Accuracy растет. В целом потенциал этой сетки маленький, потому что у нее есть узкое место — 3 скрытых нейрона, в которые с помощью полносвязного слоя отображаются все 28 в квадрате значений, приходящих на вход. Данная сеть вряд ли сможет добиться Accuracy выше 70%.

Поменяем количество скрытых нейронов на 10 — столько же, сколько во входном слое. Лосс становится гораздо меньше, чем при прошлом запуске обучения. После первой эпохи Accuracy составляет 90%, т.е. мы правильно определили самое узкое место, в котором нейросети следовало дать больше свободы. Ей было трудно кодировать признаки с помощью настолько короткого вектора - всего из трех компонентов.

Увеличиваем размер скрытого слоя до 100. Accuracy сразу становится 93%.

Вернемся к графику обучения. К финалу Accuracy составляет 96% (в 96% случаев цифра была написана правильно), только за счёт увеличения размера скрытого слоя.

Попробуем увеличить количество слоев в сети. Заводим еще один линейный слой, вставляем его в середину.

Сначала будет происходить преобразование из размерности входного вектора в скрытый слой, потом из вектора, размером со скрытый слой в аналогичный вектор и после уже на выход. Запускаем и наблюдаем.

В конце первой эпохи Accuracy составляет 93%. На третьей эпохе — 95%. Самостоятельно можно добавить еще один такой же слой и посмотреть результат изменения размера этого слоя.

Стоит помнить, что полносвязный слой — это слой, в котором все нейроны связаны друг с другом. Полносвязный слой из 100 в 100 — это 10 тысяч весов. Из-за такого большого количества на маленьких наборах данных может произойти запоминание dataset.

Accuracy перестала расти, и зеленый график, соответствующий train, стал чуть выше красного, соответствующего test, и перестал опускаться. Происходит выучивание dataset. Мы дождались, что Accuracy упала с 9626 до 9621. Разброс такого рода может случиться при разной начальной инициализации и находится в районе статистической погрешности. В целом мы можем сделать вывод, что обучение прекратилось.

4.2.3 Реализация классификатора изображений на сверточных сетях

Рассмотрим цифру 6. Если взять два произвольных пикселя, находящихся рядом друг с другом, их цвет будет не сильно отличаться, за исключением пикселей на границе. Цвета будут одинаковы. Более всего для определения цифры нас интересуют границы. Ориентируясь на форму объекта, мы можем достаточно достоверно понять, что он из себя представляет. Возможнось эффективно выделить из всего входного изображения то, что наилучшим образом описывает его форму и не только при получении компактного представления входного изображения, может помочь нам в построении классификатора с хорошими характеристиками. Для этого на следующем семинаре нам пригодятся сверточные нейронные сети. Мы будем проходить ядрами — матричками, как правило квадратными, по изображению и с каждым последующим слоем будем выделять все более высокоуровневые признаки. В отличие от представления вектора в данной сети, сверточная сеть будет «знать» о том, что соседние пиксели как-то связаны, и свертки будут учиться выделять признаки, свойственные соседствующим пикселям.

Создание сети

Поменяем метод, с помощью которого мы подходим к решению задачи классификации цифр, оставив данные неизменными. Зададим более сложную архитектуру сети. Создадим сверточную нейронную сеть и посмотрим на ее работу — какие преобразования она делает с входным tensor и как заданные руками фильтры работают на элементах набора данных. Импортируем библиотеки, загружаем данные и применяем к первому элементу первого batch из train dataset фильтр с ядром из единиц 3 x 3. Разница между сверткой и корреляцией, которая здесь применяется, — функция filter2D для симметричных ядер отсутствует. Ядро из единиц является симметричным относительно диагонали. Результат применения фильтра 2D совпадает с результатом применения свертки с таким же ядром.

```
import matplotlib.pyplot as plt

for data, target in train_loader:
    for i in range (4):
        print (target [i])
        img = data [i, 0, :, :]
        plt.imshow (img)
        plt.show ()

        # kernel = np.ones((3, 3), np.float32)
        kernel = np.array([[-1, 0, 1]], np.float32)

        convolved = cv2.filter2D(img.detach().cpu().numpy(), -1, kernel)
        plt.imshow(convolved)
        plt.show()

    break
```

Берем первый batch из train_loader и сразу после работы с ним выходим из цикла. Далее из первого batch берем нулевую картинку — i = 0. Берем из нее нулевой канал, единственный из доступных, поскольку картинки одноканальные. Остальные измерения мы берем полностью, как наборы. Получается двумерное изображение, которое мы выводим. Затем делаем свертку с ядром, состоящим из единиц. Поскольку в середину ядра в пиксель результирующего изображения попадает усредненное значение всех соседей, включая сам средний пиксель с коэффициентами 1, можно увидеть, что границы значительно смазались. Вместо двух выделяющихся на фиолетовом фоне пикселей, ниже и выше, размазанные пиксели — так работает усреднение. Частный случай фильтра и при неком наборе данных один из фильтров, соответствующий одному из каналов, может научиться делать именно это. Нужно держать в голове, что такие фильтры обучаются и коэффициенты в них — это не единица. Здесь мы видим пример применения такого единичного фильтра к изображению — исходная картинка сверху более резкая картинка, снизу — менее. Применим к картинке, а не к одномерному сигналу, другой фильтр — выделяющий границы. Ядро фильтра - [-1, 0, 1] — этот фильтр вычисляет интенсивность переходов. Похоже на приближение с точки зрения вычислительной математики дискретной производной. Если мы посмотрим на пиксели в середине тройки, то при переходе из области левее вправо значения увеличиваются, поэтому производная здесь положительная. При переходе из середы в область правее значение уменьшается, поэтому производная здесь отрицательная. Бирюзовый цвет — это 0, желтый — больше, чем ноль, синий — меньше, чем 0. Таким образом работает фильтр, выделяющий границы. Если бы здесь было равномерное поле, залитое темным цветом и на нем круг из светлого цвета, то не фоновый цвет в результирующем изображение был бы только на границах, в маленькой окрестности, задаваемой размером ядра.

Дальше определим сверточную сеть иным образом.

```
class Conv_net(nn.Module):
    def __init__(self):
        super(Conv_net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        #torch.Size([64,1,28,28])
        x = self.conv1(x)
        #torch.Size([64, 32, 26, 26])
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        #torch.Size([64, 64, 12, 12])
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        #torch.Size([64,9216])
        x = self.fc1(x)
        #torch.Size([64,128])
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        #torch.Size([64,10])
        output = F.log_softmax(x, dim=1)
        #torch.Size([64,10])
        return output
```

Определяем для сети следующие элементы — двумерная свертка с параметрами: 1 feature map на входное изображение (т.к. это будет первая свёртка во всей сети, количество каналов здесь равно количеству каналов во входе) оттуда сразу получается 32 feature mapа (мы будем учить 32 разных ядра делать 32 разные свертки) размер ядра необходимо указывать — в данном случае 3 stride по умолчанию 1. Вторая свертка будет переводить 32-канальные изображения, полученные после первой свертки, в 64-канальные с размером ядра 3 и также со stride = 1. Далее идут два слоя dropout - это слои, в которых с вероятностью 0.25 для первого и 0.5 для второго значение заменяется на 0. Это улучшает стабильность обучения — добавляет стохастичности и при разных проходах по одному и тому же элементу из dataset у него видны разные

пиксели, что повышает способность сети к обобщению признаков. После определены 2 полносвязных слоя. Первый из них отображает одномерный вектор, который получается из вектора, полученного из результатов второй свертки после flatten. Он отображается в вектор размерностью 128. Второй полносвязный слой отображает такой вектор в вектор размерностью 10. 10 — это наш ответ. Посмотрим на forward и на то, какие размерности у входных tensor. На вход приходит tensor размерности 4 — у него 64 картинки в batch, один канал у каждой картинки и размеры по X и по Y у картинок 28 x 28 пикселей. Применяем к X первую свертку. Становится, по-прежнему, 64 batch, на этот раз 32 канала и уменьшается на 1 размер по X и по Y, т.к. когда свёртка приложена в углу, она может посчитать значение только для не углового элемента, а находящегося рядом с ним на диагонали. Другими словами, мы не можем посчитать значение для углового элемента с нулевым pooling. Если нет pooling, размер картинки уменьшается на 1 пиксель с ядром размера 3. Далее применяем активацию. Применяем вторую свертку. Она делает практически то же самое, что и первая, только переводит 32 канала 64 — это будет видно позже. Размер batch остается неизменным до конца — 64. Помимо этого, изображение из 26 x 26 становится изображением 24 x 24. Применяем maxpooling квадратом 2 x 2, т.е. вытягиваем из них максимальное значение и после размер tensor остаётся следующим - 64 картинки в batch, 64 канала и картинка 12 x 12, потому что до pooling она была 24 на 24. Далее следует первый слой dropout, не меняющий размерности и flatten — приведение параметров к одномерному вектору размером 9216. После применяется первый полносвязный слой, который оставляет неизменным batch_sz и переводит 9216 элементов в 128. Активация. Второй dropout. Второй полносвязный слой, который переводит 128 фичей в 10 фичей - в конечную вероятность классов, которые нас интересуют.

Обучение сверточного классификатора. Сравнение с полносвязным

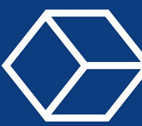
Мы рассмотрели устройство forward. В результате train и test остались такими же. Также мы определили сверточную сеть вместо простой сети — посмотрим, как она учится.

По сравнению с прошлым результатом в 96%, Accuracy стала 98% процентов сразу после начала обучения. Это произошло за счет добавления сверток, способных выделять некоторые признаки, свойственные именно изображению как упорядоченному объекту на плоскости. Учитывая, что мы знаем, какой пиксель с каким соседствует. Свойства изображения хорошо описывается через свойства групп пикселей. После первой эпохи лосс лучше, чем был после 10 эпохи для не сверточной сети. За счет того, что свертки содержат сравнительно немного параметров относительно полносвязных слоев — они компактно хранятся, — и за счет выделения признаков, характерных для соседствующих групп пикселей, они в свое время обошли человека в задачах, связанных с компьютерным зрением.

4.3 Практическое занятие

Реализация классификации мяча на датасете: , используя знания из этого семинара.

5. Детекция объектов



5.1 Лекция

5.1.1 О задаче детекции

Детекция объектов — процесс выделения объектов на изображении.

Начнем с рассмотрения иллюстрации, на которой расположено множество объектов разного формата, формы и размера. Мы уже знаем про задачу классификации, где у нас на входе есть картинка, а на выходе мы предсказываем её класс. В детекции задача немного расширяется и, помимо определения класса, нам необходимо локализовать объект. На входе мы имеем изображения, а на выходе — набор предсказаний, содержащих описание рамки и класс объекта, а также вероятность нахождения конкретного объекта в ограничивающей рамке.

Важно учитывать, что на картинках могут быть объекты (things) и материалы (stuff). Объекты — это структуры, имеющие определенный размер и форму. Материалы, как правило, тоже имеют форму, однако они более обширны и определяются однородным или повторяющимся шаблоном мелких деталей без определённого размера. Соответственно, для решения задачи найти человека можно использовать модель детектирования, а если мы хотим зафиксировать материалы, то с этой задачей справится модель из семантической сегментации.

5.1.2 Наборы данных

Машинному обучению для тренировки алгоритма нужны данные. Все накопленные на текущий момент коллекции данных делятся на два типа: многоклассовые (Pascal Visual Object Classes, Microsoft COCO) и одноклассовые (лица, машины, пешеходы, дорожные знаки).

Подробнее про многоклассовый тип. Pascal VOC содержит в себе 20 классов, 11530 изображений, 27450 регионов интересов (RoI) и 6929 сегментационных масок. Microsoft COCO содержит в себе 80 классов, более 200 тысяч изображений (train, val, test), 1.5 миллиона экземпляров объектов и 500 тысяч выделенных объектов (с масками).

Существует обширное количество коллекций данных, называемых benchmarks. На сайте paperwithcode.com в разделе Object Detection находятся те датасеты, на которых чаще всего сравнивают алгоритмы по детекции.

5.1.3 Метрики качества

Основные метрики качества: Intersection over Union (IoU), Точность (Precision), Полнота (Recall), Average Precision (AP), Mean Average Precision (mAP).

IoU

Вы видите картинку с двумя ограничивающими рамками (или иначе bounding box). Синяя рамка представляет собой правильную разметку, а красная получена с помощью какой-либо модели детектирования. Мы можем посмотреть на пересечение между первой и второй рамкой, а также можем посмотреть их объединение. Если рамки накладываются

друг на друга, то отношение пересечения с объединением будет равно единице, т.к. первая и вторая величины равны. Если же красная рамка находится далеко и никак не пересекается с синей рамкой, пересечение IoU будет равно нулю. Таким образом, метрика показывает хорошее качество, если она ближе к единице и ухудшение качества в случае близости к нулю.

Также на изображениях вы можете увидеть примерные значения IoU в различных случаях пересечения предсказания и настоящей рамки.

Precision и Recall

Когда мы знаем IoU для двух рамок, мы можем определить порог — обозначим его p. Если значение IoU больше p, то мы говорим, что у нас предсказание true positive. Если IoU меньше p, то предсказание false positive. Случаи пропущенного примера, когда для размеченного bounding box нет предсказания, называются false negatives. По приведённой иллюстрации можно четко понять суть каждой из данных величин. Наша задача — детекция людей. Мы видим четыре реальных bbox, которые были размечены изначально. Три из них предсказаны более-менее корректно, и в случае выбора небольшого порога алгоритм засчитает их как true positive. Также есть рамки, которые не попадают ни на одного человека, — их IoU меньше нашего порога, — и это false positive.

Такая формализация позволяет дальше подсчитывать метрики и задачи классификации.

Точность (Precision) — доля истинных объектов основного класса среди всех классифицированных как основной класс.

Precision = Number of true detections Number of detections

Полнота (Recall) — доля правильно распознанных объектов основного класса среди всех объектов основного класса из тестовой выборки.

Recall = Number of true detections Number of gt objects

На нашей картинке всего 5 детекций — 5 красных прямоугольников. Но при этом true positive, т.е. верно предсказанных прямоугольников, всего лишь 3. Соответственно, их отношение, т.е. точность будет 3/5 или 0,6. Если мы считаем Recall, то соотносим 4 bbox для каждого человека на картинке и 3 правильно предсказанных, получая 3/4 или 0.75.

Теперь мы можем построить график зависимости Precision и Recall от порога. Мы определяем p — наш порог, — по которому решим, является ли пример true positive или false positive. Соответственно, мы можем его варьировать от нуля до единицы, потому что IoU принимает значение от нуля до единицы. Например, мы можем идти шагом 0.1 и с каждым шагом пересчитывать Precision и Recall и посмотреть, где находится оптиум. Как правило, на подобных кривых Precision сначала большой и впоследствии начинает падать, а Recall, наоборот, нарастать. Это связано с тем, что, как мы уже знаем, в Precision сначала смотрится количество правильных детекций к общему количеству детекций (подразумевается ground true, т.е. настоящие размеченные детекции). Если при оценке IoU у нас маленький порог, мы берем предсказанный прямоугольник, даже если он находится далеко или имеет значение 0, т.е. нулевое пересечение, и всё равно говорим, что он true positive. Соответственно, количество реальных детекций будет равно количеству детекций в целом, потому что мы сообщаем нашему детектору, что можно

вообще не угадывать реальное местоположение и даже не пересекаться с реальным bbox — он будет прав в любом случае.

Если мы начнем повышать Recall, получим больше ограничения на близость реального bbox и предсказанного и кривая будет падать.

AP и mAP

AP и mAP — это кумулятивная величина. Представим, что у нас есть n классов и мы учитываем величину piterp(r), где r — значение порога. Можем двигаться шагом 0.1.

piterp(r) — максимальная точность для полноты меньше r.

Снова обратившись к графику, мы фиксируем Recall, который меньше 0.1, 0.2 и т.д., и смотрим, какой у него Precision самый большой.

mAP - среднее AP по классам.

Т.к. у нас может быть не один класс детекций, как в Pascal VOC и Microsoft COCO, у нас есть возможность посчитать AP для каждого класса отдельно, а после усреднить значение.

5.1.4 Методы решения

Теперь переходим к решению задач детекции. В первую очередь может появиться мысль адаптировать задачу классификации к задаче детекции, т.к. задача классификации — один из пунктов задачи детекции. Обсудим, какие проблемы могут возникнуть в процессе.

Первый метод — скользящее окно. Мы берем исходное изображение, фиксируем размер скользящего окна и проходимся им по всему изображению по пикселю слева-направо, снизу-вверх и каждую часть, т.е. классифицируем каждое окно. Таким образом, для каких-то объектов мы сможем определить местоположение, а также будем знать класс. Очевидно, что такой способ создаст проблемы. Если мы берём фиксированное окно, то будем учитывать только фиксированный размер объектов. В зависимости от дальности расположения объекта или по причине разных размеров объектов по своей природе, например, собак, — с размером необходимо что-то сделать. Также важны пропорции, т.к. объекты могут быть развернуты, стоять на двух или четырёх лапах и bbox будет увеличиваться, расширяться, сужаться и т.д. Бывают ситуации, когда объекты перекрываются и один объект находится на фоне другого объекта, что тоже надо разрешить, потому что в рамках одного окна мы можем сделать мультиклассификацию, получить два класса и нужно определять bbox для второго класса. Учитывая то, что мы двигаемся скользящим окном попиксельно, наш классификатор может определять объект много раз в какой-то окрестности. На примере мы можем увидеть множественные отклики — множество bboxes, каждый из которых, по сути, правильно локализует человека, но есть небольшое некритичное смещение, всё равно требующее корректировки.

Рассмотрим решение каждой проблемы. Начнем с размера. Понятно, что можно увеличить или уменьшить окно. Альтернативный способ — сделать пирамиду изображений разных размерностей, например, шагом 2, 8, 16, 32 и так далее, и проходиться нашим окном по каждому из слоев этой пирамиды. Способ долгий по причине отдельного запуска классификатора для разных масштабов на каждом окне. Форму можно

варьировать — самим задать различные размеры, высоту и ширину и также проходить скользящим окном по всему изображению. Если мы хотим одновременно учитывать и форму, и размер, то придётся менять масштаб или окна, или изображения.

Как выбрать наилучшие отклики? Когда в ограниченных окрестностях много раз предсказывается один и тот же объект, используется алгоритм NMS (Non-maximum Suppression). Берем bbox, предсказанный с максимальной вероятностью. Смотрим bboxes, имеющие с ним большой IoU. Выбираем гипер-параметр порог — например, 0.9. Если с исходным bbox, имеющим наибольшую вероятность, IoU больше 0.9, тогда мы откидываем следующий bbox. Продолжаем до тех пор, пока не сойдёмся.

Для какого-то из множественных откликов будет наиболее уверенное предсказание и, ориентируясь на него, можно будет отбросить все отклики в окрестности. Со следующим объектом поступить аналогично, пока не останутся только bboxes, не пересекающиеся с другими по порогу.

5.1.5 Архитектуры нейронных сетей

R-CNN

Теперь перейдём к архитектуре современной нейронной сети R-CNN (Region-Based Convolutional Neural Network). Это двухстадийный детектор. Существует довольно большое семейство моделей, относящихся к R-CNN. У нас есть исходное изображение, далее с помощью алгоритма со стороны мы извлекаем приблизительно 2000 регионов, в которых предположительно находятся объекты. Каждый регион приводим к одному размеру и прогоняем полученное изображение через свёрточную сеть. Выходы сверточной сети направляются в классификаторы. Каждый классификатор говорит, находится ли какой-то конкретный объект в регионе. Когда мы имеем регионы, классы, полученные из классификаторов, то можем сказать наши детекции, т.е. у нас есть локализованное пространство на картинке и относящийся к нему класс.

Метод Selective Search выдаёт около 2000 регионов разного размера и соотношения сторон. Изначально метод использовался для сегментации изображения — попиксельного определения класса, — где итеративно размещаются маски и генерируются предположения об объектах. Там используются форма, цвет и другие характеристики. В настоящее время, когда у нас есть Region Proposals, т.е. наши возможные регионы, мы берем изображение — наш регион, — и вырезаем его (если изображение слишком большое, можем разделить его на куски). Стандартная практика — дополнить изображение до квадрата с помощью какого-нибудь paddinga, привести полученное изображение к размеру 227 на 227 и направить полученные изображения в свёрточную сеть. Нам важно, чтобы размер был фиксированный, т.к. сверточная сеть с фиксированным входом.

Подсчет внутренних признаков. Предпоследняя стадия, на который мы берем наш регион и хотим получить из него какое-то внутреннее представление с помощью сверточной сети. В первой версии, т.е. в R-CNN, исследователи брали предобученную сеть, уменьшающиеся свертки и параллельно увеличивалось количество каналов.

Особенности: Предобучена на ImageNet. Fine Tuning (кусочек сети, который относится к предсказаниям) на N+1 класс (N = 20, N = 200). В итоге нужен 4096-размерный вектор признаков.

Классификация регионов. Мы изменили размер наших регионов и у нас получились тензоры одинакового размера. Пропускаем тензоры через свёрточную сеть, получается embedding, который мы можем дальше пропустить через полноценный слой и получить 4 числа, bboxes. Для определения bboxes можно использовать только 4 числа. Существуют разные подходы — например, можно зафиксировать какой-то из углов, ширину и высоту bbox, зафиксировать центр bbox и также высоту и ширину, либо зафиксировать два диагональных угла. В любом случае нужно 4 числа. Соответственно, выход одной полносвязной ветки будет их предсказывать. Выход второй ветки будет направляться в SVMs, которые решают задачу бинарной классификации. Для каждого класса в нашей выборке свой SVMs.

Следующая модификация, позволившая улучшить качество модели, — это доработка ограничивающих рамок (bbox). На примере видно, что детектор в целом справился с задачей — нашёл лица, но немного сдвинутые. Была идея предсказывать смещение нашего предсказания от реального, размеченного bbox.

Последняя стадия. У нас есть выход сверточных блоков, предобученнных на ImageNet. Берём две ветки: Уточнение bbox, на входе используем карту признаков из CNN, на выходе четыре параметра (dx, dy, dw, dh), модель линейная регрессия; Классификация регионов, на входе используем карту признаков из CNN, на выходе вероятности наличия каждого класса, модель состоит из N линейных SVM

Общая структура. Сеть R-CNN можно разделить на шаги: выделяем регионы-кандидаты с помощью Selective Search. Преобразовываем регионы к одному размеру. Каждое изображение пропускаем через предобученную нейросеть — получаем эмбеддинги. Для каждого эмбеддинга у нас есть N-бинарные классификации. Доработка ограничивающих рамок.

Какие проблемы? 1. У нас есть около 2000 предположений, где находятся объекты с помощью Selective Search. Предположения могут пересекаться. Соответственно, какие-то пиксели изображений мы будем прогонять через сверточную сеть несколько раз, что ведет к избыточным вычислениям. Трюк для сокращения вычислений — взять изображение, прогнать его через свёрточную сеть и только после этого брать полученные признаки. 2. Необходимость масштабирования гипотез. 3. Процедура обучения довольно сложная из-за множества стадий раздельных блоков. 4. Selective Search долго работает.

Fast R-CNN

В ходе попыток разрешения существующих проблем исследователи пришли к новой архитектуре Fast R-CNN. Есть несколько модификаций. Мы берём изображение, сразу прогоняем его через свёрточную сеть, также берём Selective Search, применяем его к исходному изображению, масштабируем гипотезы и накладываем их на тензор, полученный из свёрточной сети. Далее используем pooling-слой, после чего у нас следуют две полносвязных ветки — первая работает для предсказания bboxes, а вторая предсказывает класс. На иллюстрации вы видите полную архитектуру.

Структура модели: подсчёт карты признаков всего изображения подсчёт признаков гипотез на карте признаков с помощью ROI-pooling обучение нейросети с 2-мя типами выходов — классификатором типа объекта и регрессором bbox относительно гипотезы

Все модули обучаются совместно. Можно дообучить наш классификатор, что является преимуществом при работе с необычными данными.

ROI-pooling.

ROI = Region of Interest.

У нас было исходное изображение, которое мы прогнали через свёрточную сеть и пространственная размерность (высота и широта) уменьшилась — допустим, в 8 раз. Нам нужно наложить регионы, которые мы сгенерировали с помощью Selective Search. Это можно сделать, масштабировав bbox: уменьшаем в 8 раз и округляем. Следующий шаг — мы подсчитываем ROI-pooling. Для простоты на нашем примере один канал и пространственное разрешение 8 на 8, исходное разрешение было 256 на 256. Каждый такой пиксель соответствует 16 пикселям исходного изображения. Допустим, у нас была ограничивающая рамка — регион интересов или режим proposal был от нуля до 224. Соответственно, мы можем его сюда перенести. Наш регион интересов в тензорном представлении сверточной сети имеет разрешение 5 на 7, а мы хотим сделать из этого эмбеддинг окно, которое будет содержать 4 числа. Для этого нам нужно сделать ROI-pooling. Мы можем разделить всё это окно на 4 блока — поделить ширину на 2, взять целую часть, в результате один блок будет 3, а другой 4 и по аналогии по высоте мы снова делим на 2, получаем 2,5 и разбиваем 2/3. Например, если бы мы хотели сделать эмбеддинг размера 9, то могли бы разбить этот блок на 9 ячеек, т.е. на 3 столбца и 3 строки. Соответственно, разбиение было бы 2, 2, 3 и 2, 2, 1. После разбиения мы можем выбрать максимум либо применить какой-либо pooling. Самое простое — это максимум, т.е. в первой ячейке 0.85, во второй 0.84, в третьей 0.96, в четвёртой 0.97. Эти 4 числа мы можем вытянуть в вектор и дальше направить в полносвязный слой. Так мы делаем для каждого региона, т.е. есть для каждого региона у нас получится эмбеддинг фиксированного размера и по этой причине мы сможем объединить их в один тензор и дальше направить в полносвязный слой.

Если мы берём просто ROI-pooling, то отображаем наши bboxes, т.е. наши регионы интересов на маленькую матрицу и округляем. Однако может быть неточность — например, если у нас было 0.16 пикселей и 3 пикселя, тогда мы бы округлили вниз и т.д. Из-за такого грубого округления падала точность. В результате появилась следующая модификация, позволившая ещё улучшить модель, — использование ROI-Align, немного похожей на ROI-pooling. Теперь bbox, наш регион интересов, отображается корректно, без округления, т.е. у нас получаются дробные значения. На изображении вы видите рамку — теперь окна будут содержать в себе куски значений. Иначе говоря, мы имеем 0.22, 0.54 и т.д. В данном окне есть 6 значений, в следующем окне есть ещё 6 значений — мы находим элементы тензора, которые попадают к нам в окно, и интерполируем значение всех попадающих в окно пикселей в одно число.

На изображении вы снова видите всю архитектуру Fast R-CNN на немного другой схеме.

На графике отображено изменение скорости работы при использовании разных алгоритмов.

Faster R-CNN

Следующая модель — Faster R-CNN. Модификация была не масштабная, однако позволила существенно ускорить и сделать модель более элегантной. Как мы помним, и в R-CNN, и в Fast R-CNN у нас все еще есть блок, где работает данный алгоритм Selective Search, который мы можем заменить, чем и занимались исследователи. Они взяли Fast R-CNN и добавили некий RPN-блок, расшифровывающийся как Region Proposal Network. Она обучаемая и автоматически генерирует гипотезы. Вся сетка становится единой, начиная от генерации внутренних представлений и генерации гипотез, заканчивая предсказаниями классов и bbox.

Как работает RPN? Мы можем взять нашу feature map, т.е. карту признаков, полученную в сверточной сети и пройтись по ней фильтром 3 на 3. Мы проходимся sliding window, сворачиваем значения, при этом фиксируем, чтобы количество наших окон было 256. Таким образом, для каждого окна мы получим вектор 256. Этот вектор можно дальше направить в две полносвязных сети, классифицировать объект и регрессировать bbox. Существует проблема, связанная с тем, что если мы используем скользящее окно только 3 на 3, то не учитываем разные формы и разные размеры. Соответственно, исследователи добавили так называемые гипотезы, которые назвали «якоря». Эти гипотезы можно определить самому, некоторые варианты изображены на примере. Для каждого окна, «якоря» происходит скользящее окно, и мы для него предсказываем некий объект и 4 координаты для bbox. Улучшение качества происходит за счет учитывания разных размеров и пропорций.

На схеме вы можете подробно рассмотреть, как выглядит архитектура Faster R-CNN.

На следующем изображении отображен процесс обучения одной сети с четырьмя выходами. Обучаем все совместно в multi-task режиме.

Также приведен подробный пример работы модели.

5.1.6 Single Shot MultiBox Detector (SSD)

Поговорим про одностадийный детектор. У нас есть Single Shot MultiBox Detector (SSD). Внутри есть 3 основных пункта, основных частей блока: Base convolutions, дополнительные сверточные слои Auxiliary convolutions и голова Prediction convolutions, которая предсказывает. На изображении вы можете рассмотреть структуру Base convolutions.

Изучим дополнительные сверточные слои. У нас есть выход FM 7, который мы пропускаем через нано-слои. Полоска — это слой, KERNEL 1x1 — это свёртка 1x1, PAD (padding) — то, как мы заполняем, CH (channels) — количество выходных каналов, глубина нашего выходного тензора. Таким образом мы создаем 4 тензора, каждый из них приблизительно в 2 раза меньше, чем предыдущий и имеет разную степень глубины обобщённости представлений. Т.е. чем более глубокая сеть, тем мощнее представление внутри неё, но при этом чем менее глубокий исходный тензор, тем у него меньше потеря пространственной информации. Мы можем рассмотреть разные выходы сверточных блоков — будут разные размерности, но при этом, с одной стороны, чем больше размерность, тем лучше будем схватывать какие-то пространственные паттерны, а с другой стороны, у нас будет лучше понимание взаимосвязей между частями отображения.

Также мы можем задать сами регионы интересов, наши bboxes.

В исходной статье есть feature maps разных размеров. Для каждого из них генерируются priors - это bboxes, которые мы сами задаем и хотим улучшить и определить внутренние классы. Для каждого блока фиксируется какое-то количество priors. Как это можно посчитать — у нас есть пространственное разрешение 38 на 38 тензоров, мы можем перемножить эти значения и дополнительно умножить на 4. 4 — это как раз значение prior. Высокий, но узкий, широкий, но длинный, просто квадрат и квадрат поменьше. Т.к. разрешение уменьшается примерно в 2 раза, то количество гипотез также уменьшается примерно в 2 раза. Каждый prior далее направляется в финальную часть нашей сети, которая предсказывает, с одной стороны, bboxes, а с другой стороны — вероятность классов. Это две отдельных ветки по аналогии с R-CNN моделями. В итоге получится 8732 предсказанных бокса. Очевидно, что это много, поэтому после нам необходимо использовать NMS — Non-maximum Suppression.

5.1.7 You Only Look Once (YOLO)

Следующая архитектура — YOLO. Мы масштабируем изображение, запускаем convolutional network и также используем non-max suppression. Мы делим изображение на квадраты, исходный размер 7 на 7, т.е. 49 клеток. На каждой клетке мы фиксируем несколько bboxes и класс какого-то объекта. Далее мы прогоняем наше исходное изображение через сверточную сеть, берем наши region proposals и через одну ветку, по аналогии с предыдущими моделями, предсказываем сдвиг bbox, а через другую предсказываем класс bbox. После объединяем предсказания.

5.1.8 Дополнительные виды архитектур

Рассмотрим одну из самых современных архитектур EfficientDet. Как мы уже знаем, в SSD мы берем выходы разных сверточных блоков, которые имеют разную пространственную размерность, прогоняем их через две дополнительные сети для головы и каждая голова для конкретного тензора предсказывает класс bbox, который относится к этому тензору, и ограничивающую рамку. На нашей схеме FPN мы видим P от 7 до 3 — это тензоры, полученные друг за другом и имеющие разное пространственное разрешение. С каждым из этих тензоров можно работать отдельно, направив в разные ответвления со своими предсказаниями. Можно усилить действие SSD. Например, взять информацию полученную на последнем слое — самое маленькое разрешение, — и объединить ее с информацией, полученной с предпоследнего слоя. С одной стороны, у нас остается пространственный контекст, а с другой — мощные P-признаки.

Можно рассмотреть данные 5 тензоров и попробовать различными способами передавать информацию между слоями. Есть PANet, которая не просто из самого маленького разрешения передает информацию в самое большое, а еще и делает следующий шаг, передающий информацию обратно, что тоже дает прирост.

Есть некие архитектуры NAS-FPN, которые автоматически подбирают способ передачи информации, чтобы было наилучшее качество. Также есть Fully-connected FPN, где, как в полносвязном слое, информация переходит из каждого в каждый.

DETR. В 2020 году в Facebook решили применить модель трансформеров в компьютерном зрении для детекции. По сути, в этом случае вся архитектура end-to-end, т.е. без

non-max suppression, при этом она одностадийная и работает по аналогии с трансформером для NLP. У нас есть encoder и decoder, исходное изображение этеншены — мы можем определить этеншены для изображений, — и каждый выход декодера направляется в свою ветку, которая предсказывает класс внутри и соответствующий bbox. Это интересно придуманный «чёрный ящик». Обучается медленно, работает с относительной скоростью, однако легко пишется.

5.2 Семинар

Данные для компьютерного зрения — это картинки. А набор данных (картинок) называют датасетом. Создание датасета — не такое простое занятие. Если мы хотим научить компьютер видеть мяч, в датасете должна содержаться информация, по которой наш алгоритм будет учиться этому. Компьютеру понадобится отвечать на вопрос, есть ли мяч на картинке или нет, а также определять его местоположение. Например, рисовать вокруг него прямоугольник. Такой прямоугольник называют bounding box. То есть нам потребуется решить сразу 2 задачи компьютерного зрения: классификации (есть ли мяч или нет) и регрессии (указывать координаты мяча).

5.2.1 Подготовка датасета

У нейросетей различаются требования к формату информации об объектах. Мы работаем с YOLOv5. Ей понадобиться файл разрешения txt для каждой картинки, в котором эта информация об объекте будет указана в таком виде:

Номер класса	x координата левого верхнего угла	y координата левого верхнего угла	ширина

Подготовка реальных данных

В следующих примерах нас будут интересовать 3 объекта (класса): мяч, робот и штанга (ball, rhoban, goal post). Номера классов таковы:

0 – rhoban   1 – ball   2 – goal post

Пример 1

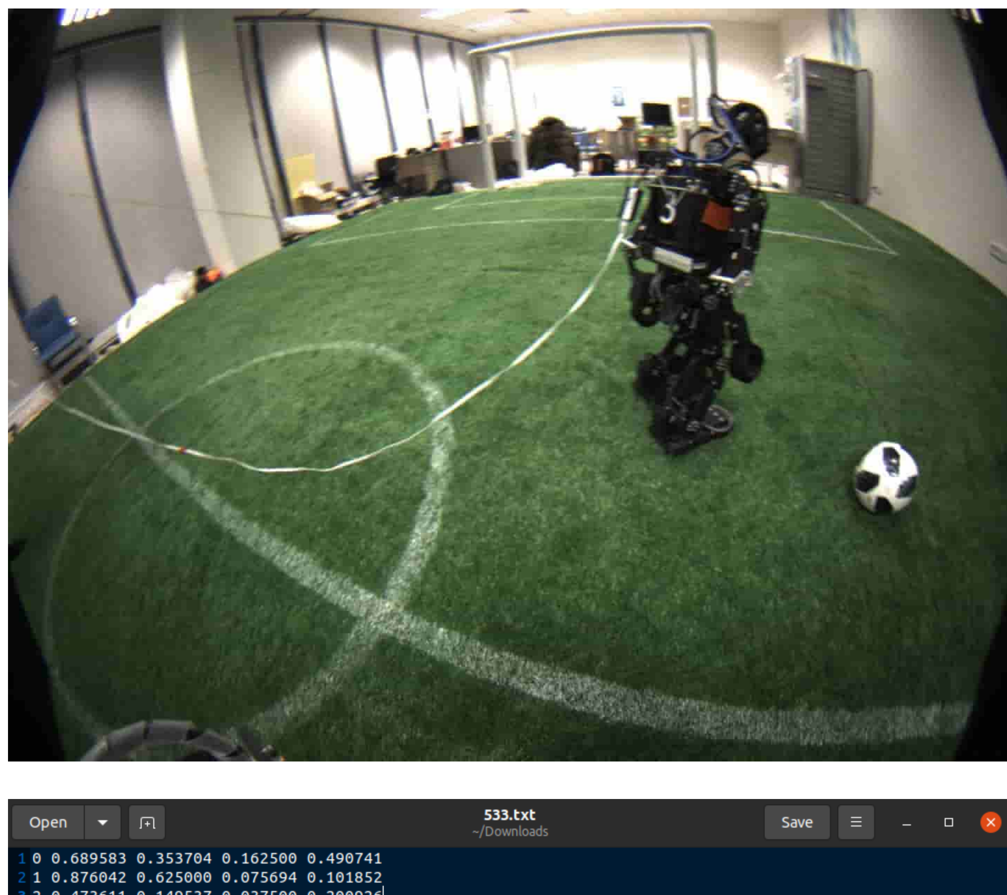
Как можно заметить, каждый из объектов встречается один раз. Поэтому в txt файле 3 строки, каждая из которых несет информацию о соответствующем объекте.

Пример 2

В этом примере мяч и штанга встречаются дважды. В txt файлике мы видим 4 строки, по 2 на каждый класс объектов.

Разметка датасета

В настоящее время используют программы, которые помогают записывать данные относительно объектов. Например, labelImg. Мы работаем с лэйблами формата YOLO. Соответвственно, путем нажатия на нижнюю клавишу из списка нужно выбрать YOLO.



Открываем картинку, нажав на кнопку Open и выбрав нужный файл.

Разметим объекты на картинке, нажатием клавиши w. После выбора прямоугольника для объекта, возникает окно, в котором нужно ввести имя класса.

После того, как мы разметили все объекты на картинке и указали их классы,

сохраняем (Ctrl+S) и смотрим на результат, то есть на созданный нами txt файл, в котором описана наша разметка.

Таким образом размечается целый датасет, причем имена созданных лэйблов должны совпадать с названиями соответствующих фотографий. Кроме того, папка с картинками — images и папка с лэйблами — labels должны лежать в одной директории. Например, data. Либо можно создать отдельную директорию для этих папок.

Сбор датасета из Webots

Помимо тренинга на настоящих данных, существует метод, не требующий бегания с фотоаппаратом и последующей разметки каждой полученной картинки. Современные симуляторы реальности позволяют делать разметку непосредственно в своем искусственном пространстве. Конечно полученные данные отличаются от реальных, но для некоторых задач они подходят. В данном семинаре в качестве симулятора будет использоваться платформа Webots. Программа КОТОРУЮ НУЖНО СЮДА ДОБАВИТЬ

позволяет запускать сбор датасета из построенного вами пространства внутри Webots. Наш скрипт, который собирает датасет находиться в этой директории:

Заходим в директорию с Webots. Собираем проект:

Запускаем мир в Webots:

После загрузки, открывается окно Webots с нашим миром

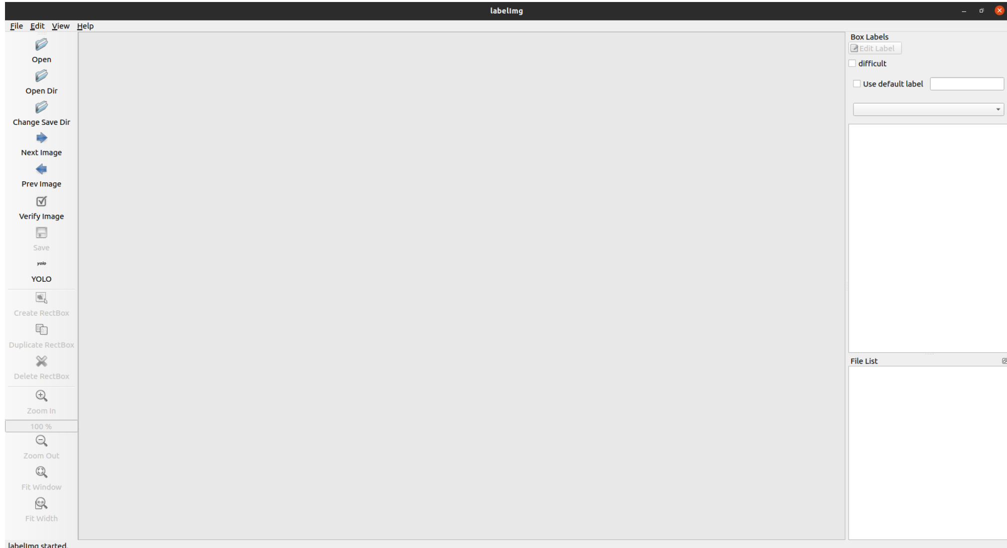
В левом баре нажимаем на DEF RED PLAYER 1 SAHRV74 и переходим в папку controller «player». После чего нужно поменять исполняющуюся программу на seminar dataset controller. То есть мы меняем player на seminar dataset controller с помощью кнопки Select... Снизу:

Проверяем поменяли ли мы controller.

Видим, что поменяли, тем самым наш датасет начал создаваться. Теперь мы можем найти его в директории $/webots_datasets/$

5.2.2 Обучение YOLOv5

После того как мы научились передавать информацию, необходимую для обучения, компьютеру. требуется ее разделить. Ведь алгоритм должен понимать, правильные ли связи в системе он построил при обучении, то есть проверять свои гипотезы и предположения. То есть понадобиться разделить датасет на 3 части: 1) Train 2) Val



(validation) 3) Test Разделить готовый датасет поможет данный скрипт. Причем full.txt – файл, состоящий из путей к картинкам.

Как вы могли заметить, мы делим не сами картинки, а соответствующие им лэйблы. Деление происходит в соотношении 8:1:1 (train:val:test).

После разделения датасета требуется создать файлик с его описанием. В нем нужно прописать пути к картинкам, к лэйблам из трэйна, из валидации и из теста, а также названия и количество классов. Выглядеть он должен таким образом.

Перейдем непосредственно к обучению YOLOv5 на готовом датасете. В некоторых нейросетях заранее предусмотрена такая программа, как train.py, которая автоматизирует запуск обучения. Нам повезло, так как в YOLOv5 она присутствует. Соответственно, все, что требуется для запуска, — написать команду.

Обратите внимание на использованные флаги.

Флаг epochs — продолжительность обучения. Флаг data — путь к файлу, в котором содержатся данные, необходимые для обучения. Флаг weights — путь к претрененным весам, которые мы используем. Флаг project — путь к файлу, в котором будут сохранены результаты обучения. Флаг imgsz — разрешение картинок, которые будет распознавать нейросеть. Однако есть и другие флаги. Они описаны в коде train.py.

Запуск обучения

Теперь мы готовы к запуску обучения, поэтому исполняем команду, описанную выше. Ждем промежуточных результатов. После каждой эпохи подсчитываются такие параметры, как precision, recall, mAp, loss и другие. По ним оценивается успешность обучения, в то время как алгоритм продолжает учиться. Кроме того, с помощью tensorboard есть возможность строить графики, которые также показывают насколько успешно обучается нейросеть. Примеры запуска Tensorboard:

Построение графиков обучения в Colab

Флаг logdir — передаем в него путь, прописанный во флаге — project у train.py. Ведь графики строятся по результатам каждой эпохи. Откроется интерактивное окно, в котором можно будет увидеть графики обучения.

Построение графиков обучения в Ubuntu 20.04

Для перехода к графикам необходимо скопировать выделенный адрес и перейти к нему в браузере. После чего остается только любоваться графиками и следить за обучением.

После окончания обучения мы получаем веса, ради которых весь этот процесс и затевался. Остается проверить насколько хорошо обучена нейросеть. Помимо графиков, которые определяют успех обучения, мы запустим скрипт, который будет автономно показывать работу нейросети в реальном времени.

5.2.3 Подготовка встраиваемого решения и проверка результатов

Данный скрипт позволяет проверить работу обученной нами нейронной сети в реальном времени. Запускаем его и в появившемся окошке наблюдаем за тем, как нейросеть детектирует объекты. Как можно видеть, сетка действительно обучилась. Подведем итоги, мы построили датасет, обучили на нем модель yolov5s, проверили, насколько хорошо она работает. Теперь мы настоящие МАШИНЛЕРНЕРЫ. Ура!

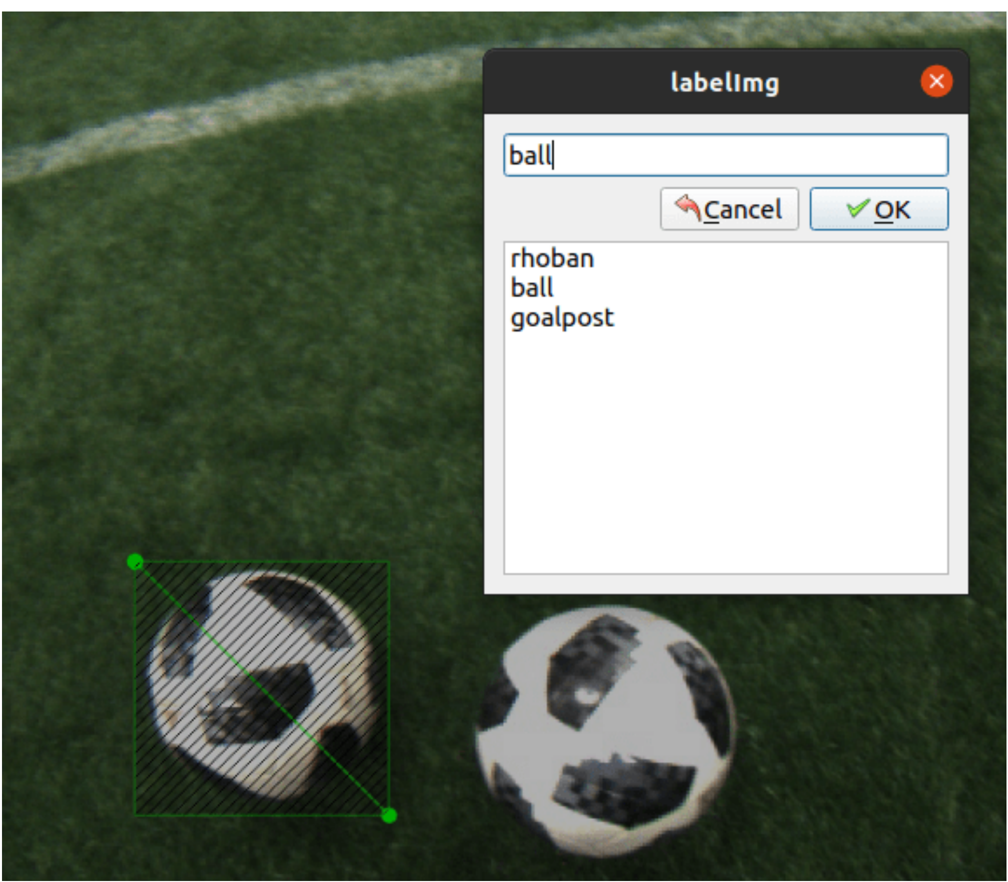
5.3 Практическая работа

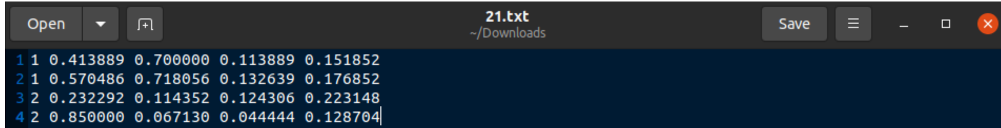
Open

Open Dir

Change Save Dir

Next Image


labelImg
ball
Cancel
OK
rhoban
ball
goalpost

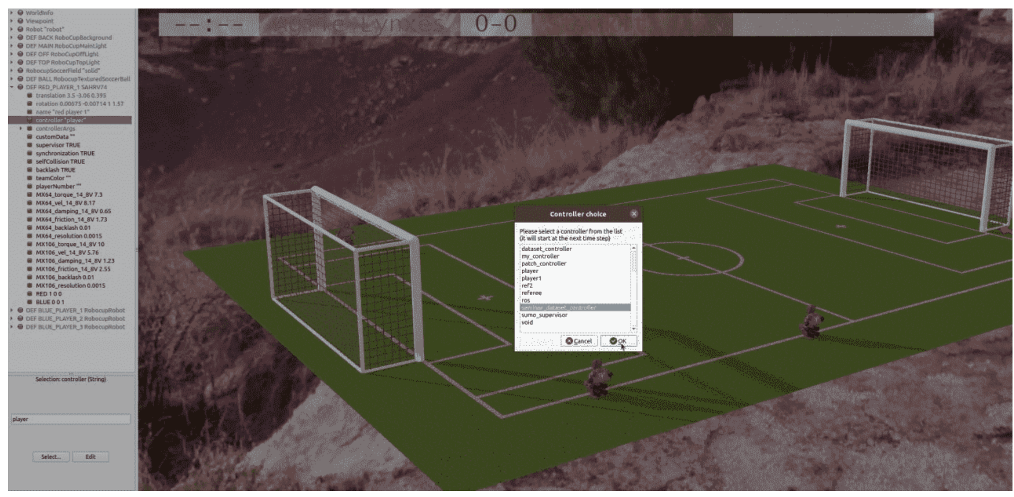


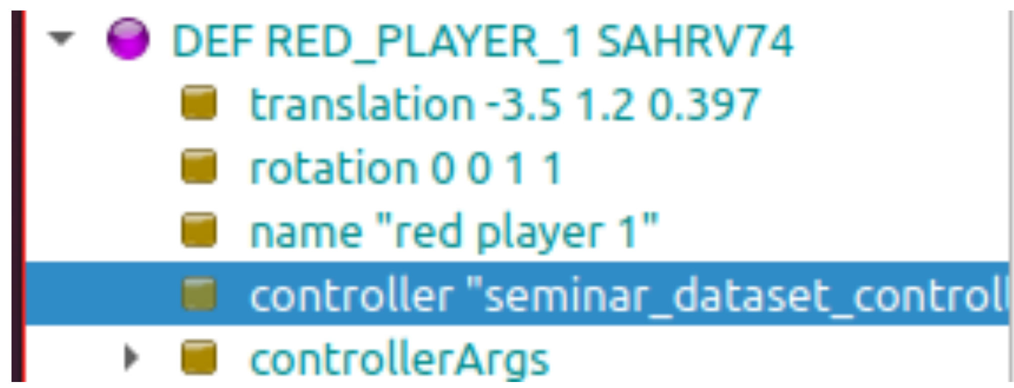

```
~/webots_robocup/webots_controller/controllers/seminar_dataset_controller$ _
```

```
~/webots_robocup$ make -j8
```

```
~/webots_robocup$ ./webots_
```

WorldInfo
Viewpoint
Robot "robot"
DEF BACK RoboCupBackground
DEF MAIN RoboCupMainLight
DEF OFF RoboCupOffLight
DEF TOP RoboCupTopLight
RobocupSoccerField "solid"
DEF BALL RobocupTexturedSoccerBall
DEF RED_PLAYER_1 SAHRV74
translation 3.5 -3.06 0.395
rotation 0.00675 -0.00714 1 1.57
name "red player 1"
controller "player"
controllerArgs
customData ""
supervisor TRUE
synchronization TRUE
selfCollision TRUE
backlash TRUE
teamColor ""
playerNumber ""
MX64_torque_14_8V 7.3
MX64_vel_14_8V 8.17
MX64_damping_14_8V 0.65
MX64_friction_14_8V 1.73
MX64_backlash 0.01
MX64_resolution 0.0015
MX106_torque_14_8V 10



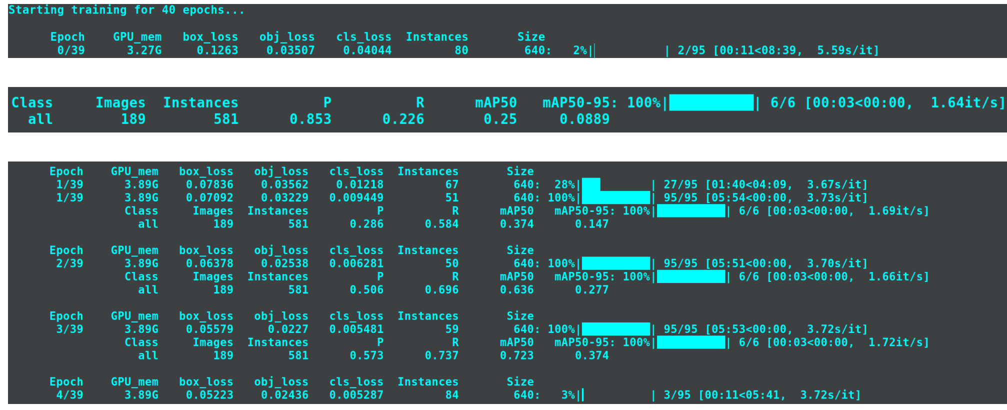

DEF RED_PLAYER_1 SAHRV74
translation -3.5 1.2 0.397
rotation 0 0 1 1
name "red player 1"
controller "seminar_dataset_controll
controllerArgs

```
%cd /content/yolov5
import random

path = "/content/yolov5/full.txt"
path1 = "/content/yolov5/train.txt"
path2 = "/content/yolov5/val.txt"
path3 = "/content/yolov5/test.txt"

full = 3094                              # amount of pictures
for_train = int(0.8*full)                # amount of pictures for train (80 percents of dataset)
for_val = int(0.1*full)                  # amount of pictures for validation (10 percents of dataset)
for_test = full - for_train - for_val    # amount of pictures for test (10 percents of dataset)


#train

with open(path, "rb") as source:
  lines = [line for line in source]
  random_choice = random.sample(lines, for_train)
  for j in range(0, for_train):
    i = 0
    y = full
    while i<=y:
      if lines[i] == random_choice[j]:
        lines.pop(i)
        y -= 1
        break
      else:
        i += 1
  source.close()

with open(path1, "wb") as f:
  print("amount of pictures for train is " + str(len(random_choice)))
  f.writelines(random_choice)
  f.close()

#val

random_choice = random.sample(lines, for_val)
y = full - for_train
for j in range(0,for_val):
    i = 0
    y = full - for_train
    while i<=y:
      if lines[i] == random_choice[j]:
        lines.pop(i)
        y-=1
        break
      else:
        i+=1

with open(path2, "wb") as f:
  print("amount of pictures for validation is " + str(len(random_choice)))
  f.writelines(random_choice)
  f.close()

#test

with open(path3, "wb") as f:
  print("amount of pictures for train is " + str(len(lines)))
  f.writelines(lines)
  f.close()
```

```
path:   # dataset root dir
train: train.txt  # train images
val: val.txt  # train images
test: test.txt  # test images
nc: 3  # number of classes
names: ['rhoban', 'goal post', 'ball'] # class names
```

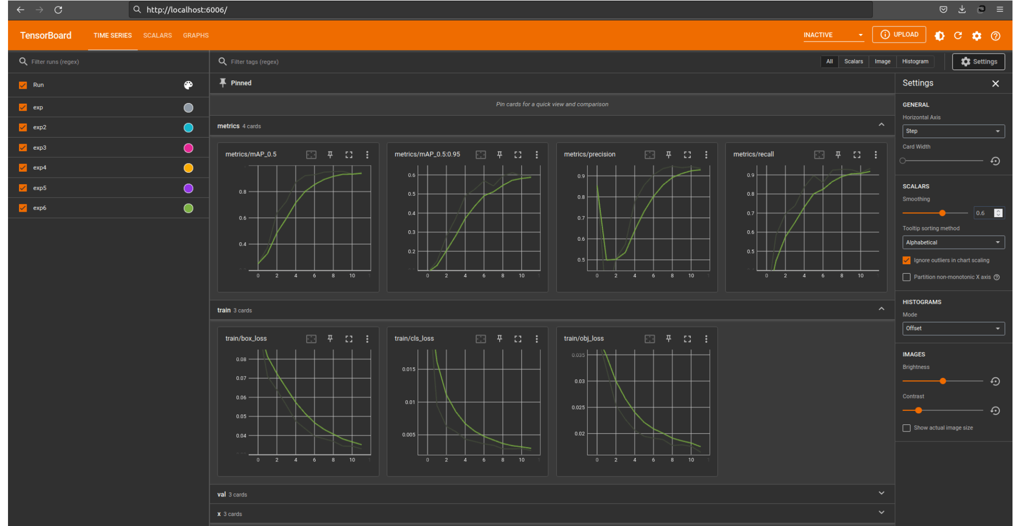
```
!python train.py --epochs 40 --data custom_dataset.yaml --weights yolov5s.pt --project 'results' --imgsz 1088
```

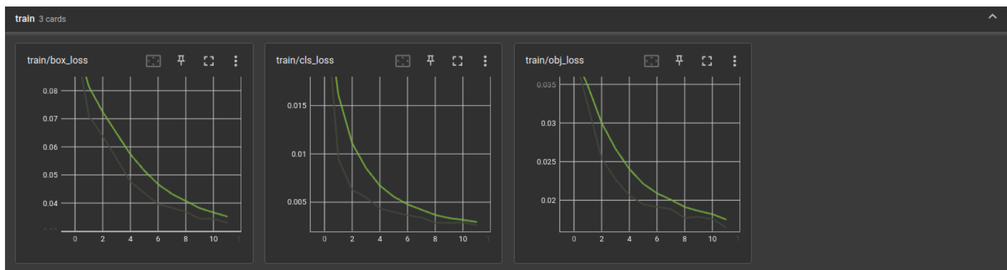
```
parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
parser.add_argument('--epochs', type=int, default=300)
parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs, -1 for autobatch')
parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
parser.add_argument('--rect', action='store_true', help='rectangular training')
parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
parser.add_argument('--noval', action='store_true', help='only validate final epoch')
parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
parser.add_argument('--noplots', action='store_true', help='save no plot files')
parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
parser.add_argument('--name', default='exp', help='save to project/name')
parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--quad', action='store_true', help='quad dataloader')
parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')

# Weights & Biases arguments
parser.add_argument('--entity', default=None, help='W&B: Entity')
parser.add_argument('--upload_dataset', nargs='?', const=True, default=False, help='W&B: Upload data, "val" option')
parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set bounding-box image logging interval')
parser.add_argument('--artifact_alias', type=str, default='latest', help='W&B: Version of dataset artifact to use')
```

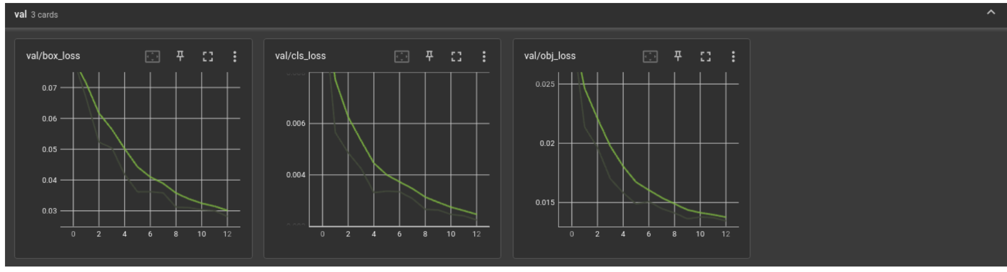


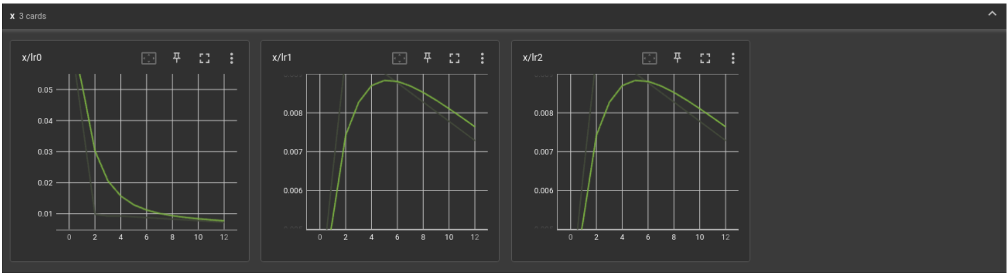

```
# Tensorboard  (optional)
%load_ext tensorboard
%tensorboard --logdir runs/train
```

```
ss21mipt@ss21mipt-OMEN-by-HP-Laptop-15-dc1xxx:~/seminar_yolov5/yolov5$ tensorboard --logdir='results'
TensorFlow installation not found - running with reduced feature set.

NOTE: Using experimental fast data loading logic. To disable, pass
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.10.0 at http://localhost:6006/ (Press CTRL+C to quit)
```


http://localhost:6006/
TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
UPLOAD
Settings
Pinned
metrics 4 cards
metrics/mAP_0.5
metrics/mAP_0.5:0.95
metrics/precision
metrics/recall
train 3 cards
train/box_loss
train/cls_loss
train/obj_loss
val 3 cards
x 3 cards
Run
exp
exp2
exp3
exp4
exp5
exp6

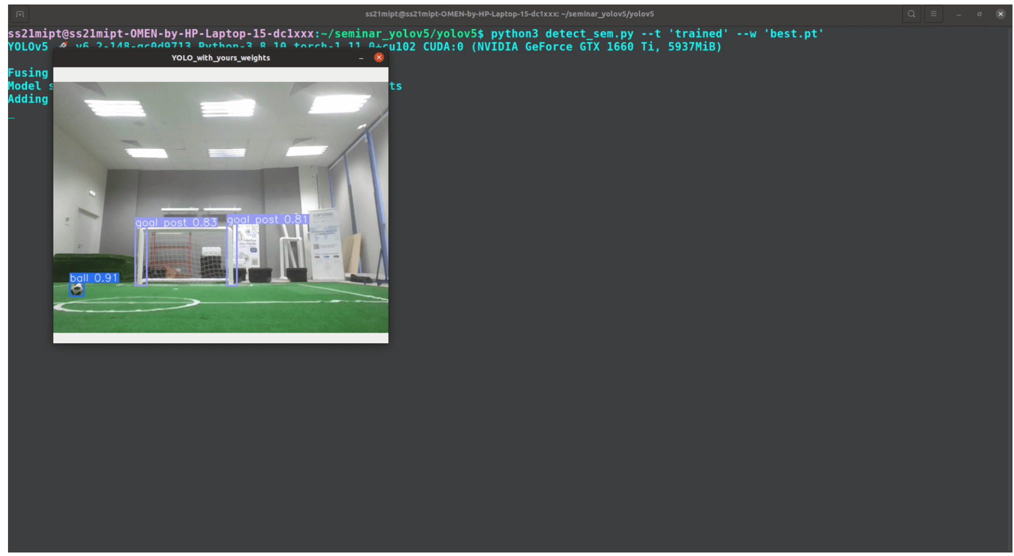

train 3 cards
train/box_loss
train/cls_loss
train/obj_loss


val 3 cards
val/box_loss
val/cls_loss
val/obj_loss

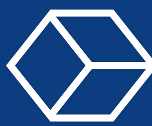

x 3 cards
x/lr0
x/lr1
x/lr2

```python
import torch
import cv2
import argparse

ap = argparse.ArgumentParser()
ap.add_argument('-t', '--type',
                help = 'if weights are pretrained - pretrained. if weights are trained - trained', default='pretrained')
ap.add_argument('-w', '--weights',
                help = 'path to weights', default='best.pt')
args = ap.parse_args()

# Model
if args.type == 'pretrained':
    model = torch.hub.load('.', 'yolov5s', pretrained=True, source='local')    # load a pretrained model (yolov5s) from a local directory

if args.type == 'trained':
    model = torch.hub.load('.', 'custom', path=args.weights, source='local')   # load a trained model (custom) from a local directory
    '''
                                          path='yolov5s.pt')  # PyTorch
                                          path='yolov5s.torchscript')  # TorchScript
                                          path='yolov5s.onnx')  # ONNX
                                          path='yolov5s_openvino_model/')  # OpenVINO
                                          path='yolov5s.engine')  # TensorRT
                                          path='yolov5s.mlmodel')  # CoreML (macOS-only)
                                          path='yolov5s.tflite')  # TFLite
                                          path='yolov5s_paddle_model/')  # PaddlePaddle
    '''
# Images

cap = cv2.VideoCapture(0)                                                  # creates a video capture object, which would help stream or display the video.

while cv2.waitKey(1) < 0:

    hasframe, image = cap.read()                                           # returns boolean (is read() returned something?) and image in numpy.array
                                                                           # format

    # Inference
    results = model(image)                                                 # the impact of the neural network on the picture. It returns an object of
                                                                           # models.common.Detections class. You can look at it here
                                                                           # https://github.com/ultralytics/yolov5/blob/master/models/common.py

    result = results.render()                                              # render() is a method of models.common.Detections class that returns
                                                                           # the result of the neural network in numpy.array format (picture with
                                                                           # predicted labels and confidence of belonging to a class)

    cv2.imshow("YOLO_with_yours_weights", result[0])                       # This method is used to display an image in a window.
```

```
ss21mipt@ss21mipt-OMEN-by-HP-Laptop-15-dc1xxx:~/seminar_yolov5/yolov5$ python3 detect_sem.py --t 'trained' --w 'best.pt'
YOLOv5 🚀 v6.2-148-gc0d9713 Python-3.8.10 torch-1.11.0+cu102 CUDA:0 (NVIDIA GeForce GTX 1660 Ti, 5937MiB)

Fusing layers...
Model summary: 213 layers, 1763224 parameters, 0 gradients
Adding AutoShape...
```



6. Локализация



6.1 Локализация

Постановка задачи локализации, примеры решения задачи локализации в реальном мире, особенности автономных роботов, способы решения задачи локализации, триангуляция, одометрия, фильтр частиц.

6.1.1 Задача локализации

Как было сказано в первой лекции, одно из умений, которым должен обладать **автономный** робот, — умение определять свое положение в мире. В этом и заключается задача локализации. Но что значат слова *положение* и *мир* в данной фразе? Широта и долгота на Земном шаре? Расстояние от какой-нибудь точки, например Северного полюса? На самом деле все это может быть положением робота в мире, все зависит от того, что мы возьмем за точку отсчета. Если нам потребуется понять, где находится робот на планете Земля, то скорее всего мы хотим узнать его географические координаты, то есть широту и долготу, в которых находится робот. Фактически мы делаем это каждый день, а еще наши смартфоны, умные часы, автомобили и прочая достаточно умная техника.

Но все это не совсем автономные системы, ведь для работы gps нам нужна сеть спутников, находящихся на орбите. Более того, погрешность gps не позволяет нам определять положение с точностью до нескольких метров. Поэтому нелогичным будет использовать такой метод ориентирования для робота-пылесоса в обычной жилой квартире. Или если говорить о робофутболе, нам необходима точность до 10-5 см. Получается, что метод определения своего положения с помощью gps нам не подходит, к тому же это потребовало бы от нас установки дополнительного оборудования в виде антенны.

Замечание: *для роботов-футболистов использование датчиков gps запрещено из-за ограничений антропоморфности, ведь у человека таковых нет.*

Для того чтобы искать методы решения данной задачи, опишем ее чуть более подробно. Сформулируем задачу локализации так: необходимо определять свое положение в мире, где под положением подразумевается положение в определенной системе координат, заданной в ограниченном конечном пространстве, которое мы будем называть миром. В случае робота-пылесоса миром будет квартира, а точкой отсчета координат можно, например, обозначить док-станцию. Для роботов-футболистов миром будет футбольное поле с центром координат в центре поля. Также для обоих случаев мы можем ограничиться плоскостью, так как ни роботы пылесосы, ни роботы-футболисты летать пока не умеют.

Вообще говоря, человек никогда не знает и знать не может своего положения абсолютно точно, ни в мире ни относительно какого-либо объекта, но человеку это и не нужно. С самого рождения ребенок учится распознавать объекты, определять насколько они далеко от него и как находятся относительно него. Взрослея человек приобретает способность определять свое примерное положение в пространстве и достигает такого

автоматизма в этом, что даже почти не задумывается в определении своего положения в знакомом месте. Сложности могут возникать в случае попадания в незнакомую среду, например при попадании в новый город, незнакомый район или в более сложную для опознавания среду, такую как лес.

Мы будем рассматривать следующую задачу локализации: наш робот будет определять свое положение на игровом поле. Для решения задачи роботу потребуются косвенные данные, полученные из мира.

Для определения положения в пространстве человек в основном использует ориентиры. В данном случае ориентир мы можем определить как быстро и легко распознаваемый объект, расположение которого мы хорошо знаем или помним. Ориентиры являются объектом зрительного восприятия, таким образом, понятие ориентира можно перенести на компьютерное зрение.

Другим источником информации могут служить данные с положениями сервомоторов. Имея эти данные, мы, зная геометрию робота, можем смоделировать положение робота в пространстве в любой момент времени. Этот тип данных называется одометрией.

Таким образом, мы получаем 2 источника данных для робота — данные со зрения и данные передвижения (одометрия). С помощью этих данных робот и должен определять свое положение.

Итак, у нас есть данные, достаточные для решения задачи определения положения робота. Теперь нужно выбрать метод. Существует много различных способов. Мы рассмотрим самые простые в реализации и понимании.

6.1.2 Триангуляция

Метод триангуляции широко использовался в мире для определения положения тела на земле. Объект, для которого требуется определить его позицию, обладает излучателем и приемником. Излучатель отправляет сигнал на спутники, которые в свою очередь, отправляют ответ обратно. Зная скорость ответа от каждого спутника и положение их в пространстве, спутники образуют сферы определенного радиуса, и на месте пересечения 3 сфер и будет положение объекта. ***Замечание:*** *строго говоря для спутников такой метод называется* ***трилатерицией*** *и реализуется он сложнее. Понятие триангуляции тоже существует, но в нем положение определяется по стационарным станциям с известными мировыми координатами.*

Для робота задача будет чуть проще. Он должен использовать информацию о положении видимых ориентиров, которое заранее известно. Для ориентирования на плоскости достаточно знать расположение 2 объектов. Рассмотри эту задачу для робота на поле, причем ориентирами выступают стойки ворот. Общий принцип работы будет выглядеть так:

- Робот получает информацию о положении ориентира 1 и ориентира 2 из модуля зрения. Координаты объектов передаются в собственной системе координат робота.
- Составляем уравнения окружностей для точек ориентиров. При этом центры окружностей будут лежать в точках ориентиров в глобальной системе координат, а радиусы будут получены из позиций ориентиров в системе координат робота.
- Полученная система из двух уравнений окружностей решается и полученный

ответ будет являеться искомым положением робота. (Так как пересекающиеся окружности могут иметь 2 точки пересечения, то и ответа может быть 2. Но так как робот может находиться только на поле, то второе решение мы можем отмести)

Этот способ является самым простым в реализации и понимании, требует только данные из модуля зрения и при качественной работе последнего весьма точен. Но в нем присутствуют серьезные недостатки. Самый серьезный из них — это то, что нам необходимо минимум 2 наблюдаемых ориентира (в данном случае стоек), чтобы узнать свое положение на поле. А четкое наблюдение сразу двух стоек не всегда возможно как из-за физического аспекта (робот может на них просто не смотреть), так и из-за модуля зрения (в реальном мире алгоритмы компьютерного зрения работают неидеально). Также мы должны требовать, чтобы ориентиры были уникальными объектами. Например, в большинстве «взрослых» лиг робофутбола стойки союзников и противников имеют одинаковый цвет.

6.1.3 Одометрия

Как уже было описано выше, из данных о движении сервомоторов можно получить положение робота в пространстве. Теперь, зная о стартовой точке робота можно построить траекторию перемещения в любой момент времени до текущего. Для этих измерений не требуются измерения с камеры. Но здесь присутствуют недостатки. Первый недостаток заключается в высокой ошибке сервомоторов. Эта ошибка большая из-за того, что на каждую ногу приходится несколько моторов. Так как каждая новая позиция робота зависит от всех предыдущих происходит накопление ошибок. Таким образом, с течением времени робот будет понимать свое положение неправильно. Также мы не сможем считать положение робота достоверным, если он упал и встал так как низка вероятность того, что робот после падения встанет идеально в ту точку и с тем же направлением, как до него. Несмотря на вышеуказанные проблемы, мы не можем не использовать эти данные, так как они все равно несут полезную информацию.

6.1.4 Комбинированный способ

Существенная проблема определения положения триангуляции заключается в том, что для успешного вычисления позиции роботу необходимо наблюдать минимум два ориентира. В случае стоек ворот существуют области поля, из которых робот не может видеть сразу 2 стойки. Также не следует забывать, что модуль зрения не всегда способен точно распознавать объекты, являющиеся ориентирами. Наконец, стойку может загородить вражеский робот или другое препятствие. Таким образом робот будет понимать свое положение не всегда, а только в определенные моменты времени. Однако для полноценного построения стратегии и поведения роботу необходимо постоянно понимать свое положение. Эту задачу можно решить, используя вместе два метода, описанных выше. В те моменты, когда робот распознает 2 и более ориентиров, он использует триангуляцию, а в остальное время он опирается на данные одометрии.

6.1.5 Фильтр частиц

Фильтр частиц — самый сложный метод для локализации и самый затратный к вычислительной мощности. Но также он является самым устойчивым к помехам, падениям и другим игровым ситуациям Общая работа алгоритма состоит в следующем:

1. Стадия инициализации. На этой стадии происходит генерация гипотез о положении робота (частиц) в некоторой окрестности от известного начального положения робота (если мы имеем информацию о начальном положении) или равномерно по полю. Каждая частица обладает своими координатами X и Y и углом.
2. Сбор данных о положении робота, если он двигался. Данные одометрии передаются всем частицам и их положения изменяются. Фактически каждая частица «перемещается» на столько же насколько переместился робот.
3. Получение данных об ориентирах из модуля зрения. Для каждой частицы передаются положения ориентиров в собственной системе координат робота. Положения ориентиров пересчитываются в глобальную систему координат для каждой гипотезы.
4. Вычисление веса для всех частиц. Вес частицы определяется тем, насколько хорошо положение координат ориентиров подходит для их истинных положений. Чем меньше разница положений тем больше вес.
5. Среди всех частиц выбираются частицы с наибольшим весом и оставляются, частицы с меньшим весом удаляются и заменяются на новые случайно сгенерированные.
6. Обновляется положение робота на основании значения частиц с наибольшим весом. Возвращение к пункту 2.

Разберем подробнее все пункты.

- На начальном этапе генерируются гипотезы (частицы). Их количество зависит от условий среды, точности модуля зрения и порога точности. Обычно это число лежит в пределах от 100 до нескольких тысяч. Каждая частица симулирует возможное положение реального робота. В процессе работы алгоритма именно по положению частиц определяется положение робота. Генерация гипотез зависит от начальной ситуации. Если мы знаем точку начала движения робота, то частицы необходимо генерировать в некоторой области в окрестности робота, чем ближе к положению робота, тем больше плотность частиц. Помимо положения, у частиц определяется угол направления, при этом к нему добавляется небольшой шум, такой же, как для координат.
- Робот начинает движение и начинает получать данные одометрии. Эти данные передаются для всех частиц, и они перемещаются на такое же расстояние, на которое передвинулся робот (если робот поворачивается на определенный угол, все частицы поворачиваются на этот же угол).
- Далее идет работа с данными полученными с модуля зрения. Алгоритму на вход требуются положения известных ориентиров в системе координат робота. Эти данные пересчитываются в реальные координаты для каждой частицы. Таким образом каждая частица будет «видеть» положения ориентиров как будто это видел робот.
- Каждой частице присваивается вес в зависимости от того, как хорошо «виденье»

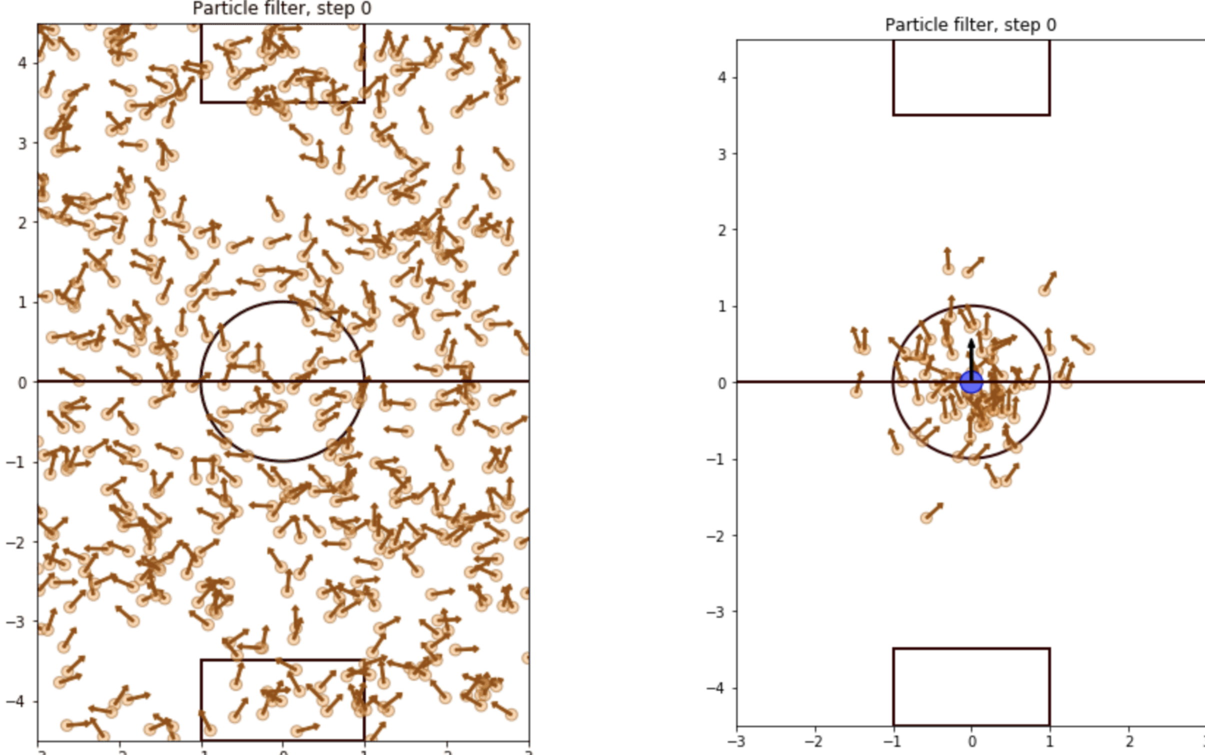


Рис. 6.1: Возможные способы генерировать частицы

ориентиров подходит их реальным положениям на поле. Чем меньше разница положений видимых и реальных положений ориентиров тем больше вес частицы.

- Среди всех частиц выбираются частицы с наибольшим весом и оставляются (то есть проходят отбор, критерий отбора выбирается при реализации алгоритма), частицы с меньшим весом удаляются и заменяются на новые случайно сгенерированные с нулевым весом. Обычно их следует генерировать абсолютно случайно, чтобы в случае различных дестабилизирующих ситуаций их положение могло подойти положению робота.
- На последнем этапе алгоритм проводит вычисление положения робота, основываясь на положении частиц с большим весом. Это можно сделать, например, при помощи взвешенной суммы, в таком случае координата X робота равна сумме координат X всех частиц, умноженных на их вес. То же самое с координатой Y и углом. Теперь робот знает свое текущее положение и цикл можно начинать сначала.

Так как робот действует автономно, ему было бы неплохо как-то оценивать то, как хорошо он определяет свое положение. Такая оценка называется уверенностью. Объясним ее работу на примере фильтра частиц. После получения положения робота мы можем передать на него то же самое «виденье» ориентиров в системе координат робота, которое алгоритм делал для каждой частицы. Чем меньше будет разница между положениями ориентиров для вычисленных координат робота и их реальными положениями, тем

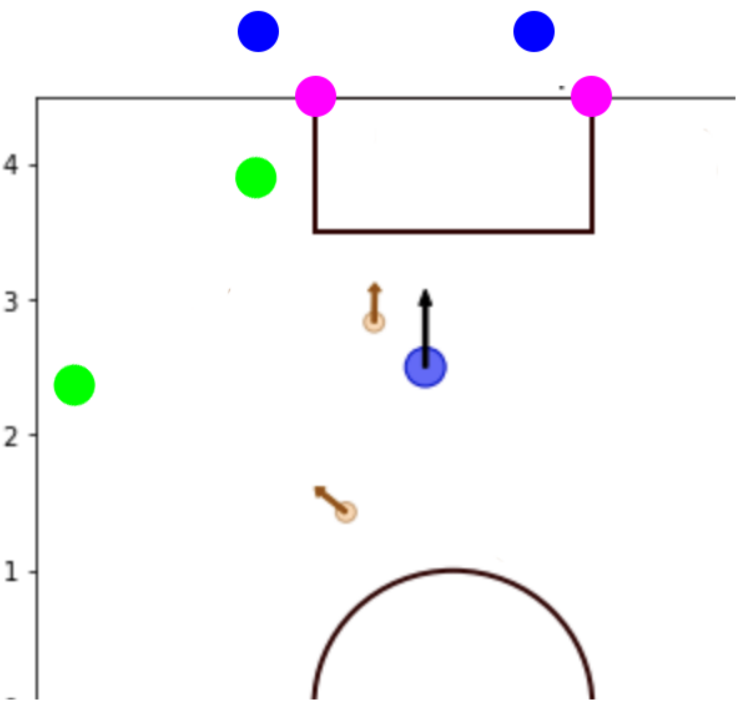


Рис. 6.2: "Примерка"положений ориентиров для частиц. Синий вектор - реальное положение робота, 2 желтых вектора - частицы

больше будет уверенность робота в том, что он локализуется правильно. Этот алгоритм умеет бороться с помехами. Ошибка одометрии нивелируется большим количеством частиц разряженных по облаку вокруг робота. За счет разброса сильно повышается вероятность того, что частица попадет в место реального расположения робота. Например, при падении робот может встать на некотором расстоянии от своего изначального положения и по другому ориентированным. В таком случае шумовой шлейф частиц со случайными положениями и углами может частично подойти новому положению робота и он не потеряет свое положение.

В данной главе были рассмотрены некоторые методы для автономной локализации роботов в пространстве. Конечно в реальности их намного больше. Выбор метода определяется прежде всего набором данных, доступных для анализа и вычислительной мощностью устройства.

6.2 Семинар

Написание модуля локализации. Реализация триангуляции по 2 меткам в пространстве на искусственных данных. Встраивание модуля в симуляцию.

6.2.1 Практическая работа

Для выполнения данного задания потребуется компьютер с симулятором Webots, любой текстовый подходящий для python, редактор/ide и «мир» для симуляции. Скачайте репозиторий `Starkit-Education/AI_track`.

- Откройте файл localization.ipynb в jupyter notebook. Он лежит в папке `seminars/localization/`. Загрузите информацию о ключевых точках с диска и выведите его на экран с помощью следующего кода:

```
    import json
    with open("landmarks.json", "r") as file:
        landmarks = json.load(file)
    print(landmarks)
```

- Теперь сделайте то же самое с информацией об измерениях, загрузите и выведите ее на экран из файла measurements.json
- Возьмите данные по ключу `first_mesurement` из словаря measurements и реализуйте нахождение положения робота через триангуляцию. Для этого сопоставьте координаты стоек в реальном мире, взятые из landmarks.json, с данными из measurements.json. Составьте уравнение пересечения окружностей с центрами в координатах стоек, полученных из measurements.json. Радиусы считаются из теоремы Пифагора, где радиус — это гипотенуза, а катеты — это координаты стойки в собственной системе координат робота. Для этого используйте метод math.sqrt(x) — извлечение квадратного корня. Решением этой системы уравнений будет положение робота в системе координат первой стойки. Теперь из двух решений нужно выбрать такое, при котором робот находится на поле (для координат поле существуют следующие ограничения: $-1.8 \leqslant X \leqslant 1.8, -1.3 \leqslant Y \leqslant 1.3$).
- Возьмите данные по ключу `second_measurement` словаря measurements. Здесь есть «ложное» измерение, которое нам необходимо отфильтровать. Подобные ошибочные измерения имеют шанс появиться ввиду неидеальности компьютерного зрения, к примеру, из-за изменения освещения модуль зрения может распознать стойки ворот в стене или другом объекте за территорией поля. Для того чтобы хоть частично избежать связанных с этим проблем, — требуется фильтровать данные. Реализуйте проверку входящих данных на слишком удаленное положение (те находящиеся слишком далеко, невозможные для поля). Из 3 измерений должно остаться 2.
- Вынесите весь функционал в отдельную функцию, которая должна принимать на вход измерения, фильтровать их, вычислять позицию робота и возвращать ее.
- Откройте Мир `localization_test.wbt` (он находится в папке `worlds`) в Webots и файл Localization.py (находится в папке `controllers/localization_test_controller`), у вас перед глазами будет такой файл:

```
import math
import json

```

```
class Localization():
    def __init__(self, landmarks = "landmarks.json"):
        with open(landmarks, "r") as file:
            self.landmarks = json.load(file)
        self.position = ()

    def update(self, mesurements):
        """
        Здесь будет ваш код
        """
        pass

    def return_postion(self):
        return self.position
```

Вам нужно будет вставить проверку на количество ориентиров, фильтрацию плохих данных, функцию, вычисляющую положение робота, а также вставить обновление позиции робота в конце функции update. Можно писать любые дополнительные функции для использования внутри функции update. **Внимание:** писать код нужно только в файл localization.py, остальные файлы менять не нужно. Если у нас недостаточно данных для высчитывания положения, то мы не должны обновлять значение позиции робота. Учтите, что вы должны проверять, хватит ли нам данных для вычислений. Чтобы проверить работоспособность вашего кода, необходимо перезапустить симуляцию. Код, работающий на роботе будет запущен автоматически.

- При успешной реализации, позиция будет выводиться в консоль симуляции. Проверьте корректность работы высшего модуля, перемещая робота по полю и поворачивая его. Также вы можете щелкнуть на робота и посмотреть в его поле translation, там будут его идеальные координаты, которые высчитывает сам симулятор. Сравните свой ответ с выводом симуляции.

6.3 Практическое занятие

В первом упражнении вид данных со зрения немного изменен. В реальной жизни абсолютно точных измерений не бывает, так как в измерениях всегда присутствует погрешность. В следующем упражнении данные со зрения будут приходить с шумом. Существуют разные способы уменьшить ошибку.

Упражнение 13 В данном упражнении нужно дополнить написанный на семинаре модуль локализации с учетом шума в данных.

- Откройте мир `localization_test_noise` и проверьте работу написанного модуля в нем. В этом мире данные подаются на вход нашему модулю не каждый такт

цикла, а собираются в массив и подаются на вход с определенной регулярностью.
- Добавьте фильтрацию данных, основанную на выделении наиболее часто встречающихся координат (те координаты, которые чаще встречаются - с большей вероятностью относятся к истинным данным).
- Добавьте сохранение истории положений, дополните данные о положении временными метками (модуль python time).
- Также нам хотелось бы получать угол поворота робота в глобальной системе координат. Это можно получить зная координаты робота в мире и координату ориентира в его системе координат.

■

Упражнение 14 `Продвинутый уровень`. Реализуйте алгоритм фильтр частиц для локализации. Для упрощения будем считать что робот стоит на месте.

Для реализации алгоритма потребуется написать несколько важных функций внутри класса `Localization`:
- Функция генерации гипотез
- Функция оценки качества гипотезы
- Функция отсева
- Функция нормализации весов

Для тестирования работы написанного алгоритма используйте мир `localization_test_noise`. Используйте написанную часть кода из прошлого упражнения для выбора лучших ориентиров.

■

Определение

После того как каждой частице был поставлен в соотвествие вес, нужно провести процесс отсева. Отсев нужен для того, чтобы отсеять частицы с низким весом, потому что они являются не подходящими под измерения гипотезами. Вес отсеиваемых частиц мал, а это означает, что вероятность расположения робота рядом с этими частицами пренебрежительно мала. Так зачем же нам и дальше моделировать движение и измерения этих частиц, если мы уже решили что они не соответствуют положению робота — намного эффективнее будет заменить их на новые частицы, которые повысят точность оценки состояния объекта (в данном примере — координат робота).

Суть отсева в том, чтобы из массива N частиц составить новый массив из N частиц, в который войдут только частицы с наибольшим весом. Если быть точнее, то каждая частица из первого массива переходит во второй (переживает отсев) с вероятностью равной её весу.

Пример алгоритма «Колесо отсева», который можно использовать для отсева частиц:

```
#пример реализации колесса отсева
index = random.randint(0, 6)
betta = 0
for i in range(N):
        betta = betta + random.uniform(0, 2*max(weight))
        while betta > weight[index]:
                betta = betta - weight[index]
                index = (index + 1)%N # индекс изменяется в цикле от 0 до N
        newParticleList.append(particleList[index])
particleList = newParticleList


```

Для упрощения разработки и отладки можно написать алгоритм в jupyter notebook с применением библиотек opencv или matplotlib. С их помощью можно схематично нарисовать поле, на котором находится робот (вид сверху), а также отобразить положение гипотез и их поведение под действием алгоритма. Так как гипотеза включает в себя координаты и угол, схематично это можно представить как кружок-вектор направления. На **??**

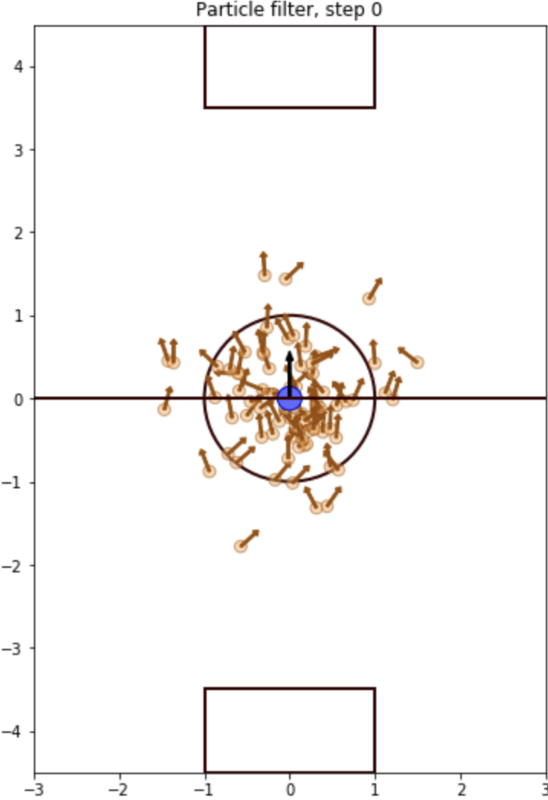


Рис. 6.3: Пример отрисовки с помощью matplotlib

6.4 SLAM

Методы локализации, про которые мы говорим, можно разделить на два типа: относительные и абсолютные. К относительным относится аппаратная одометрия, про которую написано выше, и визуальная, про которую будет написано далее. К абсолютным относятся известный уже нам фильтр частиц и пока еще неизвестный SLAM. Относительные и абсолютные методы различаются тем, что в первом случае робот измеряет свое положение относительно своей стартовой позиции, а во втором — относительно каких-то еще объектов. В первом случае карты нет, во втором она есть. Таким образом, далее будет говориться о таком методе, который позволит нашему роботу ориентироваться в неизвестной местности. Этот метод называется сламом — simultaneous localization and mapping, что переводится как одновременная локализация и построение карты.

6.4.1 Визуальная одометрия

Рассмотрим понятие визуальной одометрии. Для аппаратной одометрии использовались положения сервоприводов, датчики давления, гироскопы и акселерометры. Для визуальной одометрии используется камера.

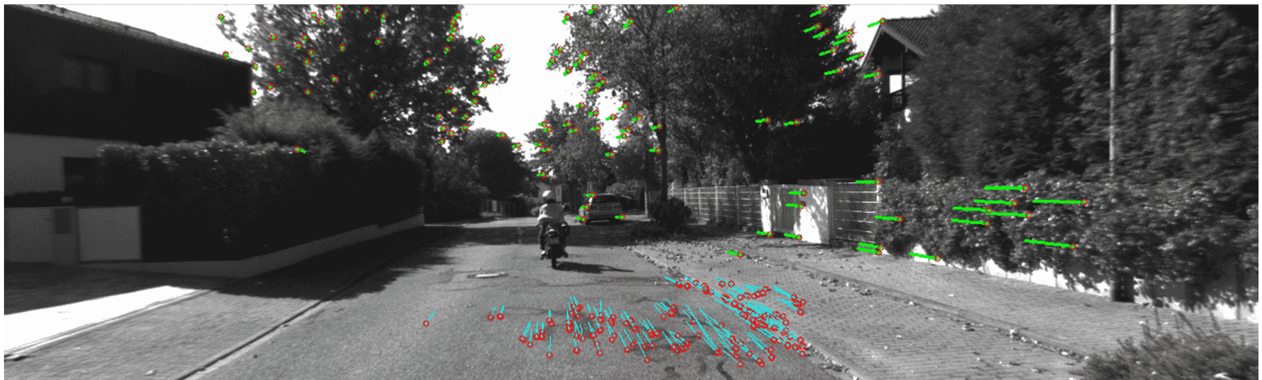


Рис. 6.4: Визуальная одометрия

Давайте представим себя человеком в движущейся машине. У него нет данных о том, как и с какой скоростью крутится каждое колесо машины, чтобы посчитать апаратную одометрию и построить свою траекторию. При этом, глядя в лобовое стекло автомобиля человек вполне может понять, куда, с какой скоростью и как он движется и в принципе может воссоздать свою траекторию. В данном случае человек решает задачу визуальной одометрии. Идея ее достаточно проста. На последовательных кадрах идущих друг за другом мы смотрим как сместились одни и те же точки. На картинке видны траектории смещения одних и тех же точек при продвижении вперед. Точки в нижней части картинки движутся сверху вниз, в правой части картинки — слева направо. Более наглядно это можно выразить так: на первом кадре канализационный люк находится дальше, на последующих ближе — из этого можно сделать вывод, что мы переместились по направлению к нему. У человека есть два глаза, т.е. две камеры,

и он, даже не задумываясь, решает эту задачу. Робот со стереокамерой тоже может ее решить, но что делать, когда камера одна? В таком случае решить напрямую задачу не получится, это на плоском поле мы можем посчитать расстояние от камеры до объекта, а объемном мире робот не знает геометрии всех объектов, поэтому и расстояния до них определить не сможет.

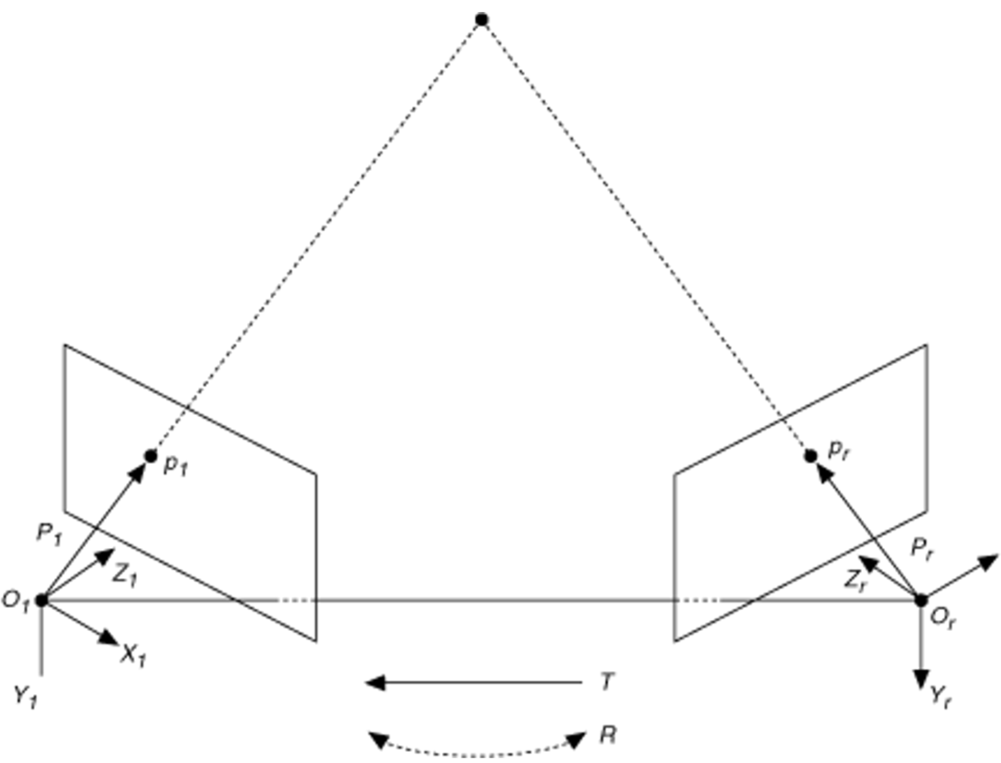


Рис. 6.5: Стереопара

На картинке мы видим две камеры или стереопару с двумя кадрами, разнесенные на некоторое расстояние. Если стереокамера у нас на работе установлена, то задача визуальной одометрии решается практически «в лоб». Это возможно благодаря тому, что мы из стереопары можем получить расстояние до точки. Расстояние до точки может быть найдено из-за того, что в стереопаре камеры разнесены в пространстве, и, таким образом, у одной камеры есть смещение относительно другой. На самом деле это смещение можно получить и с одной камеры. Если мы наблюдали одну точку с одной позиции камеры, а потом переместились и снова наблюдаем эту самую точку, то это точно такой же процесс, как если бы мы использовали не одну камеру, а две. Другими словами стереопару мы можем симулировать, перемещая нашу единственную камеру. Сравнив кадры с одними и теми же объектами, но снятыми на разных расстояниях, мы можем сделать вывод о перемещении этой камеры.

Это приводит нас к понятию оптического потока. Это то, как точки смещаются от кадра к кадру. Визуально это можно изобразить так. Вот у нас есть изображение кружка под номером 1, и он повернулся на картинке под номером 2. Оптический поток для этого

кружочка изображен на картинке 3. Здесь каждый векторочек означает то, как соответствующий пиксель на кружке переместился. Картинки сверху изображают перемещение самого объекта. Для движущейся камеры виды оптического потока нарисованы снизу. Для камеры летящей строго вперед на какой то удаленный объект, предположим что мы летим в космосе, центр изображения будет оставаться неподвижным, а по краям векторы будут двигаться по направлению к ближайшим краям кадра, как на той картинке 2 слайда назад, где точки двигались от центра. Если камера будет вращаться, то оптический поток будет выглядеть на картинке 5. Человек может это понимать и сам, а вот робота нужно научить понимать оптический поток. Еще раз, если можем посчитать как у нас

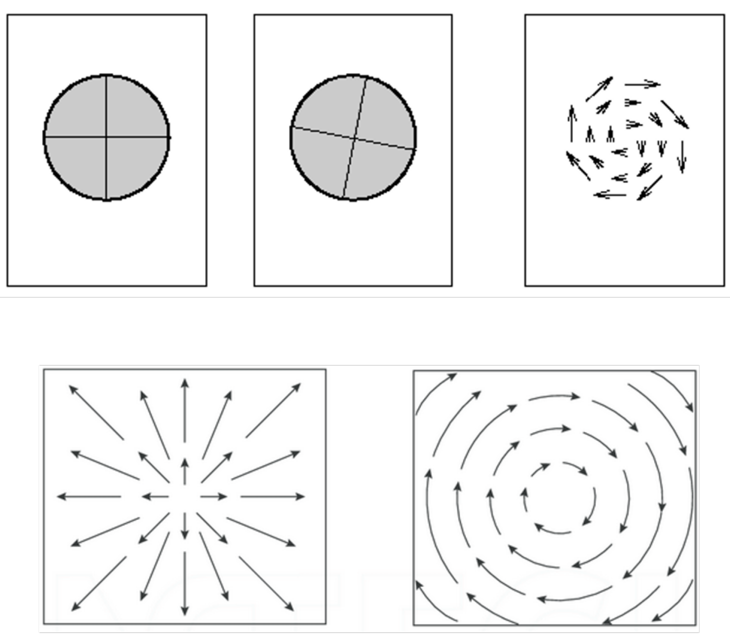


Рис. 6.6: Оптический поток

сместились какие-то кусочки изображения, то, оценив характер этого движения через модель камеры, мы можем оценить как камера в пространстве перемещалась

Для того чтобы понять траекторию перемещения камеры, нам необходимо несколько кадров и одно ключевое условие — мы в кадре постоянно видим одни и те же точки, по крайней мере, на смежных кадрах, чтобы понять, как мы перешли от одного положения к другому, анализируя положение одних и тех же точек.

В итоге напрашивается некоторая система понимания, куда мы сместились, которая состоит из 3 частей: в первой части мы захватываем изображения с камеры и раскладываем их во временной ряд. Дальше на этих изображениях понимаем какие кусочки изображения куда сместились. И в последней части строим соответствующую модель камеры и понимаем траектории движения камеры.

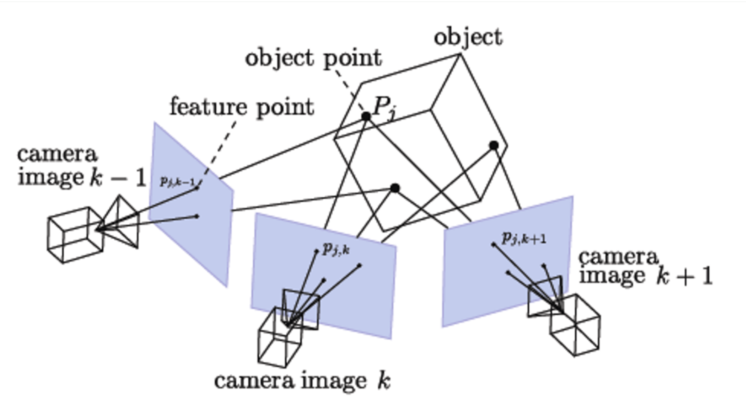


Рис. 6.7: Стерео на одной камере

6.4.2 Дескриптор особенностей

Особые точки

Разберём понятие особенности изображений. Если мы выберем однотонный кусок изображения, например, часть стены, то, перемещаясь относительно этой стены, даже человеческим глазом мы не поймем, что происходит по причине неизменности серого цвета. Невозможно различить, на какую именно часть стены мы смотрим. Однако у нас есть какие-то объекты - с человеческой точки зрения достаточно очевидные, - и мы различаем карандаш, вазу, картину и на одном кадре, и на другом. Мы понимаем, как эти объекты смещаются. Нам достаточно найти некоторые кусочки изображения, которые выглядят настолько характерно, что при их смещении из-за передвижения камеры мы можем найти их на любом кадре.

Такие кусочки называют особыми точками. На изображении видна работа детектор углов Харриса. Он обычно называется детектором углов, потому что отлично цепляется именно за углы. Выглядит это следующим образом: если в качестве анализируемого кусочка изображения выбрана однотонная область, не имеющая в себе переходов, то куда бы она не сместилась, мы это не поймем. Если в качестве части изображения выбрано ребро, то мы можем понять, когда оно смещается влево-вправо, но не можем понять, когда оно смещается вверх-вниз. Если выбран угол, то мы можем понять, что он смещается влево-вправо, вверх вниз, потому что будет сильное изменения внешнего вида этой особенности.

Детектор углов Харриса - это классический метод нахождения точек, за которые удобно зацепиться и впоследствии по ним понимать, откуда они сместились и куда сместилась камера. Но у самых простых решений всегда есть проблемы. Например, детектор углов не работает с различными масштабами. Алгоритм будет распознавать один и тот же объект на разном расстоянии по разному. Это один из самых весомых минусов, портящий обнаружение особенностей и для визуальной одометрии.

Рис. 6.8: Особенности на изображении

Дескриптор

Рассмотрим более сложную модель описания особенностей изображения - дескриптор особенностей. Для него мы рассматриваем не одну точку (как в детекторе углов Харриса), а группу точек на изображении. Эта группа точек будет определяться двумя характеристиками: направлением и характерным размером. Для примера мы рассмотрим детектор особенностей SIFT (scale invariant feature transform). Ключевая идея данного метода содержится в нескольких сущностях. Допустим, у нас есть изображение кота и мы будем его размывать. В SIFT это размытие происходит с помощью преобразования Гаусса - по кривой Гаусса мы размываем интенсивности пикселей стороны. Делаем это по одному и тому же изображению несколько раз. Видим, что изображение становится постепенно все более размытым. Справа видны отдельные участки: кусок корки с усами. Видно, что при переходе от первого кадра к последнему усы практически исчезли вследствии размытия. Значит, усы - это некоторые особенности изображения, причём достаточно маленькие, так как они исчезают первыми. Кажется, что усы - такая тонкая структура изображения, за которую можно зацепиться при понимании, что и куда переместилось. Однако это особенности маленького масштаба. А глаз и на первом, и на последнем изображении выглядит примерно одинаково, поэтому он - особенность достаточно большого масштаба. Мы сможем определить его положение при достаточно большом смещении, что очень удобно.

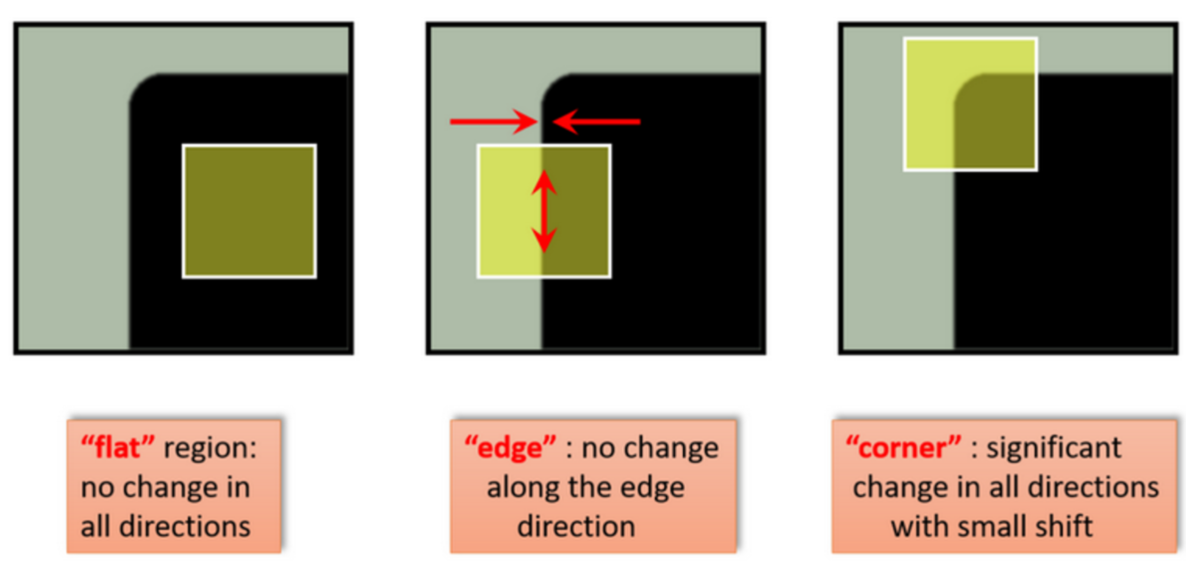


Рис. 6.9: детектор углов Харриса

.

Рис. 6.11: Размытие с целью получения характерного размера

Идея метода SIFT в плане детектирования особенностей заключается в том, что разные части изображения с разной скоростью исчезают при постепенном размытии изображения. Насколько быстро исчезают кусочки, настолько большие они по размеру и настолько большой вклад они дают в составные части изображения в плане особенностей.

Как нам это более алгоритмизировать? Во-первых, давайте разобьем изображение на пирамиду масштабов. Мы будем размывать исходное изображение, уменьшенное в 2-4-8 раз и т.д. Это так называемые октавы. Мы будем следить за тем, какие кусочки изображения насколько быстро исчезли, перестали быть выделяемыми при размытии на разных масштабах. Понятно, что на большом изображении размытие усов спустя несколько кадров будет значительным, а на маленьком уже размытие глаза спустя те же самые несколько кадров будет значительным. Если мы поймем, на каком масштабе какой кусочек заметно размылся, то сможем оценить его характерный размер. Как это сделать? Мы можем вычитать последовательные размытия изображения одно из другого.

На примере заметно, что на данном масштабе нет практически никаких изменений во внешнем виде изображения. Следующий этап размытия уже показал какие-то результаты. На изображении видна некоторая ненулевая часть, какие-то изменения, значит в данном месте в изображении содержатся некоторые особенности, которые имеют такой-то масштаб соответствующей ступени размытия. Далее мы вычитаем сделанное на предыдущем шаге и ищем локальный минимум максимума. Это детектор краев Кенни, описанный в лекциях по классическому компьютерному зрению. Мы получаем точки (локальный минимум, либо максимум) – это и есть наши искомые особенности, за которые можно потом цепляться, понимать, как они смещаются и определять, например, положение камеры.

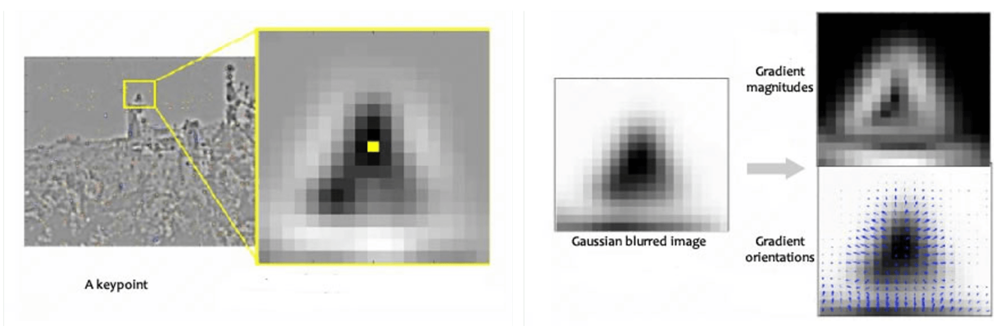


Рис. 6.12: Градиенты

Другой важной характеристикой является ориентация особенностей. Детектор особенностей SIFT дает нам знания об ориентации особенностей. Ранее на лекциях рассматривался детектор краев Собеля и описывалось, что при продифференциировании изображения по горизонтали и по вертикали и последующем сложении, через амплитуду этой разницы мы можем понять яркость перехода. А через фазу изменения яркости понять ориентацию ингредиентов в данной точке. Именно это делается в методе SIFT. Мы берем наше размытое изображение оператором Собеля, получаем значение его градиента и по формуле определения фазы по двум компонентам можем понимать ориентацию градиента в каждой точке. Там, где мы видим перепад сверху вниз, стрелочки будут направлены сверху вниз. Там, где виден перепад по диагонали, соответственно, стрелочки направлены по диагонали. Для нас главное то, что на выходе мы получаем направление, т.е. ориентацию особенностей.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

Рис. 6.13: Размытие с целью получения характерного размера

Дескриптор особенности. Далее мы берем нашу особенность и очерчиваем вокруг неё некоторую рамочку, которая ориентирована по той ориентации, которая вычислена

на предыдущем шаге и имеет размер, который соответствует характерному масштабу, вычисленный на позапрошлом шаге. Тем самым мы получаем новое изображение, где каждый квадратик - это пиксель. В данном случае берется кусочек 16 на 16 пикселей. В этом окне мы снова находим направление градиентов, но разбиваем их на отдельные корзинки - области размером 4x4. Берем левую верхнюю четверть - она содержит 16 направлений градиентов, мы сводим их в одну точку и строим гистограмму градиентов. Так мы делаем для всех кусочков 4x4 и получаем некоторое распределение градиентов в каждой клеточке. У нас 16 возможных направлений и всего этих клеточек тоже 16. Таким образом мы можем получить 128-размерный вектор, т.е. набор из 128 чисел, каждое из которых описывает длину векторов-стрелочек. Всего 128 стрелочек. 128-размерным вектором мы можем отлично описать эту особенность. Если на изображении несколько особенностей, то для них 128-размерное описание будет разное.

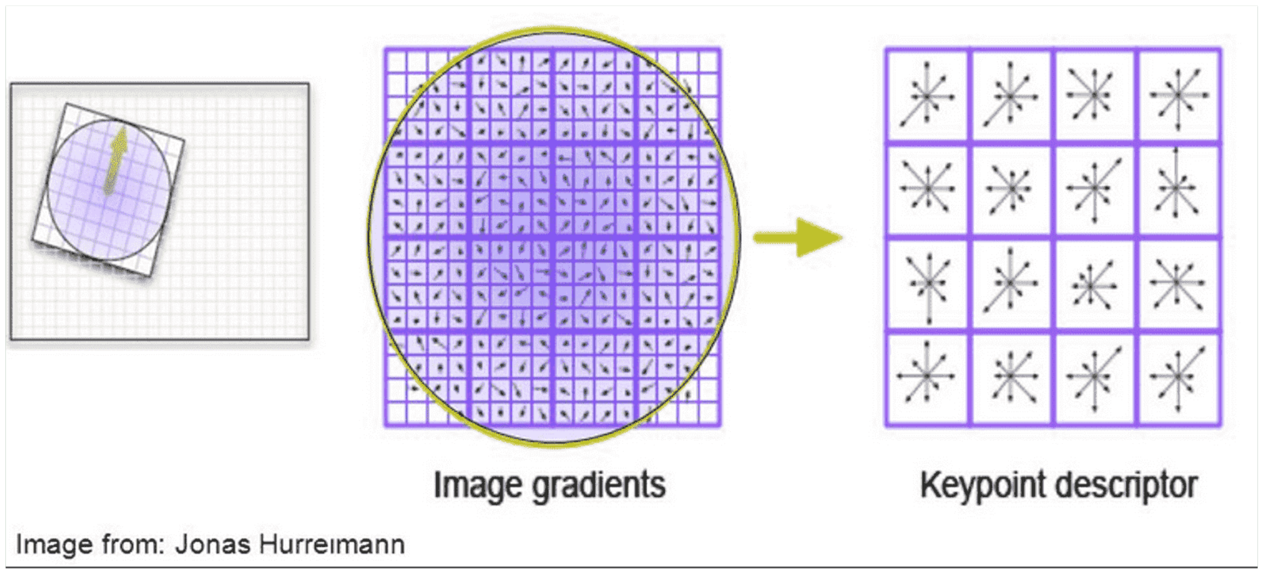


Рис. 6.14: Размытие с целью получения характерного размера

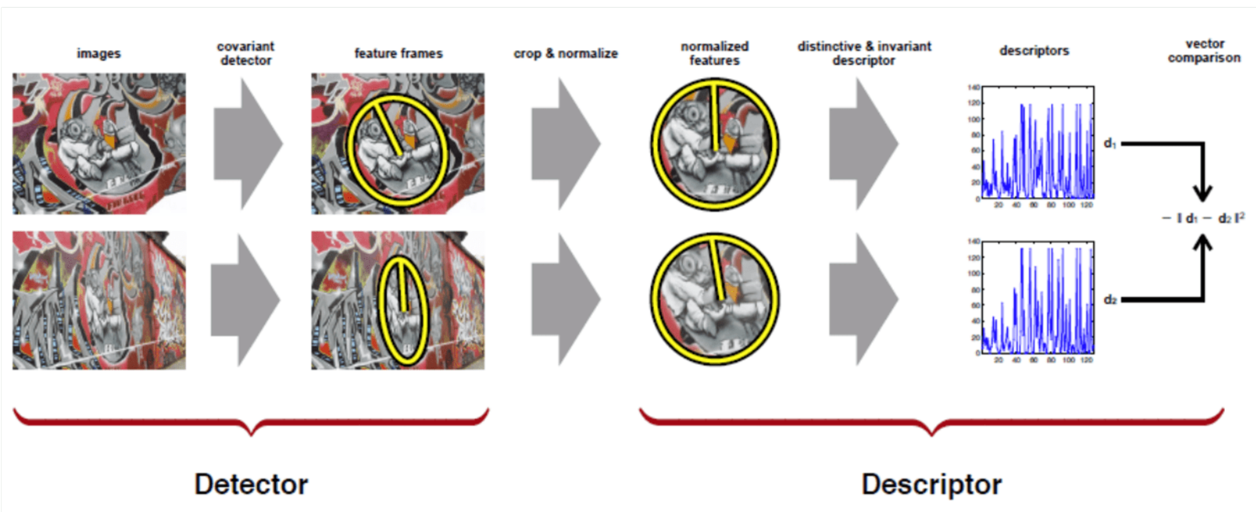


Рис. 6.15: Размытие с целью получения характерного размера

Что нам это дает? Есть некоторые изображения - допустим, изображение граффити. Постепенно его размывая, мы можем сказать, что выделенный кусочек изображения имеет определенный характерный размер, которым мы описываем размером этой окружности и имеет некоторое характерное направление, взятое из предыдущей математики. Мы строим 128-размерный вектор по этой особенности и получаем его - 128 чисел. После берем другое, совершенно новое изображение и также находим на нём особенности, поднимаем их масштаб и ориентацию, выравниваем и опять получаем наш 128-размерный вектор. Случается некоторая магия - для одного и того же кусочка, взятого с разных изображений, под разным масштабом и под разным углом, в любом случае 128-размерный вектор будет примерно одинаковый. Он будет описывать внешний вид этого кусочка, невзирая на масштаб. Кусочек может быть побольше или поменьше, как-то повёрнут, потемнее или посветлее, но его характерный вид и размер будет определены.

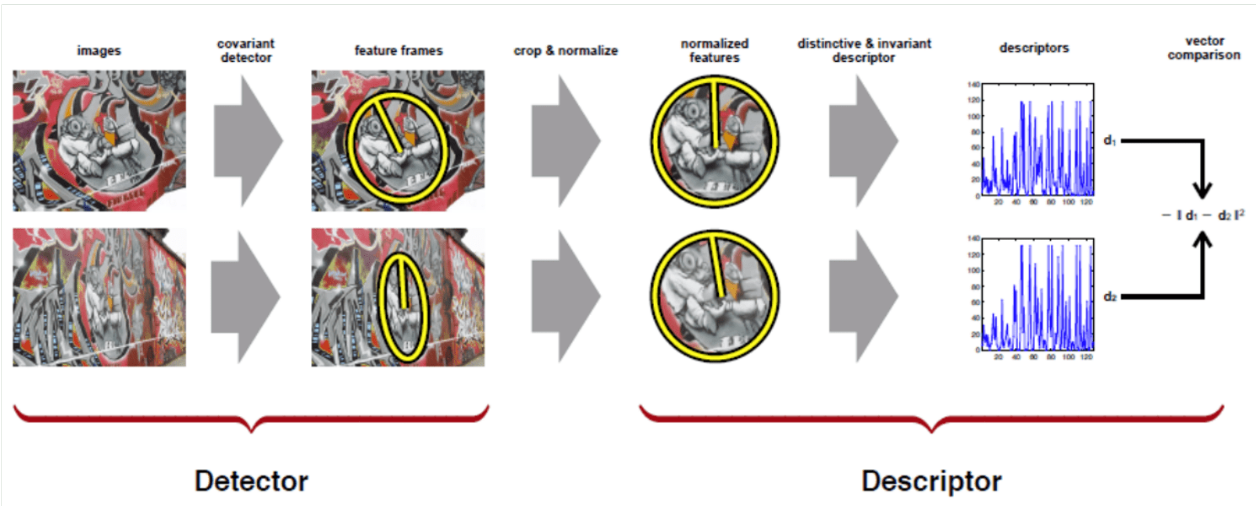


Рис. 6.16: Размытие с целью получения характерного размера

Это позволяет решать задачу нахождения каких-то частей изображения на большом изображении - не отдельных особенностей, а некоторых больших кусочков. Эти куски составлены из большого количества мелких деталей, где каждая деталь - наша особенность. Если мы посчитаем дескрипторы для каждой из этих особенностей и потом попробуем поискать эти дескрипторы на большом изображении, то получим более или менее соответствие - дескрипторы, соответствующие предметам слева, расположены на картинке справа. Но нам это нужно не для поиска картинки в картинке, а для понимания, какая особенность куда сдвинется при смещении камеры.

Рис. 6.17: Нахождение частей изображение

Визуально-инерциальная одометрия

Существует проблема - если мы определяем, как смещается камера исключительно по изображению, то появляется некоторая неоднозначность. Мы не можем однозначно сказать, произошел сдвиг или поворот камеры. Особенно, если у камеры все объекты находятся примерно на одном и том же расстоянии относительно нее. Т.е. смещение объекта может быть обусловлено как смещением самого объекта, так и смещением камеры. Поэтому если мы хотим понять, как смещается камера только по изображению с неё, то могут возникать неоднозначности. Например, неоднозначности разграничения поворота или смещения, если сцена является неудобной для визуальной одометрии, если она не содержит объектов разной глубины.

Поэтому существует визуально-инерциальная одометрия, когда на работе стоит не только камера, но и инерциальное измерительное устройство (IMU). Данные с IMU мы можем использовать одновременно с данными с камеры. Инерциальная система отсчета - это, прежде всего, акселерометр и гироскоп. Гироскоп реагирует на изменение своей ориентации, а акселерометр реагирует на линейные ускорения. В качестве примера: по изображению с камеры мы не очень понимаем, камера повернулась или сместилась, а по данным гироскопа и акселерометра понимаем это достаточно неплохо. Если камера повернулась, то у нас будут большие значения для гироскопа и маленькие значения с

акселерометра, а при смещении наоборот. Тем самым, совместив данные из изображения и инерциальной системы отсчета, мы можем сделать более качественную одометрию. Она уже будет называться визуально-инерциальной одометрией.

6.4.3 SLAM

Далее подходим к сути метода одновременной локализация и построения карты. Мы уже узнали, что визуальная одометрия - это метод, позволяющий по потоку изображение с камеры определить траекторию движения камеры. Также мы можем запоминать, какая особенность в каком месте была расположена на изображении при смещении камеры. Мы можем ставить точку в этом месте при возникновении новой особенности и построить карту из этих точек вдоль траектории. Еще мы знаем, что у нас со временем идет постепенная потеря точности положения и чем дольше мы занимаемся таким измерением своего положения, тем хуже качество его понимания. Почему? Представим ситуацию, где у нас есть помещение с замкнутым квадратным коридором и дверью. Мы можем обойти вокруг и вернуться в то место, где мы были. С точки зрения человека, при встрече с той же самой дверью, мы поймём, что обошли коридор по кругу. С точки зрения робота, если он просто занимается измерением одометрии, он нарисует какую-то траекторию, но она может быть не замкнута, если у него накопилась ошибка. Т.е. та карта, которую мы строим только по одометрии не будет уточнена.

На примере мы движемся по каким-то траекториям и по изображению понимаем, как мы движемся и заодно отмечаем те точки (особенности), которые увидели, строя карту. В результате мы приходим в некую точку на картинке - это разрыв в зеленой линии. Видя глазами человека одно и то же место, понимаем, что стоим в одном и том же месте, а, из-за набежавшей ошибки по одометрии, на карте эти места разные. Данная ошибка в методе визуальной одометрии никак не ликвидируется. Поэтому возникает метод SLAM, позволяющий избежать эту проблему. И в визуальной одометрии, и в SLAM мы можем строить карту, но только в SLAM есть дополнительная сущность, понимающая, что мы пришли в то же самое место, где были и корректирующая карту, чтобы траектория стала замкнутой. «Дрифт», который постепенно добегает в одометрии, будет обнулен. Таким образом у нас появляется долговременная стабильность локализации. Мы перемещаемся достаточно длительное время и как только смотрим на одни и те же объекты из примерно одной и той же точки, учитываем определенный «дрифт» и строим более точную карту без набегания ошибки от времени.

Одним из самых первых методов SLAM, то есть одновременной локализации и построения карты, был PTAM (parallel tracking and mapping). Метод сочетает особенности разделения на локализацию и построения карты, детектирования особенностей и замыкания при прохождении одного и того же участка.

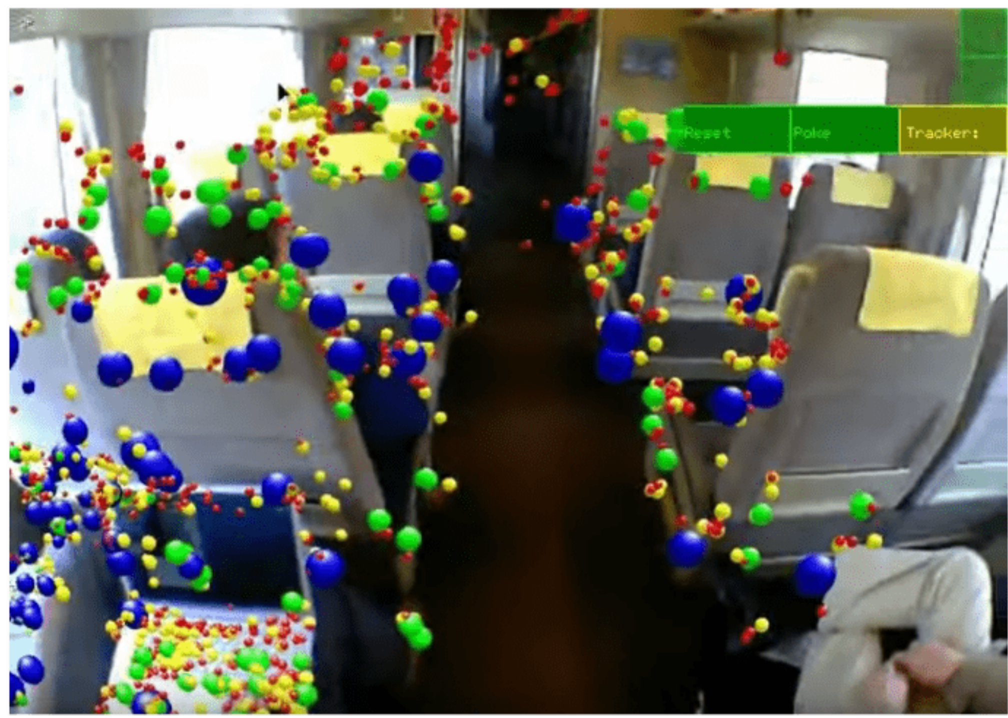


Рис. 6.18: PTAM

PTAM - родоначальник методов SLAM, первый разделивший одометрию, т.е. понимание, как камера перемещается, и построение карты на два отдельных потока. Замыкание траектории работает по следующему принципу - PTAM строит маленькие копии каждого кадра и сохраняет их в память. При поступлении нового кадра с видео сравнивает, не видел ли он то же самое в предыдущий раз и если схожесть в кадре достаточно высокая, понимает, что мы вернулись. Траекторию можно замкнуть и скорректировать набежавшую ошибку. Понятно, что со временем у нас будет достаточно много этих маленьких кадров, сравнивать новый кадр с каждым предыдущим будет все затратнее и система будет работать все медленнее. Ключевым понятием, отличающим SLAM от визуальной одометрии будет понятие ключевого кадра.

В наших роботах мы используем SLAM под названием ORB-SLAM (Oriented FAST and Rotated BRIEF). В рамках игры в футбол есть вражеские и наши роботы, которые являются достаточно большими движущимися объектами, и если SLAM будет за них цепляться, то это будет существенным минусом. Из-за того, что детектор особенностей в PTAM плохо работает с движущимися объектами, будет происходить потеря трекинга. Поэтому мы используем ORB-SLAM с детектором особенностей ORB. Создатель ORB-SLAM перед разработкой сравнил существующие детекторы особенностей и как они работают. Родоначальник - SIFT, который мы уже рассматривали. Он достаточно медленный из-за обилия математики размытий, также он запатентован, т.е. за его использование в коммерческой разработке необходимо заплатить. Все детекторы работают по одним и тем же принципам, но в разной реализации с разной производительностью. Отличием реализации ORB является то, что для особенности строится не 128-размерный

вектор, а двоичное число.

Сама система ORB-SLAM также двухпоточная. Один поток занимается визуальной одометрией, т.е. пониманием как камера двигается, а второй поток занимается построением карты. Она включает в себя подсистему поиска замыкания траектории для минимизации ошибки с одометрией.

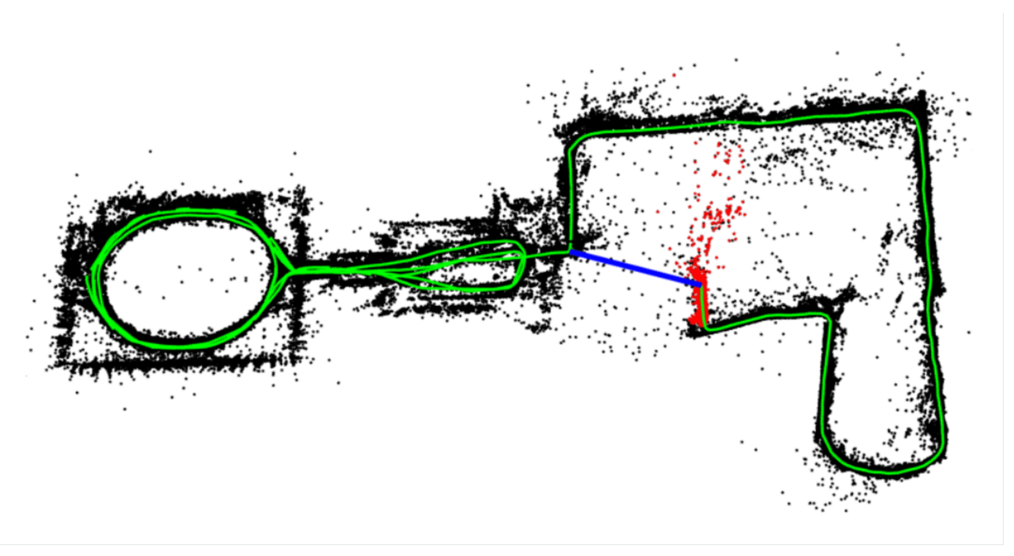


Рис. 6.19: Размытие с целью получения характерного размера

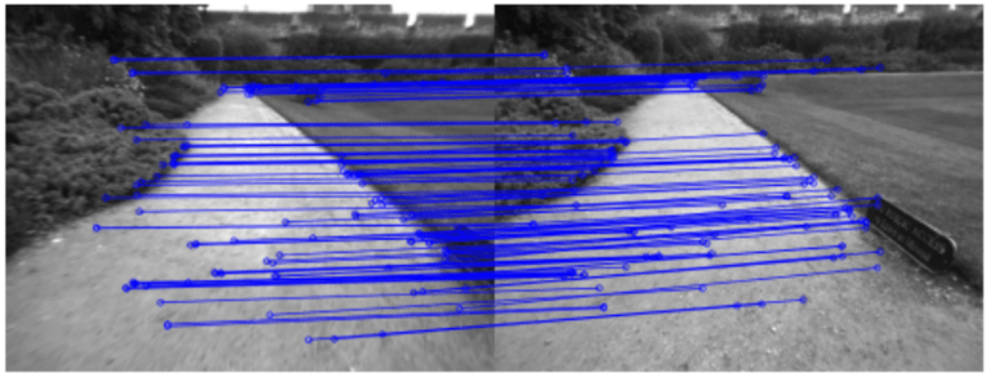


Рис. 6.20: Обнаружение разрыва

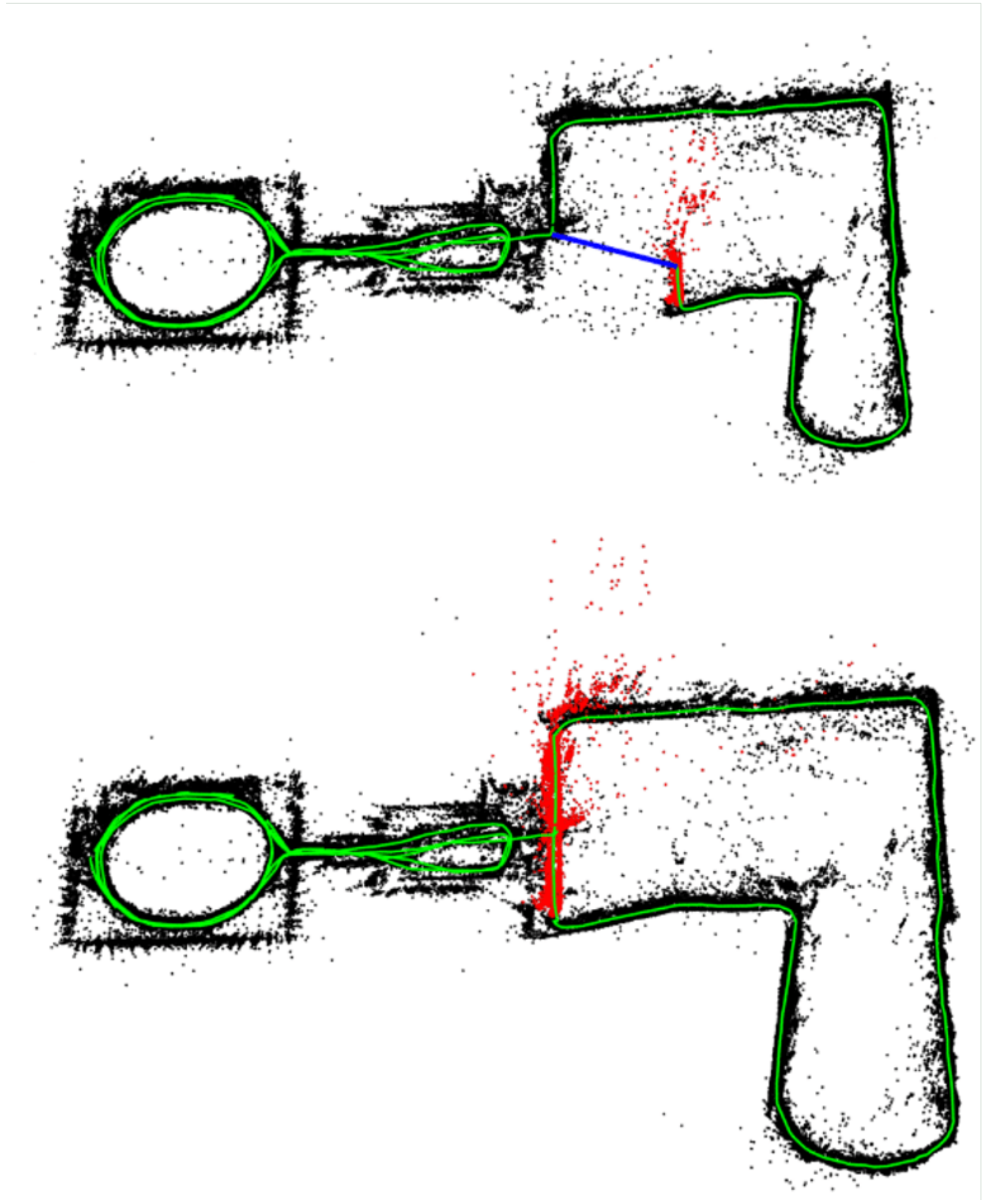


Рис. 6.21: Замыкание траектории

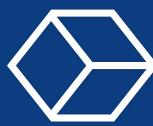
Снова взглянем на изображение, где мы прошли по достаточно длинной траектории и построили однократно локальную карту. По изображениям, снятым в двух точках, понимаем, что одно и то же изображение, потому что особенности расположены примерно в тех же местах. Тем самым алгоритм может скорректировать эту траекторию. Часть траектории, в процессе которой набежала ошибка, скорректирована так, чтобы точка замыкания находилась в одном и том же месте. Это делается с помощью оптимизации графа-позиции камеры. Эволюция камеры в пространстве представляет из себя граф, в котором вершинами являются ключевые кадры, а ребрами являются перемещения. На

этом графе можно обнаружить, что на какой-то части траектории робот систематически не доворачивает на полградуса вправо. И если систематический недоворот скорректировать, то замыкание будет обнаружено и исправлено. В ORB-SLAM системе за это отвечает библиотека g2o graf optimizations. Она позволяет минимизировать некоторую целевую функцию - в данном случае набежавшую ошибку. Само детектирование замыканий выглядит как на примере. Одновременно делается локализацию по изображению, строится карта и делается глобальной. Если робот пришел в то же самое место, где он был, алгоритм скорректирует карту и она станет точной.

Подытожим: мы можем научить робота ориентироваться в незнакомой среде, используя метод SLAM. Он заключается в использовании визуальной инерциальной одометрии и построении карты по пройденной траектории. Визуальная одометрия построена на принципе отслеживания смещения интересных точек на изображении. Построенная карта корректируется, когда мы приходим в то же место, в котором уже были.

Мы рассмотрели многие методы локализации и теперь можем научить робота понимать свое местонахождение.

7. Введение в ROS



7.1 Лекция

ROS (Robotic Operation System) — это гибкая платформа для разработки программного обеспечения роботов, набор разнообразных инструментов, библиотек, определенных правил целью которых является упрощение задач разработки программного обеспечения для роботов. Эту платформу использует множество разработчиков по всему миру. В этой главе будут рассмотрены базовые понятия и возможности ROS.

7.1.1 Зачем нужен ROS

Давайте рассмотрим части система робота: у нас есть отдельные модули отвечающие за такие задачи, как зрение — то есть распознавание объектов на изображении, полученном с камеры. Есть задача локализации — понимание роботом своего ориентирования в пространстве. Задача движения, которая, как нетрудно догадаться из названия, отвечает за перемещение робота и за управление его сервоприводами. Стратегия, которая генерирует список команд и последовательность в который робот должен исполнять. Все эти

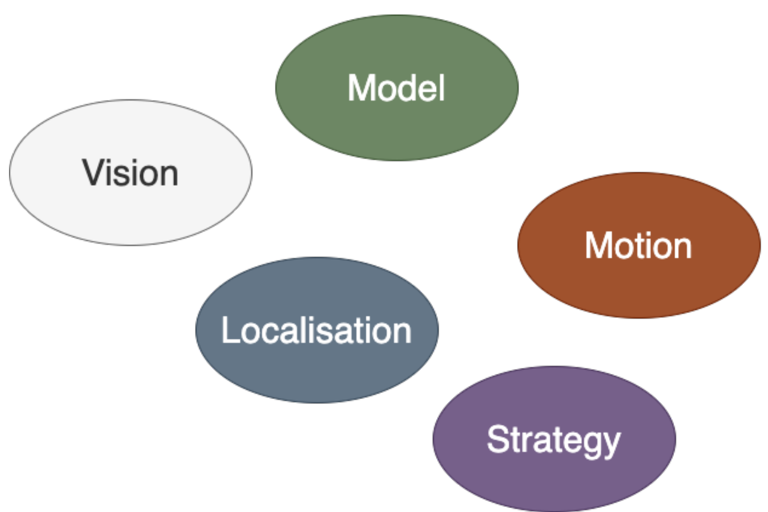


Рис. 7.1: Модули робототехнической системы

модули создаются и действуют независимо. Это обуславливается многими причинами: первая причина — над каждым модулем работает обычно своя команда разработчиков. Это необходимо для ускорения всего процесса разработки целостной системы, чтобы каждый специалист отвечал за свою тему, в которой он разбирается. Следующей причиной является масштабируемость, то есть мы один модуль можем встраивать во всю систему или как-то расширять независимо от других модулей. Еще одна причина модульности системы — это возможность вставлять, убирать модули, заменять их на другие, при этом не изменяя всю систему. При реализации таких целостных систем возникает несколько проблем организации работы этих модули между собой.

Первый вопрос самый важный -взаимосвязь между модулями, то есть создание общение между ними. Что это значит? Это значит, что мы должны разработать какой-то канал связи и какая какие-то какой-то определенный способ передачи данных между модулей.

Рассмотрим такой пример - модуль стратегии предложил в текущий момент времени некоторые указания о том, что роботу нужно пройти в точку с глобальными предметами x и y модуль движения должен принимать эту команду и соответственно реализовывать движение. Появляется вопрос о том, как модулю стратегии информации донести до модуля motion как ему motion ее корректор воспринять. На данной картинке можно видеть, что стратегия сообщает какую-то команду модулю. Там могут быть, например координаты и угол или количество шагов, которые робот должен сделать, но в данном случае наш motion он не знает, как это прочитать. Мы его не научили распознавать сообщение, и он не будет понимать то, что от него хочет стратегия. Другой пример.

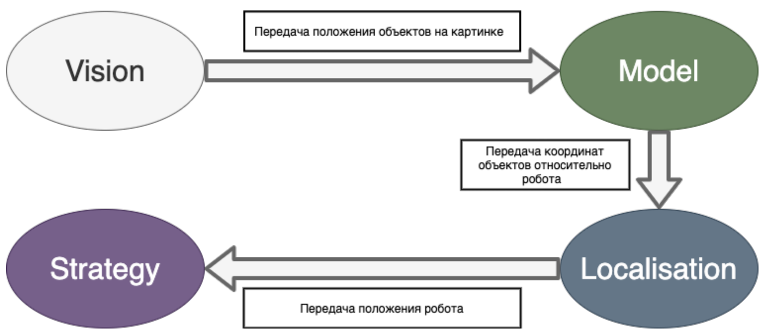


Рис. 7.2: Пример последовательной связи

предположим что модуль зрения с помощью некоторых внутренних алгоритмов нашел на изображение с камеры несколько объектов которые робот будет использовать свой локализации в пространств теперь эти данные необходимо передать в модуль model для того чтобы исходя из геометрии робота и положению робота получить расположение этих объектов в собственной системе координат робота то есть например получить координат ориентиров относительно робота и после этого полученные координаты нужно передать модуль локализации для обработки получение актуального положение робот в пространстве. В решении вот такой у цепочки разработчику можно решить, как их модули будут придавать данный друг другу. Какой будет формат у сообщений? как часто модули будут отправлять сообщения? В какой последовательности это будет происходить?

Таким образом можно сформулировать глобальный вопрос как нам реализовать эту передачу сообщений внутри всей целостной системы, состоящей из множества модулей? Я напомню, что наша команда является командой робофутболу и мы специализируемся на игре роботов гуманоидов в футбол поэтому все примеры что сказанные до этого и произнесенные дальше они так или иначе касаются именно игры футбол роботов

При реализации простой системы модулей для робота футболиста можно прибегнуть к организации бесконечного цикла, в котором по очереди опрашиваются все модули и

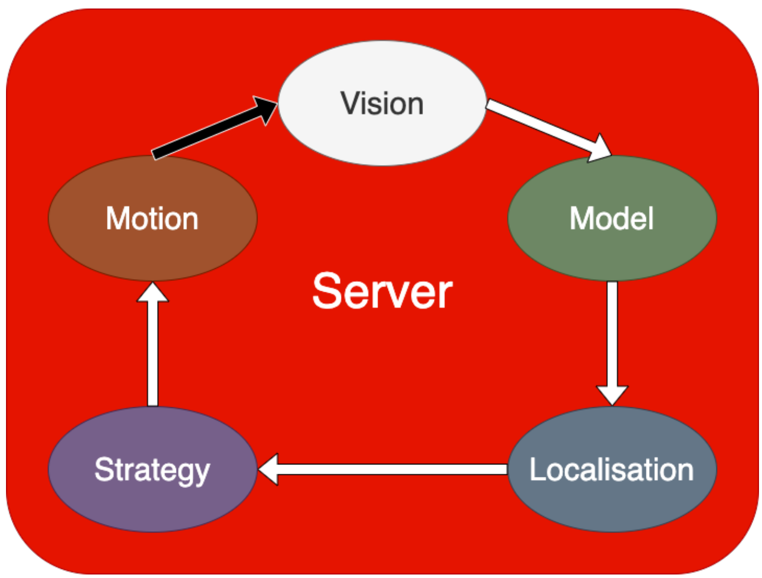


Рис. 7.3: Пример последовательной связи

последовательно передают данные друг другу. все модули находятся в рамках одного процесса, и такая реализация является самый простой и легкой в плане воплощение в реальность.

Эта система выглядит так - у нас есть глобальное пространство, которое выступает в роли сервера, в котором все происходит. Оно является буфером, который хранит и передает данные от модуля к модулю. Цикл начинается с того, что робот смотрит вокруг, замечает ориентиры и мяч потом передает это все через сервер в модуль Model. тот уже считает геометрии робота положение всех этих объектов относительно робота. Потом эти данные передается локализацию. Локализация уже говорит, где робот находится на поле и передает эти данные в стратегию, которая уже выбирает из положения мяча и самого робота дальнейшее поведение. допустим пройти на метр вперед и ударить мяч. дальше робот запускает процесс реализации движения, сказанный с помощью motion робот двигается, а наш цикл тем временем пришел к началу. Цикл крутиться пока мы неостановим игру

Такая реализация очень проста и это не требует каких-то дополнительных программных средств, но у нее есть большой недостаток и это не ориентированность на исполнение в реальном времени. Опрос модули происходит последовательно что в условиях динамической игры может показывать очень низкую эффективность. Например, зрение будет обрабатывать новую порцию кадров только после того, как завершится движение, положение робота будет изменено только после получения данных со зрением и после конца передвижение робота, сам робот будет двигаться только тогда, когда стратегия скажет ему как двигаться. Во время полного цикла может произойти изменения на поле такое как перемещение мяча и действующей в данный момент времени стратегии может оказаться уже сильно не актуальной.

Надо помнить, что под словами стратегии в данном случае подразумевается текущее

задание роботу которые ему необходимо выполнить сейчас. это можно рассматривать как такое примитивное действие, например это может быть удар или перемещение. Это была первая проблема. Вторым недостатком является то, что все запускается в одном процессе у нас есть вот этот глобальный буфер те все нам это нужно упаковать все внутрь одного файла если мы говорим про исполнение на языке Python. Это может быть просто функция main в которой задаются создаются экземпляры классов всех модулей и в какие-то в переменные внутри меня будут записываться значения данных модулей, которые они выдают и передают другим. Из-за все эти процессы отвечает main это опять же нужно все прописывать ручками – все взаимосвязи, согласования, формат сообщений и т.д. Что нам со всем этим делать?

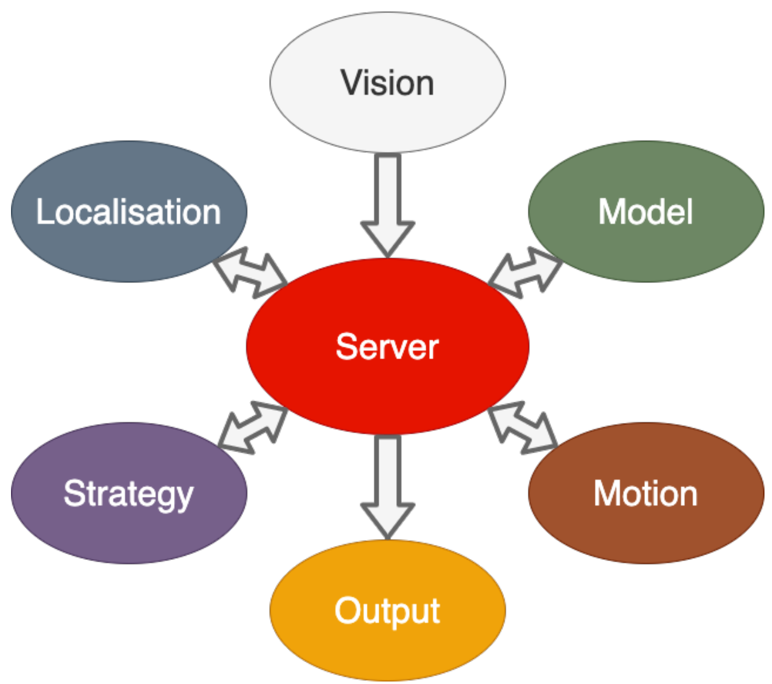


Рис. 7.4: Пример параллельной связи

Логично предположить, что для того, чтобы быть ориентированным на реальном время у нас все модули должны тоже работать в реальном времени в разных процессах. Это означает кучу запущенных параллельных процессов одновременно то есть независимо от того что делает робот допустим он двигается - зрение все равно получает картинку с камеры, локализации постоянно добавляет положение то есть робот делает шажок вперед это пришло в локализацию робот знает что он двинулся вперед его абсолютное положение на поле изменилось все считается постоянно данные приходят в нужные места тогда в момент когда они нужны и появились. И все, кажется, у нас хорошо, но опять же возникает проблема чтобы написать такую огромную целостную систему, которая будет работать параллельно чтобы эти модули хорошо взаимодействовали между собой чтобы все было стабильно — это очень большая и сложная задача. Она может занимать даже годы времени.

А еще есть проблема, которая ставится перед нами разработчиками это написание самих модулей. Полностью создавать все модули с нуля — это не эффективное решение, мы должны использовать готовые наработки инструменты и существующие пакеты чтобы сократить время работы своего ПО И еще важный пункт — это использование симуляции, когда мы говоримо разработке каких-то работ технических систем нас встаёт вопрос о том, что мы хотим это сначала запустить симуляцию не на реальном роботе. зачастую невозможно сразу запускать на написанный код сразу роботе в, потому что оборудование очень дорогое и в случае каких-то серьезных ошибок в коде мы можем понести серьезные финансовые потери. Так для того чтобы использовать тестовую симуляцию мы не должны эту симуляцию сами писать потому что это уж очень сложный процесс и нам бы было бы удобно использовать то что уже написано и то же самое касается в принципе всех модулей, то есть если какая-то задача была бы уже решена до нас то нам нужно это использовать

Давайте подведем небольшой итог по проблемам который встает перед нами и зачем нам нужно какое-то стороннее программное обеспечение первая проблема — это организация связь между модулями и проблема оформление типа передаваемых данных – как мы запаковываем как в каком виде это все передается вторая это создание и добавление новых модулей в уже существующую систему, то есть если мы написали какую-то готовую систему, и мы хотим добавить то модуль мы должны для этого не переписывать всю систему, а именно встроить модуль уже просто его как бы вставить деталь в конструкторе. Третья проблема - система робота должна быть ориентирована на исполнение в реальном времени это тоже усложняет процесс разработки мы не должны все описать с нуля мы должны использовать что-то готово переиспользовать написанные пакеты. Приведу аналогию с языком python если вам нужно работать с матрицами, то вам не нужно писать для этого свой код, вы скорее всего просто возьмите готовую библиотеку numpy или ей подобную. и последний пункт, который я еще не упомянул это отладка всего проекта. В проекте, состоящем из множества модулей и работающем в реальном времени нужно так же обрабатывать ошибки искать баги, отлаживать смотреть что где почему как что не работает и для решения всех этих проблем всех этих задач на помощь разработчикам приходит Robotic Operation System.

OS аббревиатура с обычно означает операционной системы общего назначения но при этомROSявляется не операционной системы в широком смысле этого слова а скорее описывается как метаоперационной системы . мета операционная система совсем не похоже на обычную операционную систему она работает поверх существующих систем .для работыROSнеобходимо базовая операционной система например linux. вообще говоряROSработает не только на линуксе но и на других системах, но это зависит от его версии . после завершения установкиROSна любую операционную систему можно использовать функции все те же функции операционной системы которые были плюс еще дополнительная оболочка сверху.ROSобеспечивают дополнительный функционал необходимый для роботов например работа с библиотеками передачи и приема данных для разных устройств, планирование, обработка ошибок. Этот тип программного обеспечение также называется промежуточным программным обеспечением или программным фреймворк. К его основным целям относятся повторное использование

программных модулей. разработанные программные модули запускаются в любом другом приложении. Вопрос установки зависимостей от других библиотек хорошо проработан и автоматизирован. Следующий пункт — это готовый протокол коммуникация. Одна из основных проблем о которой мы уже говорили это решение задачи коммуникации в рамках одного приложения. Для решения всех этих задачROSсодержит все необходимые утилиты. Любой программный модуль может быть представлен как отдельный процесс, взаимодействующий с другими процессами по протоколу. Такой подход позволяет создавать независимые простые повторения использование программных модулей, которые можно запустить остановить модифицировать на любом устройстве еще пункт — это развитые средства разработки и отладки. В ROS присутствует готовый инструментарий для отладки, инструменты 2d визуализации, инструмент 3d визуализации и даже симуляция. Еще к особенностям у ROS следует отнести активно открытое сообщество. Оно включает разработчиков робототехники из академического мира, из индустрии, из промышленности. Со всего света множество разработчиков принимают участие в разработке поддержки роса. Это платформа является открытой является open source. Существует более 5 тысяч пакетов, которые описаны более чем на 18 тысяч страницах. Другими словами, существует очень много программного обеспечения готового с написанной к нему документации. Можно сказать, что существует сформировавшиеся экосистема с разработчиками всех уровней, индустриальными компаниями. ROS является таким международным промышленным стандартом. все это находится в рамках одной платформы.

Какие цели преследовались? Главной целью было создание среды разработки позволяющий множеству разработчиков контактировать вместе и совместно разрабатывать свои сложные системы, повторно используя уже реализованный код . удобства в плане организации работы процессов в нашей робототехнической системе .структура роса создана виде минимальных единиц выполнимых процессов – нод, которые будут описаны далее, каждый процесс выполняется изолировано, взаимодействие происходит между ними только на уровне обмена сообщениями то есть то что мы говорили о том что модули должны быть желательно как можно более независимыми. ROS предлагает универсальную работу с пакетами. Если вы знакомы с установкой и работой с пакетами в unix-подобных системах, то для вас здесь не будет новой информации. В ROS таким же способом ставиться пакеты, библиотеки и программы. Из коробки доступно очень всего полезного, большое количество документации, обучающих материалов, которые помогают разработчикам работать с этим фреймворками. Также существует удобство при разработке на python и c++, потому как для них реализовано единое api. Внешне для нас не важно на каком из этих языков написан пакет, взаимодействие с ним будет одинаковое.

7.1.2 Термины ROS

Основные термины и понятия в ROS

Теперь, когда мы составили общее представление о том, что такое ROS - мы приступим к знакомству с основными терминами ROS и с тем, как это можно реализовывать в наших задачах. Основные термины ROS. Сначала мы разберем такие вещи: ноде, мастер-ноде,

о том, что такое сообщение в ROS. Далее мы узнаем о моделях обмена сообщениями подписчик-издатель, сервис, действия.

Начнем мы с пакета. Пакет является основной единицей в системе ROS, любое приложение ROS оформляется в пакет, в котором определяется его конфигурация параметры, ноды то есть процессы которые внутри этого пакета будут исполняться и зависимости от других пакетов. Это изолированный отдельный субъект. Работа с пакетами похоже на работу с пакетами в unix системах, то есть можно просто поставить готовый пакет одной строчкой в терминале, можно скачать исходники и самим его собрать, можно написать свой. В данном в контексте модули для робота мы будем говорить что каждый модуль должен являться пакетом. На сайте ROS есть список существующих готовых пакетов отлаженных которые можно смотреть изучать скачивать.

Нода

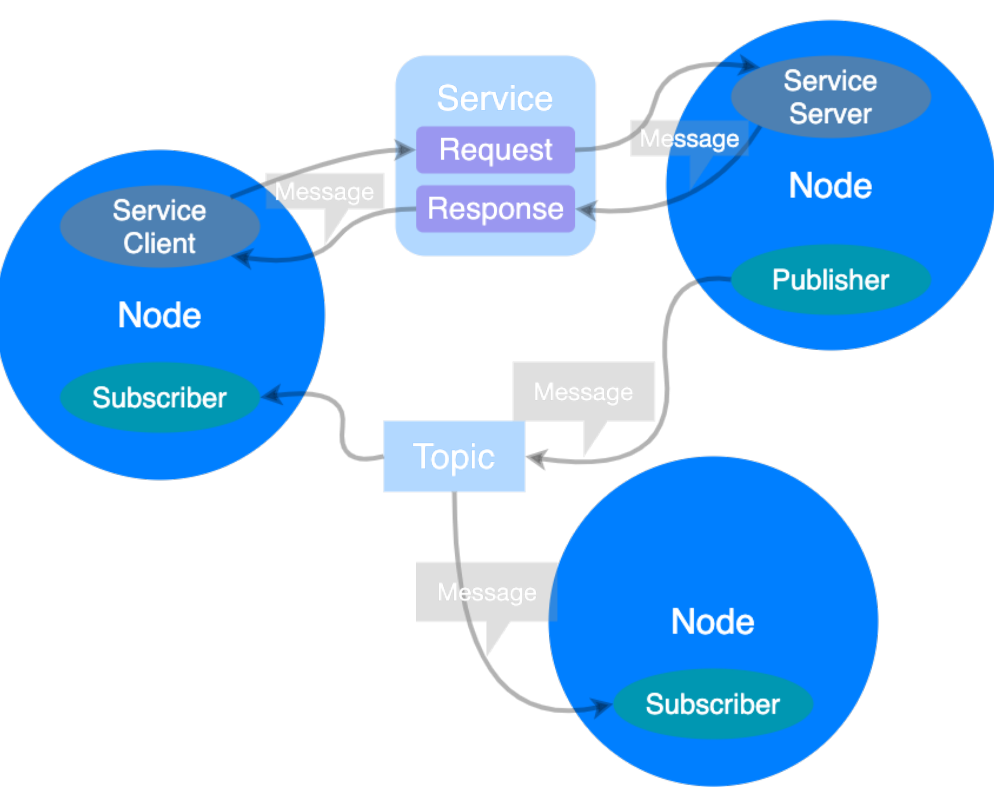


Рис. 7.5: Ноды подписаны и выделены синним

А теперь мы поговорим о не менее важном понятием в системе ROS таким же важным как пакет — это понятие ноды. Нода является наименьшей рабочий единицы используемый в рос. можно провести аналогию с одной исполняемой программы. В документации ROS рекомендуется создавать одну ноду для каждой задачи что позволит легче использовать ее в других проектах. При запуске нода регистрирует информацию себе на сервере, то есть еще 1 главный ноде и дальше зарегистрированная она может взаимодействует с другими нодами. При регистрации она дает свое имя и тип сообщений

через которые она будет общаться то есть отправлять сообщения то в каком виде она будет передавать данные с другими ноды. важно отметить что существующие мастер-нода который является главной она запускает все остальные ноды и создает единое пространство имен и типов сообщения которые уже используют все прочие ноды и больше она никак не участвует. ноды сами общаются между собой напрямую просто подтягивая с мастер ноды информацию других таких же процессах. На рисунке ноды окрашены в синий цвет. Повторим есть главная нода, которая хранит информацию об именах и сообщениях и запускает другие ноды, ноды действуют независимо и взаимодействуют между собой узнавая у мастер ноды что куда отправлять и откуда принимать и. в каком виде

Сообщение

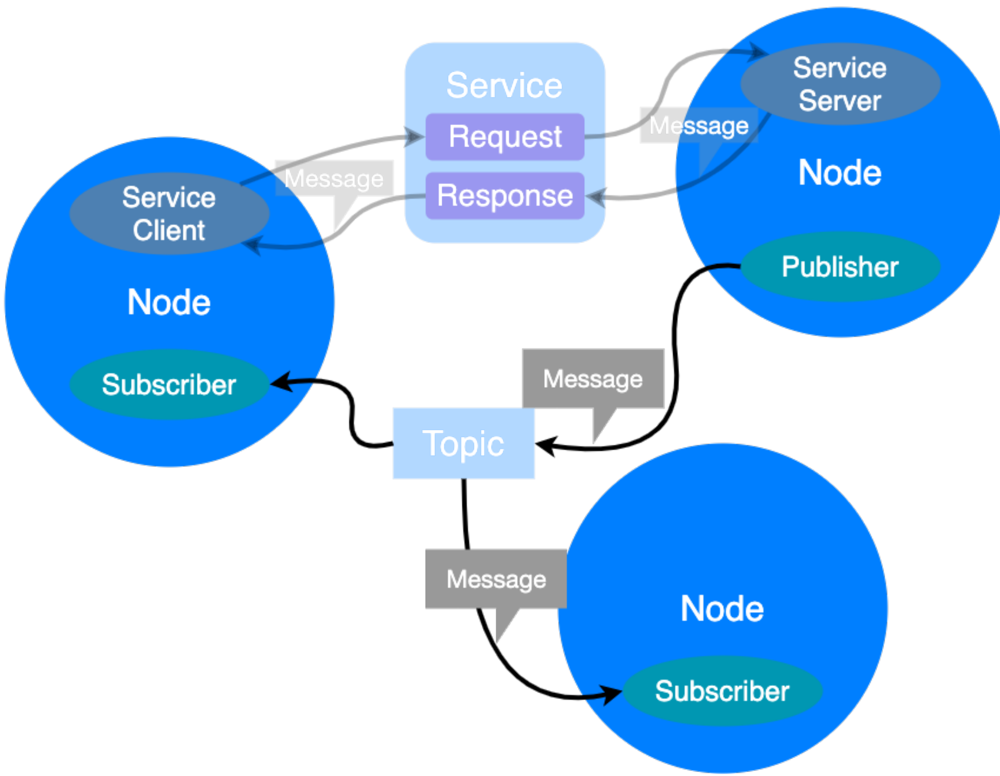


Рис. 7.6: Ноды подписаны и выделены синним

Следующее понятие, которое мы рассмотрим - это понятие сообщения. Между собой ноды общаются с помощью сообщений. Сообщения описываются в процессе разработки пакета. Они могут быть как простыми и представлять из себя знакомые по языкам программирования типы integer float string, та и сложными и состоять из комбинаций простых. Тип используемых нодой сообщений сообщается мастер ноде. Каждое сообщение может быть представлено как файл, отправляемый по внутреннему протоколу, роса. Как мы уже сказали сообщения являются средством передачи информации между содами

Сообщения чем-то похожи на структуры данных в языках семейства си. Они могут содержать в себе как базовые типы данных таких как integer float string, так и более сложные такие как сообщения ROS или даже целые массивы сообщений. Сообщения описываются в файлах с расширением msg. И описываются в виде двух столбцов, состоящих из пар тип и имя. Пример на кринке — это сообщение, содержащее 2 поля float одно поле со строкой и одно поле с типом данных Point. Point это одно из базовых сообщений ROS включающее в себя 3 поля float для описания положения точки в трехмерном пространстве. Таким образом в сообщения можно вкладывать другие сообщения. В росе доступно большое количество уже описанных сообщений, но мы можем создавать сами любые сообщения, которые нам нужны под наши конкретные задачи.

Подписчик-Издатель

Давайте теперь рассмторим одну из возможных моделей сообщения между модулями. В ROS предустренно 3 базовых моделей передачи сообщений и самая простая и спользуемая это топик (еще она называется Подписчик-Издатель или Publisher-Subscriber).

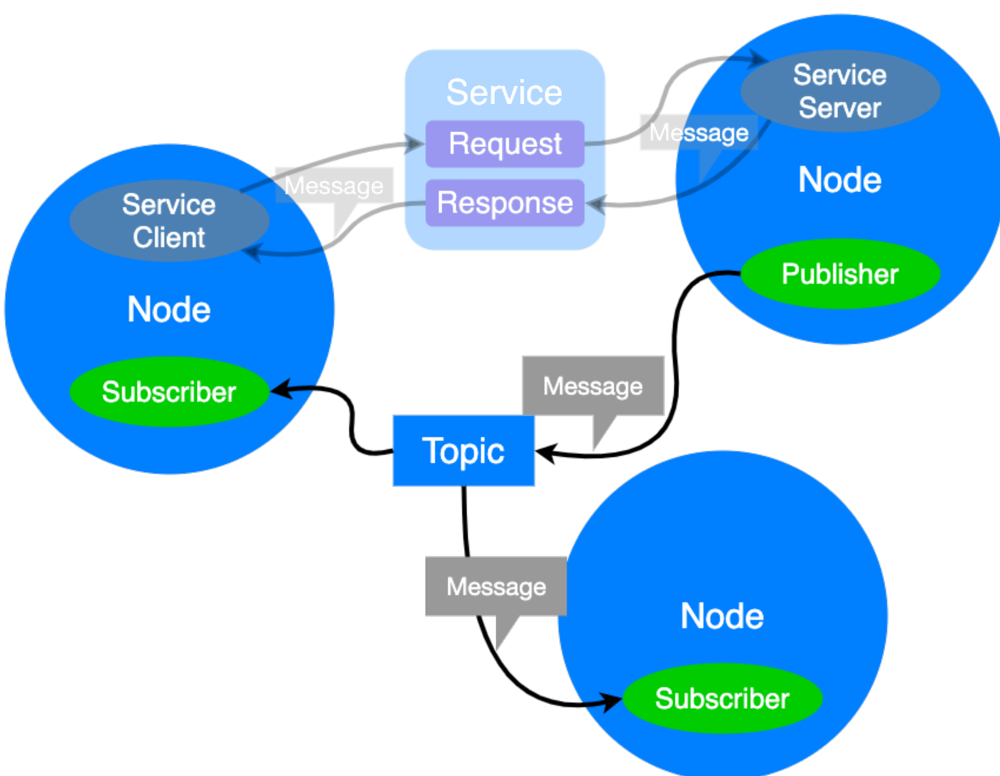


Рис. 7.7: Подписчик-Издатель

Выглядит это следующем образом. Издатель при своей инициализации сообщает мастер ноде что он является издателем с определенным топиком (Topic - Тема). Ее мастер нода регистрирует в глобальном пространстве имен, и она становиться доступной для всех. После этого издатель может начинать передавать в этот топик сообщения, где

они и будут храниться. Далее инициируется нода с подписчиком и запрашивает доступ к теме топику с требуемым именем. В результате она получает доступ к топике и может забирать оттуда сообщения. Каждый издатель и подписчик находятся в своих нодах. При этом стоит отметить, что подписчиков может быть много и они все могут забирать эти сообщения. В данном случае связь будет асинхронной - если есть издатель - не факт, что будет читатель. Можно привести аналогию с новостным изданием - редактор приносит сообщения, например на сайт новостного издания, а читатели этого издания могут эти сообщения читать. Если сравнивать с работой наших модулей в роботе, то с помощью такой связи можно организовать сообщение между модулями зрение-локализация. сервис общается с помощью таких же сообщений как топик, но для сервиса их должно быть 2 - для запроса и ответа.

Сервис

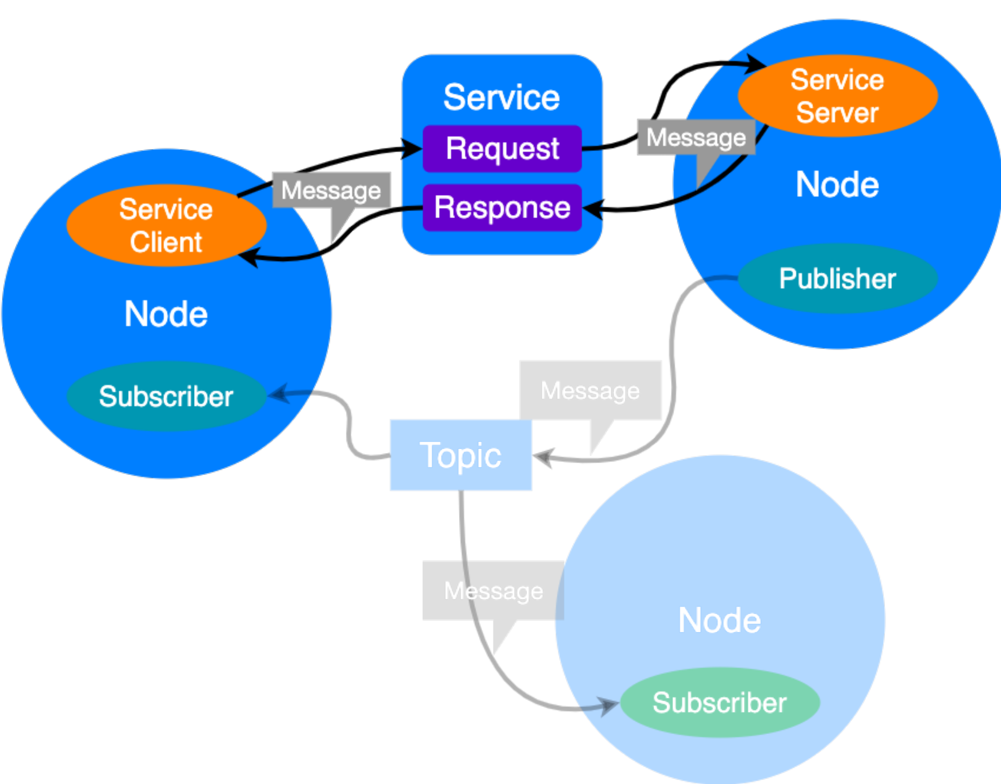


Рис. 7.8: Ноды подписаны и выделены синним

Следующая модель передачи сообщений — это сервис. В ней используются такие понятия как сервис-клиент, сервис-сервер. В отличии от предыдущего способа этот метод является синхронным. Это больше похоже на запрос-ответную систему такую как поисковик в браузере. Клиент-сервис делает запрос сервис-серверу и через глобальное пространство имен этот запрос к серверу попадает. Север выполняет запрос и направляет ответ обратно клиенту. Например, программе в клиенте требуется сложить два числа: икс и игрек. Она формирует запрос и отправьте его через сервис серверу. Тот стоит в

ожидании запросов и при его получении начинает работу, в данном случае складывает два числа, и отправляет сумму как ответ на запрос через тот же сервис. Сервис регистрируется в глобальном пространстве имен по аналогии с топиком при создании ноны. Важным отличием от модели Подписчик-Издатель является то, что сервис-сервер не работает постоянно, а активируется только при наличии запроса, а сервис клиент будет ждать пока его запрос будет выполнен.

Действие

Кратко рассмотрим еще одну модель сообщения называемой Действие (Action), которая применяется реже и для более специфических задач. Больше всего эта модель похожа на предыдущую с тем отличием, что в ней происходит поэтапное информирование клиента о том, как идет выполнение задачи. То есть сервер после начала обработки запроса отправляет сообщения, в которых содержится информация о ходе выполнения задачи. Если нам требуется следить за процессом исполнения запроса. Пример нам требуется, чтобы робот прошел 10 метров и отчитывался о каждом пройдено метре. Вот так «пройди 10 метров» - «пройден 1 метр» пройдено 2 метра и т.д.

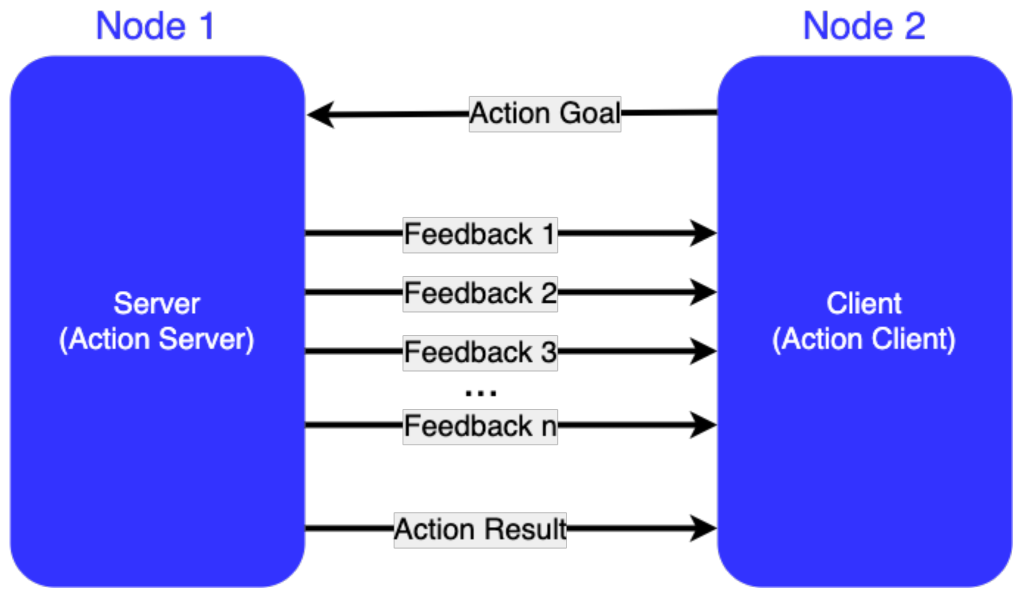


Рис. 7.9: Пример модели Действие

7.1.3 Использование ROS

Обратился теперь к моделям сообщения между нодами. На картинке представлена возможная реализация связи между модулями локализации зрения и движения. Так как система у нас работает в реальном времени зрение и движение в своих ритмах передают информацию об измерениях в свои соответствующие топики. Зрения - когда были найдены ориентиры в кадре, модуль движения - когда было совершено какое-то перемещение. Для каждого из модулей подразумевается свой тип специфический тип сообщений. Локализация будет постоянно смотреть не положили ли остальные модули

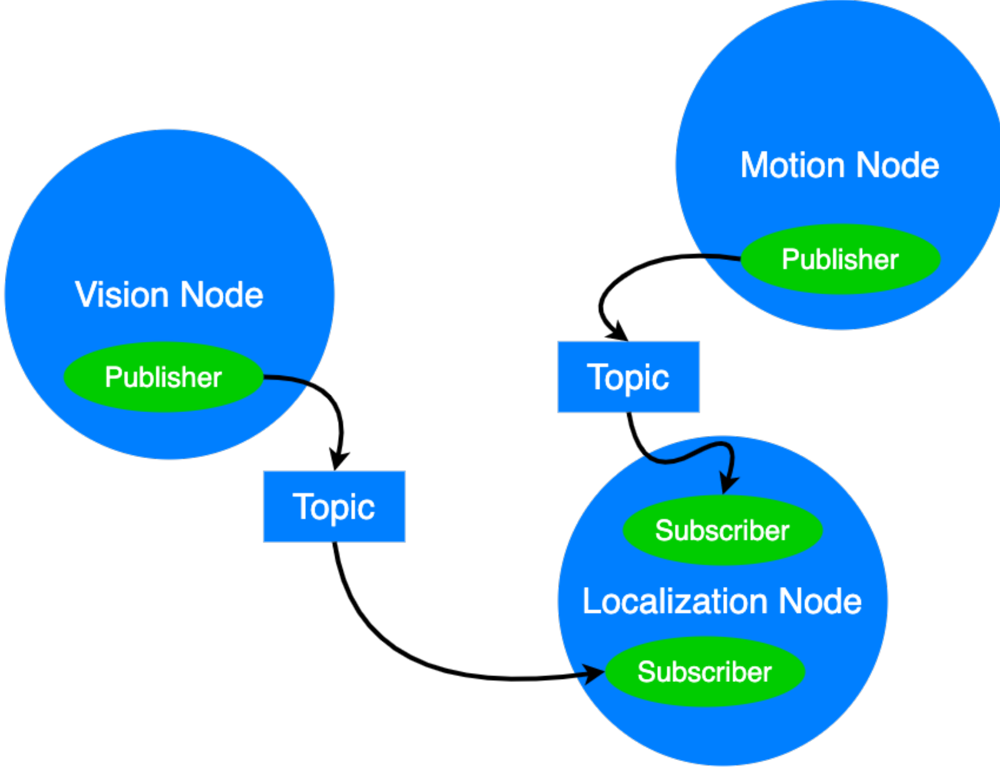


Рис. 7.10: Пример паралельной связи

новых сообщений в топики и если новое сообщение появилось, то локализация его прочитает и обработает. Каждый модуль будет работать самостоятельно и независимо. Локализация будет обновляться даже если данные со зрения не будут приходить, если при этом будут поступать данные движения. Таким образом для системы нам потребуется 3 ноды и два топика с двумя подписчиками и издателями. Внимательные зрители могут заметить, что мы почему-то передаём данные со зрения напрямую в локализацию минуя модуль model. Почему это?

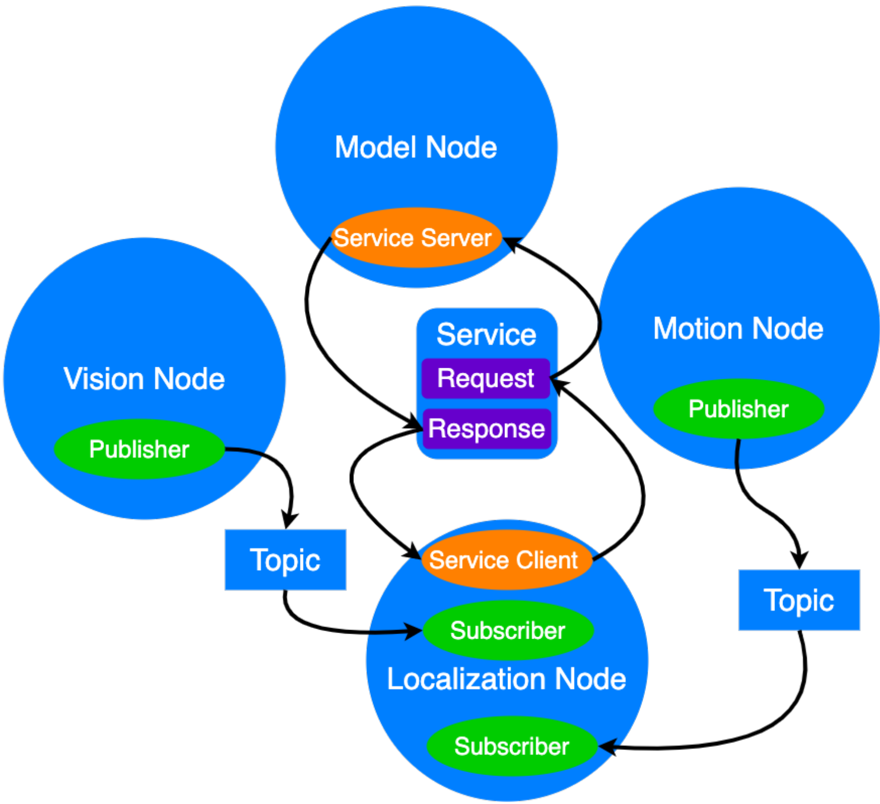


Рис. 7.11: Использование разных моделей связи в одной системе

Ответ заключается в том, что для модуля model лучше использовать сервис. Это происходит из-за того, что model нужен нам не всегда, а только когда приходят данные со зрения. В данном случае модуль model выступает как функция, которая переводит данные из зрения в удобный формат для локализации. В отличии от модуля motion который публикует данные непрерывно, модуль model не публикует ничего пока ему не будут переданы данные. Отсюда возникает идея сделать его сервисом, а локализацию неким агрегатором данных со зрения и движения. Данные со зрения приходят неоднородно по времени. И когда они приходят только тогда мы вызываем model через сервис. Еще одним аргументом в пользу такого решения является то, что неактивный model не будет отбирать ресурсы компьютера, которые у нас ограничены.

Однако стоит заметить, что данная реализация не является единственно верной и способ взаимодействия между модулями разработчик должен выбирать сам, отталкиваясь от конкретной задачи. Даже эту задачу можно решить иначе, например сделав из модуля model подписчика на зрение и издателя для локализации. ROS дает нам творческую свободу реализации.

7.2 Семинар

7.2.1 Подписчик-издатель

В этом семинаре будет разобраны практический пример работы с ROS. В качестве задачи предлагается сделать программу, состоящую из двух процессов. В одном пользователь вводит числа в одну командную строку, а в другой командной строке выводится сумма всех введённых чисел из первой. В качестве примера здесь используется синтаксис ros1 на unix подобной системе, но общие идеи будут справедливы и для ros2. Для начала работы требуется установленный ros на компьютере. В данном семинаре не будет представлена подробной установки самого роса на пк, она доступна на сайте ros.org. Если ROS установлен и рабочее окружение создано, то можно приступать к выполнению задания Для выполнения задачи нам потребуется создать новый пакет. Для этого мы в консоли переходим в папку рабочего пространства командой:

```
cd ~/catkin_ws/src
```

Далее мы создаем новый пакет командой

```
catkin_create_pkg summator std_msgs rospy
```

Эта команда создаст папку с нашим пакетом в папке src. Теперь нам нужно создать два питоновских файла в папке summator/scripts с названиями sender.py и summator.py.

Теперь открываем файл sender.py. Первое что нам нужно сделать это сделать импорт библиотеки rospy для работы с ROS из питона, а также импортировать объект сообщения с типом инт64 для наших сообщений.

```
import rospy
from std_msgs.msg import Int64
```

После этого мы создаем функцию для программы отправителя, внутри нее мы инициируем ноду отправителя с помощью функции

```
rospy.init_node('user1', anonymous=True)
```

И создаем объект Издателя. У Издателя мы задаем имя топика, куда он будет отправлять сообщения, тип сообщений – в нашем случае int64 и размер очереди сообщений.

```
pub = rospy.Publisher('user1', Int64, queue_size=10)
```

Далее мы создаем вечный цикл, внутри которого у нас будут выполняться две вещи, первая это считывание строки из консоли и преобразование ее к числу и второе pub.publish этого числа в топик

```
while(True):
        massage_str = int(input())
        pub.publish(massage_str)
```

Теперь сделаем функцию main с добавлением конструкции try/except чтобы если что отловить возможную ошибку.

```
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Все наша программа отправитель готова. Теперь надо сделать программу получатель. Теперь открываем файл summator.py Делаем аналогичные импорты

```
import rospy
from std_msgs.msg import Int64
```

Создаем класс summator Объявляем ноду и переменную под обновляемую сумму. Пишем объявление Подписчика, Внутри функции rospy.Subscriber вносится имя топика на который мы будем подписаны тип сообщения и передается функция callbacks.

```
def __init__(self):
        rospy.init_node('user2', anonymous=True)
        self.sum = 0
        rospy.Subscriber('user1', Int64, self.callback)
```

Функция callback будет вызываться во внутреннем цикле Подписчика в тот момент, когда в топике будет появляться новое сообщение и обрабатывать его

```
def callback(self, data):
        self.sum+=data.data
        print(self.sum)
```

вот так может выглядеть функция callback - у него есть аргумент дата в котором будет находиться сообщение из топика. Так как в топике будет находиться число мы его оттуда забираем и прибавляем к сумме, а также выводим сумму на экран Еще нам понадобиться функция, которая будет отвечать за внутренний цикл Подписчика мы назовем ее ран. Внутри будет только одна строчка к rospy.spin().

```
def run(self):
        rospy.spin()
```

Теперь создаем функцию мейн аналогичным способом, в ней создаем экземпляр класса и пешем функцию ран.

```
if __name__ == '__main__':
    try:
        tal = talker()
        tal.run()
    except rospy.ROSInterruptException:
        pass
```

Оба файла целиком: sender.py

```
import rospy
from std_msgs.msg import Int64

def talker():
    rospy.init_node('user1', anonymous=True)
    pub = rospy.Publisher('user1', Int64, queue_size=10)

    while(True):
        massage_str = int(input())
        pub.publish(massage_str)


if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass


```

summator.py

```
import rospy
from std_msgs.msg import Int64

class talker():
    def __init__(self):
        rospy.init_node('user2', anonymous=True)
```

```
        self.sum = 0
        rospy.Subscriber('user1', Int64, self.callback)

    def callback(self, data):
        self.sum+=data.data
        print(self.sum)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    try:
        tal = talker()
        tal.run()
    except rospy.ROSInterruptException:
        pass
```

Для того чтобы собрать пакет нужно напечатать служебную команду:

```
catkin_make
```

После сборки нам нужно сделать файлы исполняемыми:

```
. ~/catkin_ws/devel/setup.bash
```

Теперь приступаем к запуску. Для начала в отдельном окне терминала запустим roscore. Это служебный процесс котоырй отвечает за запуск всего роса.

```
roscore
```

Запускам наш отправитель командой rosrun

```
rosrun summator sender.py
```

Открываем еще одно окно и запускаем так же сумматор

```
rosrun summator summator.py
```

теперь можно проверить работу нашей программы. В окне с запущенным sender.py нужно вводить число и нажимать Enter. В окне summator.py надо проверить правильно ли выводится сумма. Если требуется прервать выполнение программы то это можно сделать комбинацией клавиш CTRL+C.

Как понимать суть этого задания? Мы создали две программы с двумя нодами внутри одного ROS пакета. Одна нода соответствует типу publisher другая типу subscriber. Они

общаются через общий топик. Причем эти программы работают из разных процессов и единственное что их объединяет это название топика и тип сообщений, который мы задали.

7.2.2 Клиент-сервис

В данном упражнение предлагается попробовать решить похожую на предыдущею задачу, только с использованием другого типа сообщения, а именно клиент-сервис. Предлагается сделать клиент, в котором пользователь будет вводить 2 числа и сервис, и сервис, который будет считать сумму. Важным отличием будет то, что нам потребуется написать свой собственный тип сообщений (в предыдущей задачей мы использовали базовые типы сообщения). В ROS существует два типа сообщений, с расширением .msg и с расширением .srv. Первое используется для типа сообщения Подписчик-Издатель, второе для Клиент-Сервис.

Формат запроса и ответа, задается специальным парным Сообщением, в котором есть два сообщения: первое для запроса (Service Request), второе для ответа (Service Response). Файлы с описанием сервисов хранятся в директории srv и имеют расширение .srv.

Рекомендуется сделать отдельный новый пакет, аналогично предыдущей задаче, но допускается и работа в существующем.

Для того, чтобы использовать .srv сообщения необходимо сделать несколько шагов. Первый шаг, нужно создать папку srv в папке пакета. В это папке нужно создать текстовый файл SumTwoInt.srv. Этот файл должен содержать следующее перечисление типов:

```
int64 A
int64 B
---
int64 Sum
```

Из этого файла после сборки пакета будут созданы файлы сообщений. Следующий шаг, нужно изменить содержание служебных файлов. Для использования Service в новых пакетах необходимо удостовериться, что установлены все зависимости и внесены изменения в конфигурацию make.

В файле CMakeLists.txt необходимо добавить следующую информацию:

```
# Проверяем, что пакет message_generation подключен
find_package(catkin REQUIRED
    COMPONENTS message_generation)

# Declare the service files to be built
add_service_files(FILES
    SumTwoInt.srv
)
```

```
9
10 # Actually generate the language-specific message and service files
11 generate_messages(DEPENDENCIES std_msgs sensor_msgs)
12
13 # Declare that this catkin package's runtime dependencies
14 catkin_package(
15     CATKIN_DEPENDS message_runtime std_msgs
16 )
```

В файл package.xml необходимо добавить или раскомментировать следующие строчки:

```
1 <build_depend>message_generation</build_depend>
2 <build_depend>message_runtime</build_depend>
3 <exec_depend>message_generation</exec_depend>
4 <exec_depend>message_runtime</exec_depend>
```

Теперь при использовании команды:

```
1 catkin_make
```

ROS подготовит сообщения для использования. Теперь нам надо написать две программы, аналогично предыдущему заданию. Программы будут называться client.py и server.py. начнем с сервера. Первое что нам нужно сделать это также сделать импорт библиотеки rospy для работы с ROS из питона, а также импортировать объекты сообщений, которые мы раньше создали.

```
1 import rospy
2 from sc_sumator.srv import SumTwoInt, SumTwoIntResponse
```

В этом импорте соответственно два типа сообщений - для запроса и ответа. Название $sc_sumator$ является названием нашего пакета.

Далее нам нужно объявить функцию, которая будет вызываться в сервере в ответ на запрос.

```
1
2 def handle_sum_two_ints(req):
3     sum = req.x + req.y
4     return SumTwoIntResponse(sum)
5
```

Переменная req в данном случае будет содержать сообщение с типом SumTwoInt. Возвращать функция должна сообщение с типом ответа. Далее мы должны описать функцию, которая будет является сервером. В ней должна находиться инициализация ноды:

```
rospy.init_node('sum_two_ints_server')
rospy.spin()
```

Создание обьекта сервиса. В скобочках перечисляются название нашего сервиса, тип сообщений и функция, которая будет вызываться для обработки:

```
s = rospy.Service('sum_two_ints', SumTwoInts, handle_sum_two_ints)

```

И функция, которая отвечает за работу сервера в цикле:

```
rospy.spin()
```

Теперь перейдем к коду программы клиента. Сначала необходимые импорты:

```
import rospy
from sc_sumator.srv import SumTwoInt
```

Далее нам нужно написать функцию клиента. Она начинается со служебной команды, которая блокирует дальнейшее выполнение кода пока сервис не станет доступным:

```
rospy.wait_for_service('sum_two_ints')
```

Далее в бесконечном цикле мы считываем два числа из строки:

```
a,b = input().split()
```

И используем конструкцию try/exept для обработки возможных ошибок:

```
try:
    ...
except rospy.ServiceException, e:
    print("Service call failed: %s"%e)

```

Далее мы создаем дескриптор для вызова сервиса:

```
sum_two_ints = rospy.ServiceProxy('sum_two_ints', SumTwoInts)

```

Теперь мы можем использовать наш сервис как обычную функцию python:

```
res = sum_two_ints(int(a), int(b))

```

И выводим результат:

```
print(res.sum)

```

Программа клиента готова. Обратите внимание, что мы строго задаем тип полей в сообщениях и в данном примере мы можем использовать только целые числа, так как задали тип int.

Оба файла целиком: server.py

```
import rospy

from sc_sumator.srv import SumTwoInt, SumTwoIntResponse

def handle_sum_two_ints(req):
    sum = req.x + req.y
    return SumTwoIntResponse(sum)

def sum_two_ints_server():
    rospy.init_node('sum_two_ints_server')
    s = rospy.Service('sum_two_ints', SumTwoInts, handle_sum_two_ints)
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

client.py

```
import rospy

from sc_sumator.srv import SumTwoInt

def sum_two_ints_client():

    rospy.wait_for_service('sum_two_ints')
    while(True):
        a,b = input().split()
        try:
            sum_two_ints = rospy.ServiceProxy('sum_two_ints', SumTwoInts)
            res = sum_two_ints(int(a), int(b))
```

```
            print(res.sum)

        except rospy.ServiceException, e:
            print("Service call failed: %s"%e)

if __name__ == "__main__":
    add_two_ints_client()
```

Так же, как и в предыдущем примере необходимо собрать пакет нужно командой:

```
catkin_make
```

Сделать файлы исполняемыми:

```
. ~/catkin_ws/devel/setup.bash
```

Запустим roscore.

```
roscore
```

И тперь можно запускать нашу программу. Запускам наш сервер:

```
rosrun sc_summator server.py
```

И клиент:

```
rosrun sc_summator client.py
```

теперь можно проверить работу нашей программы. В окне с запущенным client.py нужно ввести два числа и нажимать Enter. В этом же окне должен напечататься ответ от сервера.

7.3 Практическое занятие

На семинаре были рассмотрены два примера программ с использованием ROS с различными типами сообщения. Для практической работы обучающимся предлагается сами реализовать программы для ROS с использованием этих типов сообщений под разные задачи. Задание 1 Напишите программу с использованием модели сообщения подписчик/издатель, в которой в программе издателя пользователь вводит строку, а в программе подписчике эта строка переворачивается и выводиться на экран. Пример ввода программы издателя:

```
Съешь ещё этих мягких французских булок, да выпей чаю
```

Вывод программы подписчика:

```
юач йепыв ад ,колуб хиксзуцнарф хикгям хитэ ёще ьшеъС
```

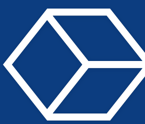
Задание 2 Напишите программу с типом сообщения сервис, у которой на стороне клиента будут вводится коэффициенты A, B, C квадратного уравнения, а севре должен будет проверять, есть ли у уравнения корни, и если есть, то находить их. Формат .srv сообщения должен быть следующим:

```
float64 A
float64 B
float64 C
---
string exist
float64 x1
float64 x2
```

Где в части запроса перечисляются коэффициенты уравнения, а в части ответа есть строка, в которую нужно вывести информацию о том, есть ли у уравнения корни. Также в ответе есть две переменных под результат решения уравнения. Вывод в ответное сообщение со стороны сервера может выглядеть так:

```

def function(req):
    ...
    return QuadResponse(exist = 'У уравнения 2 корня', x1 = '2', x2 = '5')

```

8. Системы контроля версий



8.1 Лекция

8.1.1 История

Давайте начнем наше погружение в гит с небольшой предыстории. Таким образом, мы сможем более ясно понять, чем обусловлен текущий облик гита. Сегодня буквально каждый разработчик пользуется системой контроля версий, но так было не всегда. Первый вариант такой системы появился только в 1972 году. Она называлась Source Code Control System и была разработана Марком Рочкиндом на языке C. В этой системе был следующий функционал:

- внесение файлов для отслеживания в истории,
- извлечение конкретных версий файлов для редактирования,
- внесение новых версий с комментариями, объясняющими изменения,
- отмена изменений,
- слияние изменений,
- журнал изменений.

Тут важно отметить, что работа велась исключительно с файлами, то есть не было понятия проекта. Также в 1982 году была выпущена похожая по функционалу система Revision Control System, реализованная Уолтером Тихи. В отличие от SCCC она имела открытый код. Эти две программы принято называть системами контроля версий 1-го поколения. У них были существенные недостатки которые были призваны решить системы 2-го поколения. Это Concurrent Versions System созданная Диком Груном 1986 году и subversion (SVN) созданная компанией Collabnet Inc в 2000 году. Они начали работать именно с модулями, а не с конкретными файлами, т.е. в одном коммите уже могло присутствовать несколько файлов. В связи с этим появились ветви и возможность их слияния. Ветви это разные версии проекта (модуля). И еще одной важной особенностью систем контроля версий второго поколения стал удаленный центральный репозиторий. Таким образом, появилась возможность работать не на локальной машине. Системы контроля версий второго поколения уже решали большинство проблем, встающие перед разработчиками. Однако еще было, что улучшать. Давайте перейдем к повсеместно используемой сегодня системе - гит. Ее история началась в 2005 году и связана с одним из наиболее важных проектов с открытым кодом - ядром Linux. Линус Торвальдс в процессе создания решил создать собственную систему контроля версий, которая преследовала следующие цели:

- скорость,
- простая архитектура,
- хорошая поддержка нелинейной разработки (тысячи параллельных веток),
- полная децентрализация (теперь удаленный репозиторий не является центральным, у каждой локальной копии точно такие же возможности),
- возможность эффективного управления большими проектами, такими как ядро Linux (скорость работы и разумное использование дискового пространства).

С момента своего появления в 2005 году Git развился в простую в использовании систему,

сохранив при этом свои изначальные качества. Он удивительно быстр, эффективен в работе с большими проектами и имеет великолепную систему веток для нелинейной разработки. Сразу хочется сказать, что гит и GitHub это разные вещи. По сути GitHub, GitLab и другие сервисы только предоставляют хостинг ваших репозиториев. Но безусловно они очень сильно облегчили жизнь программистов, сделав командную разработку еще проще и приятнее. Хотелось бы еще раз отметить вклад Линуса Торвальдса в современное программирование. По сути он создал два очень важных проекта с открытым кодом - ядро Linux и гит. Теперь мы готовы двинуться дальше и рассмотреть, что конкретно под капотом у гита.

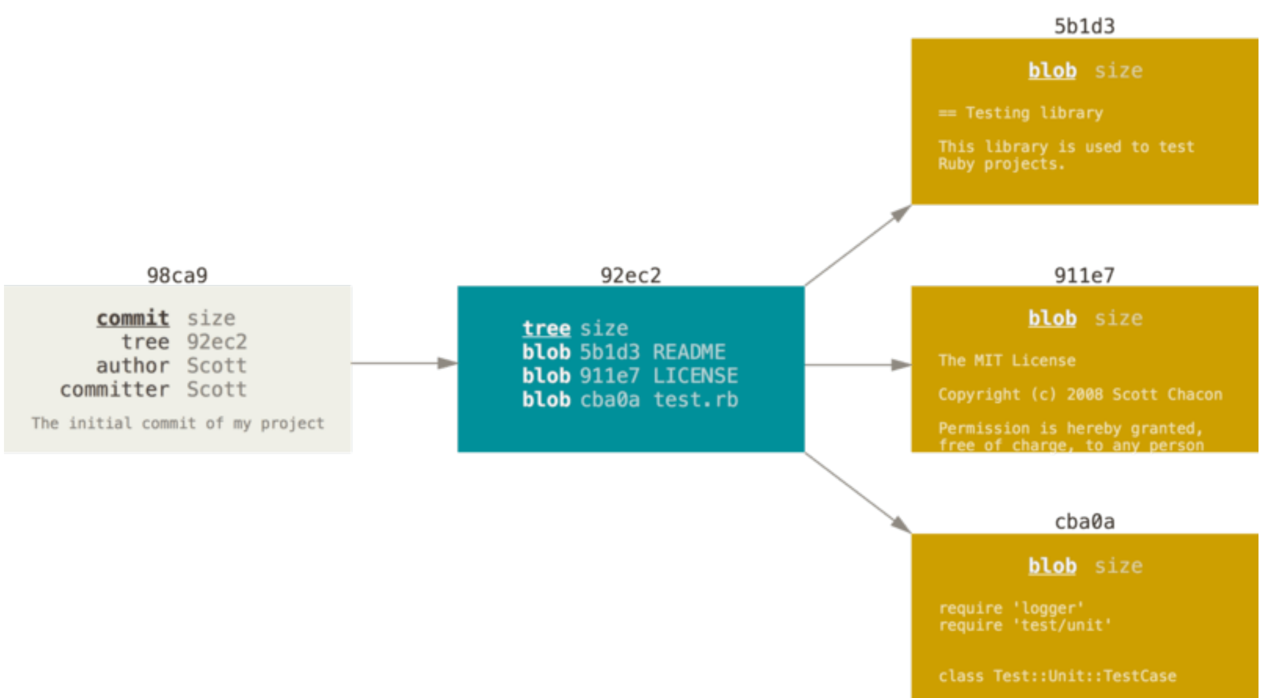
8.1.2 Зачем и когда нужен гит?

Начнем наше знакомство с гитом с вопроса, зачем нужен гит. Надеюсь, из экскурса в историю уже стала понятна боль программистов, которые начали работать до его создания, но все же давайте чуть более подробно рассмотрим эти вопросы. Давайте обратимся к определению. Git — распределённая система управления версиями. Во-первых, это все-таки система управления версиями. То есть она предоставляет возможность возвращаться к любой версии из прошлого (это может понадобиться, когда наши изменения оказались некорректными, и мы хотим от них избавиться) и хранить сразу несколько версий проекта (например, разные версии питона). Также он дает возможность просматривать историю изменений. Это просто необходимо, например, тогда, когда мы пытаемся понять, кто и где посадил багу. Для этого он предоставляет следующие сущности:

- commit - снимок текущего состояния изменений в проекте,
- branch - сущность для хранения отдельных версий проекта,
- репозиторий - каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы (короче говоря, папка в файловой системе становится репозиторием, когда мы выполняем в ней команду git init. Обычно вся информация о репозитории хранится в специальной папке .git в корневом каталоге проекта. По сути именно эта папка делает проект репозиторием).

Далее, мы подробнее разберем сущности этих понятий и команды для работы с ними. Также гит предоставляет возможность совершать совместную работу без боязни потерять данные или затереть чужую работу. Это становится удобным благодаря распределенности системы. Каждый разработчик может добавлять и тестировать изменения в своей локальной версии репозитория. Обычно перед добавлением кода в общий проект другие разработчики проверяют код коллег, ведь чем больше глаз, тем больше вероятность найти ошибку в коде.

Еще один важный вопрос - когда нужен гит? Ответ очень прост - всегда. Даже в случае, если вы разрабатываете что-то одни, гит будет вам помощником. Вы сможете отменять изменения, следить за историей, делать несколько версий проекта и в итоге опубликовать свой проект в GitHub. Опубликовывать свои проекты и даже домашние задания хорошая практика. Профиль на GitHub - своего рода портфолио для программиста. При отсутствии реального рабочего опыта именно такие проекты помогут найти

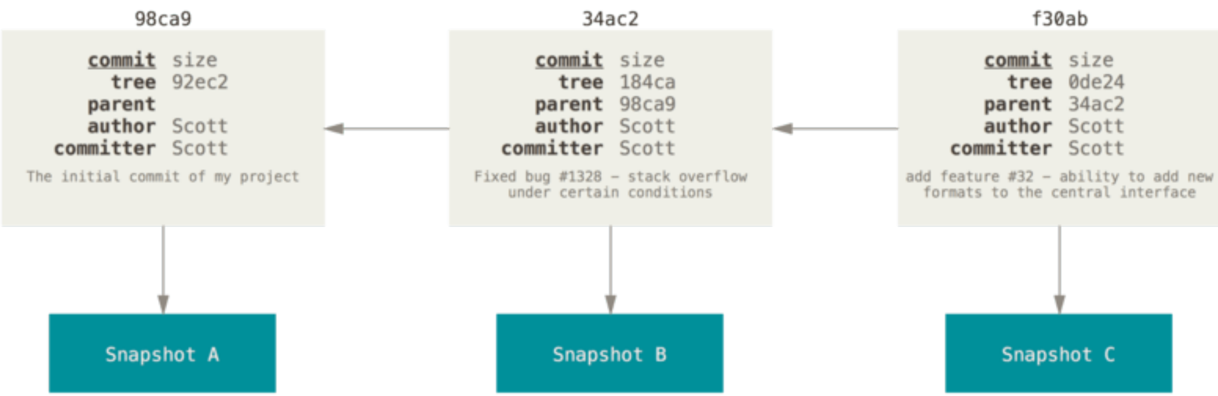


вам первую работу.

8.1.3 Коммиты

Простые вещи закончились. Теперь нам придется погрузиться в теорию. Приготовьтесь сосредоточиться и сконцентрироваться. Да, без понимания как это устроено внутри, вы сможете работать на уровне новичка, но дальше продвинуться будет сложно. Поэтому давайте разберем понятие коммита.

Коммит - единственный способ добавить новые изменения в проект. По сути это снимок изменений проекта. Из коммитов строится все дерево репозитория в гите. Давайте разберем, из чего он состоит. Смотрим на слайд слева направо. Серый прямоугольник показывает нам содержимое коммита. Во-первых, это размер коммита, ссылка на дерево изменений, автор изменений и человек который добавляет эти изменения в проект. Также над прямоугольником указанна последовательность букв и цифр. Это хэш коммита. Он необходим для навигации между коммитами и является его индивидуальным номером. Давайте посмотрим следующий прямоугольник. Это дерево изменений. Ссылка на него указана в поле tree рассматриваемого коммита. Это сущность, которая объединяет изменения разных файлов в единый коммит. Изменение каждого файла называется blob. В данном случае мы поменяли три файла и получили три blob'а. В каждом таком блобе содержатся именно совершенные изменения, а не содержимое файлов. Это сильно экономит место на диске. Давайте повторим еще раз. Мы делаем изменения в проекте и добавляем их в дерево изменений командой git add. Таким образом сформировав дерево изменений мы делаем git commit и добавляем эти изменения в репозиторий. Важная часть - сообщение коммита. Нам требуется описать изменения, которые мы добавляем в проект.



Также при создании коммита важно помнить, что один коммит должен исправлять внутри себя одну багу или добавлять единственный новый функционал.

Теперь рассмотрим как коммиты связаны между собой. Пока что рассмотрим два типа коммитов: с родителем и без родителя. Единственный коммит без родителя - это инишиал коммит, то есть самый первый коммит в проекте. От него начинается все остальное дерево гита. Остальные коммиты уже имеют как минимум одного родителя. Как видно на слайде левый коммит является первоначальным и поле parent у него пустое, а последующие коммиты уже указывают друг на друга. Можно поставить на паузу и сравнить хэши коммитов, которые как раз указываются в поле parent. Эта связь необходима, так как коммиты хранят только изменения, и чтобы получить итоговую версию проекта, нам нужно пройтись по этому дереву до первоначального коммита. Тут не стоит пугаться, весь этот функционал уже реализован и спрятан от нашего взгляда. Все работает очень интуитивно.

Мы поняли что такое коммит, из чего он состоит и как коммиты связаны между собой. В следующей лекции разберемся с ветвлением.

8.1.4 Ветвление

Мы рассмотрели, что такое коммиты и как они связаны между собой и теперь готовы перейти к понятию ветки (на английском branch). Давайте посмотрим на слайд. Серые прямоугольники - коммиты, красные - ветки. Тут можно сразу понять, что ветка это не что иное, как указатель на коммит. В данном примере у нас есть две ветки master и testing.

Тут появляется сущность HEAD. Голова определена только для локального репозитория и указывает, с какой веткой мы сейчас работаем. В обычном состоянии голова указывает на ветку, но она также может указывать на какой-то конкретный коммит. Тогда она будет в состоянии оторванная голова (detached head). Вернемся к веткам. Получается в этом случае голова указывает на ветку мастер и сейчас мы работаем именно с ней.

Если мы хотим перейти на ветку testing нам нужно выполнить команду git checkout testing. Именно эта команда меняет ветку, на которую указывает голова. Давайте рас-

смотрим эту ситуацию. Мы работали в ветке мастер, но решили сделать какой-то новый функцианал, но не уверены что он будет работать. Поэтому мы переключились на ветку testing и решили продолжить разработку в ней.

На данном слайде видно, что мы добавили новые изменения с новым коммитом. Теперь ветки master и тестинг указывают на разные коммиты.

На слайде изображена ситуация, когда нам пришлось вернуться в мастер, например, чтобы быстро поправить какой-то баг и мы оторвались от реализации новой фичи в тестинг или другой разработчик залил свои изменения в мастер. Таким образом мы подошли к проблеме слияния веток. Допустим мы закончили работу в ветке тестинг и хотим добавить новый функционал в мастер. Если бы ветка мастер осталась без изменений, то слияние прошло бы очень просто. Ветка master стала бы просто указывать на тот же коммит что и тестинг.

В случае когда в мастере тоже есть новые коммиты, слияние становится более сложным, но эта ситуация абсолютно стандартная. Давайте рассмотрим два варианта. В первом случае в ветках велась работа над разными частями кода и они никак не пересикались. В таком случае нам просто нужен новый коммит, который будет указывать сразу на коммиты C4 и C5. Внутри себя он не будет содержать никаких изменений. Теперь когда мы будем разворачивать дерево и наткнемся на этот коммит, нам нужно будет просто применить изменения из двух веток.

Чуть более сложный вариант - мы работали над одним куском кода и поменяли его в обеих ветках. В таком случае нам предстоит решить merge конфликт, то есть выбрать изменения из какой ветки нам нужно оставить. Такой коммит по сути будет таким же, как рассматриваемый до, но уже с изменениями решающими конфликт. Коммиты с двумя родителями называются merge коммииты и имеют двух родителей. Все остальные поля у них такие же.

Таким образом мы разобрались с одной из самых сложных частей гита - ветки и слияния. Это очень удобный инструмент для разработки и без него сложно было бы представить систему контроля версий.

8.1.5 Удачная модель ветвления

Мы поговорили про ветки, теперь нам нужно понять как их создавать и как сливать, чтобы создать себе как можно меньше проблем и опыт использования гит был приятным. То что я сейчас буду рассказывать не является чем-то обязательным. Это лишь один из вариантов удачной модели ветвления. К этому можно относиться как к код стайлу. Вариантов существует множество, но все же советую вам выбрать какую-то одну модель и придерживаться ее. Так работать с гитом будет сильно легче. На слайде можно видеть пример работы четырех разработчиков. Схема показывает, что существуют 4 репозитория, ассоциируемые с разработчиками. Это локальные репозитории. Также есть общий удаленный репозиторий origin. Обычно самая полная и актуальная версия проекта лежит там. Именно этот репозиторий лежит на хостинге, например на GitHub'е. Как видно, разработчики могут обмениваться изменениями между собой в частном порядке и через origin.

Пока что рассмотрим две ветки develop и master. Эти ветки обязательно должны

присутствовать в вашем репозитории. Master - основная ветка с всегда рабочей версией проекта, а в Develop происходит создание новых модулей и исправления багов. Так у вас всегда будет рабочий проект и процедура релизов нового функционала произойдет менее болезненно. Ветка девело призвана быть буфером новых изменений. Только после прохождения автоматического или ручного тестирования в этой ветке можно двигаться дальше и добавлять изменения в мастер.

Также в этой модели предусмотрены отдельные ветки для фичей. Она может называться как угодно, но должна описывать смысл реализуемой фичи. Важно, чтобы в такой ветке реализовывалась единственная фича. Она взаимодействует (сливается и наследуется) исключительно с веткой девелоп. Количество таких веток не ограничено.

Давайте рассмотрим основные ошибки, которые люди совершают при работе с гитом. Во-первых, это git push –force. Этой командой мы хотим залить наши локальные изменения в репозиторий на хостинги, и если там есть какие-то другие коммиты, мы их затрем. Запомните эту команду и старайтесь ее никогда не использовать! Далеко не один репозиторий в моей истории пострадал от этой команды. Вторую мы уже рассмотрели. В рамках ветки с новой фичей следует реализовать функционал только относящийся к этой фиче. Это позволит получить читаемую историю изменений, откатывать ненужные фичи, локализовывать баги. И последнее о чем я хотел бы сказать: добавление в репозиторий больших файлов. Настоятельно рекомендую этого избежать, так как GitHub не предназначен для хранения больших файлов и скорость скачивания там небольшая.

Мы обсудили удобную модель ветвления, с ветками мастер, девелоп и фиче бранч. Важно помнить, изменения в мастер попадают только из девелоп. Надеюсь вы опробуете эту модель в своих проектах.

8.2 Семинар

8.2.1 Работа с google collaboratory

Для прохождения нашего курса мы рекомендуем вам пользоваться продуктом для машинного обучения гугл колаб. Он основан на juputer notebook - интерактивной среде программирования для вычислений на многих языках программирования. juputer notebook также можно использовать в рамках курса, но у вас могут возникнуть проблемы с установкой библиотек. Большой плюс гугл колаб - вычисления происходят на серверах гугла и вам нужен только браузер, чтобы выполнять задания к нашему курсу. Также гугл предоставляет доступ к графическим вычислительным процессорам (GPU), что позволяет успешно обучать нейронные сети. К сожалению, есть и проблемы. Ограниченное время доступа к сервису, и для работы необходим стабильный интернет. Давайте познакомимся с ним на практике.

8.2.2 Работа с гитом

Теперь давайте перейдем к практике. На семинаре мы рассмотрим пример использования основных команд. Создадим аккаунт на GitHub‘е, сделаем первый коммит, первый раз запушим изменения, создадим первую ветку и сделаем первый merge. Мы будем

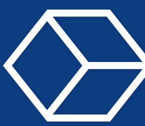
работать с гитом в колабе. Это не совсем удобно, поэтому это вынужденное решение. Конечно, проще скачать себе на компьютер GitHub Desktop или пользоваться терминальной версией. Но мне не хочется грузить вас установкой пакетов с самого начала. Если кто-то уже имеет опыт работы с гитом, можете попробовать выполнять задания в терминале, это не возбраняется. Давайте начнем!

Таким образом, мы попробовали на практике основные команды гита. Это эдд, коммит, пуш, мердж. Посмотрели как работать с GitHub'ом. Теперь вы готовы сделать домашнее задание!

8.3 Практическое занятие

Давайте рассмотрим домашнее задание. По сути вам нужно будет повторить то, что мы делали на семинаре. Для начала нужно сделать Fork, потом создать feature ветку *update_readme*. Обратите внимание, что она должна наследоваться от develop. Потом добавить в файл рид ми одну строчку и запушить изменения. Далее слить изменения в develop, добавить туда рабочий ноутбук и в конце слить всю информацию в мэйн. Также советую ознакомиться с доп материалами, если возникнут вопросы, или захочется изучить тему поглубже. Отдельно хотелось бы отметить 6 пункт в материалах к семинару. Это ссылка на интерактивный курс по гиту. Советую попробовать его пройти! Верю, что у вас все получится. Удачной работы!

9. Дополнительные главы



9.1 Стабилизация обратного маятника

```
1
2  def stabilizing_policy(x):
3      _, ang_vel = x
4
5      tau = - 1 * ang_vel
6
7      #print(tau)
8
9      return tau
10
11 def destabilizing_policy(x):
12     _, ang_vel = x
13
14     tau = ang_vel
15
16     #print(tau)
17
18     return tau
19
20 def swing_up_policy(x):
21     alpha, omega = x
22
23     tau = 0.15 * omega
24
25     if (math.cos(alpha) > 0.65):
26         #print(omega, alpha)
27         tau = - 5 * omega - 10 * math.sin(alpha)
28
29     return tau
30
31 def constant_policy(x):
32     return 5
33
34 def random_policy(x):
35     return np.random.rand() * 10 - 5
36
37 import numpy as np
38 import cv2
```

```
39 import time
40 import math
41
42 WIND_X = 700
43 WIND_Y = 600
44
45 canvas = np.zeros((WIND_Y, WIND_X, 3), np.uint8)
46
47 class Pendulum:
48     def __init__(self, l = 1, m = 1, dt = 0.01, g = 10,
49                     scale = 200, alpha0 = 2, omega0 = 0):
50         self.l  = l
51         self.m  = m
52         self.dt = dt
53         self.g  = g
54
55         self.scale = scale
56
57         self.tlim = 3 # tau_limit
58         self.olim = 5 * math.pi * 2
59
60         self.x0 = WIND_X // 2
61         self.y0 = WIND_Y // 2
62
63         self.alpha = alpha0
64         self.omega = omega0
65
66         self.I = self.m * self.l**2 / 3
67
68     def draw(self, canvas):
69         cv2.line(canvas, (self.x0, self.y0),
70             (self.x0 + int(self.scale * self.l *
71                 math.sin(self.alpha + math.pi)),
72              self.y0 + int(self.scale * self.l *
73                 math.cos(self.alpha + math.pi))),
74              (120, 230, 12), 3)
75
76     def move(self, u):
77         u = np.clip(u, -self.tlim, self.tlim)
78
79         omega_dot = (self.m * self.l * self.g / 2 *
80                         math.sin(self.alpha) + u) / self.I
```

```
81
82          self.omega += omega_dot * self.dt
83
84          self.omega = np.clip(self.omega, -self.olim, self.olim)
85
86          self.alpha += self.omega * self.dt
87
88      def get_state(self):
89          return (self.alpha, self.omega)
90
91  p = Pendulum(alpha0 = 3)
92
93  x_hist = []
94
95  while(True):
96      canvas[:, :, :] = 60
97
98      p.draw(canvas)
99
100     x = p.get_state()
101
102     x_hist.append(x)
103
104     u = swing_up_policy(x)
105
106     p.move(u)
107
108     cv2.imshow("canvas", canvas)
109
110     key = cv2.waitKey(30) & 0xFF
111
112     if (key == ord('q')):
113         break
114
115 cv2.destroyAllWindows()
116 cv2.waitKey(20)
117
118 import matplotlib.pyplot as plt
119
120 xpl, ypl = [], []
121
122 for a, o in x_hist:
```

```
    xpl.append(a)
    ypl.append(o)

fig = plt.figure(figsize = (15, 15))
ax = fig.add_subplot(111)#, projection='3d')

ax.set_xlabel("angle", fontsize=20)#, rotation=150)
ax.set_ylabel("angular velocity", fontsize=20)

plt.plot(xpl, ypl)
plt.show()

def visualize_policy_action(policy, llim1, hlim1, a1name,
                            llim2, hlim2, a2name, pnum):
    fig = plt.figure(figsize = (15, 15))
    ax = fig.add_subplot(111, projection='3d')

    # Grab some test data.
    #X, Y, Z = axes3d.get_test_data(0.05)

    X = np.linspace(llim1, hlim1, num = pnum)
    Y = np.linspace(llim2, hlim2, num = pnum)

    X, Y = np.meshgrid(X, Y)

    #set z values
    Z = X**2 + Y**2

    #print(X)
    #print(Y)

    for i in range(len(X)):
        for j in range(len(X[0])):
            x = X[i, j]
            y = Y[i, j]

            #obs = [math.cos(x), math.sin(x), y]

            t = policy((x, y))

            t = np.clip(t, -3, 3)

```

```
            Z[i, j] = t

    # Plot a basic wireframe.
    ax.view_init(30, 60)
    ax.plot_wireframe(X, Y, Z)#, rstride=10, cstride=10)

    ax.set_xlabel(a1name, fontsize=20, rotation=150)
    ax.set_ylabel(a2name)
    ax.set_zlabel(r'torque', fontsize=30, rotation=60)

    plt.show()

policies = [constant_policy, random_policy, stabilizing_policy,
                destabilizing_policy, swing_up_policy]

for policy in policies:
    visualize_policy_action(policy, -math.pi, math.pi, "angle",
        -5 * math.pi * 2, 5 * math.pi * 2, "angular_velocity", 25)

```

9.2 Симуляция упругих столкновений двумерных объектов

```

import numpy as np
import cv2
import math

WIND_X = 2100
WIND_Y = 1300

GRAY = 0

canvas = np.ones((WIND_Y, WIND_X, 3), np.uint8) * GRAY

class Figure:
    def __init__(self, color):
        self.color = color

    def draw(self, canvas):
        pass

```

```
    def move(self):
        pass

    def _move(self, x, y, vx, vy, r, ax = 0, ay = 0):
        vx += ax
        vy += ay

        x += vx
        y += vy

        if (x < r or x > WIND_X - r):
            vx *= -1
            vx -= ax

        if (y < r or y > WIND_Y - r):
            vy *= -1
            vy -= ay

        return x, y, vx, vy

class Circle(Figure):
    def __init__(self, x, y, vx, vy, r, color, m, a = 0):
        Figure.__init__(self, color)

        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy

        self.a = a

        self.m = m

        self.Fx = 0
        self.Fy = 0

        self.r = r

    def set_Fx(self, Fx, inc = False):
        if (inc == False):
            self.Fx = Fx

```

```
        else:
            self.Fx += Fx

    def set_Fy(self, Fy, inc = False):
        if (inc == False):
            self.Fy = Fy

        else:
            self.Fy += Fy

    def get_Fx(self):
        return self.Fx

    def get_Fy(self):
        return self.Fy

    def draw(self, canvas):
        cv2.circle(canvas, (int(self.x), int(self.y)),
                int(self.r), self.color, 1)

    def move(self):
        self.x, self.y, self.vx, self.vy = self._move(self.x,
                self.y, self.vx, self.vy, self.r,
                self.Fx / self.m, self.Fy / self.m)

class Line(Figure):
    def __init__(self, x1, y1, x2, y2, vx1, vy1, vx2, vy2, color, a1, a2):
        Figure.__init__(self, color)

        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.vx1 = vx1
        self.vy1 = vy1
        self.vx2 = vx2
        self.vy2 = vy2

        self.a1 = a1
        self.a2 = a2

    def draw(self, canvas):
```

```
        cv2.line(canvas, (int(self.x1), int(self.y1)),
            (int(self.x2), int(self.y2)), ((12 + 2 * self.x1) % 255,
            120, (230 + 1 * self.y2) % 255), 15)

    def move(self):
        self.x1, self.y1, self.vx1, self.vy1 = self._move(self.x1,
            self.y1, self.vx1, self.vy1, 0, self.a1)
        self.x2, self.y2, self.vx2, self.vy2 = self._move(self.x2,
            self.y2, self.vx2, self.vy2, 0, self.a2)

        #print(self.a1)

class Manager:
    def __init__(self, k = 3):
        self.k = k

        self.objects = []

    def add_object(self, new_obj):
        self.objects.append(new_obj)

    def draw(self, canvas):
        for obj in self.objects:
            obj.draw(canvas)

    def move(self):
        #self.collide()

        for obj in self.objects:
            obj.move()

    def collide(self):
        E = 0

        for obj in self.objects:
            E += (obj.vx**2 + obj.vy**2) * obj.m / 2

            obj.set_Fx(0)
            obj.set_Fy(0)

        print(E)

```

```
146         for i in range(len(self.objects)):
147             for j in range(i + 1, len(self.objects)):
148                 obj1 = self.objects[i]
149                 obj2 = self.objects[j]
150 
151                 diffx = obj1.x - obj2.x
152                 diffy = obj1.y - obj2.y
153 
154                 dist = math.sqrt((diffx)**2 + (diffy)**2)
155 
156                 depth = obj1.r + obj2.r - dist
157 
158                 if (depth > 0):
159                     F = self.k * depth
160 
161                     #alpha = math.arctg(diffy / diffx)
162 
163                     obj1.set_Fx(F * diffx / dist, inc = True)
164                     obj1.set_Fy(F * diffy / dist, inc = True)
165 
166                     obj2.set_Fx(- F * diffx / dist, inc = True)
167                     obj2.set_Fy(- F * diffy / dist, inc = True)
168 
169 manager = Manager()
170 
171 # manager.add_object(Circle(400, 700, 1, 2, 150, (120, 230, 230), 20))
172 # manager.add_object(Circle(710, 300, 3, 4, 230, (220,  30, 210), 50))
173 # manager.add_object(Circle(100, 500, 2, -3, 40, (120, 230, 230), 70))
174 # manager.add_object(Circle(310, 500, 2, 4, 60, (220, 30, 210),   5))
175 
176 manager.add_object(Line(310, 500, 230, 340, 2, 4, 6, 4, (220, 30, 210),
177                     0.123, 0.071324765213))
178 manager.add_object(Line(310, 500, 230, 340, 7, 1, 2, 5, (220, 30, 210),
179                     0.04, 0.1324765213))
180 manager.add_object(Line(310, 500, 230, 340, 2, 4, 3, 7, (220, 30, 210),
181                     0.23, 0.031324765213))
182 manager.add_object(Line(310, 500, 230, 340, 3, 1, 4, 8, (220, 30, 210),
183                     0.14, 0.051324765213))
184 
185 result = np.zeros_like(canvas)
186 r = 0.1
187 
```

```
s = 5

while (True):
    canvas[:, :, :] = GRAY

    manager.move()
    manager.draw(canvas)

    result[:, 0:WIND_X - s, :] = cv2.addWeighted(result, 1,
                                canvas, r, 0)[:, s:, :]

    cv2.imshow("canvas", result)

    key = cv2.waitKey(30) & 0xFF

    if (key == ord('q')):
        break

cv2.destroyAllWindows()
cv2.waitKey(10)

```

9.3 Обратная кинематика для плоского двузвенного манипулятора

```

import numpy as np
import cv2
import math
import copy

WIND_X, WIND_Y = 1300, 800
GRAY = 50

class Manipulator:
    def __init__(self, x0, y0, l1, l2, alpha0, beta0, color):
        self.x0 = x0
        self.y0 = y0
        self.l1 = l1
        self.l2 = l2

        self.alpha = alpha0
```

```
        self.beta  = beta0

        self.color = color

    def set_angles(self, alpha, beta):
        self.alpha = alpha
        self.beta  = beta

    def draw(self, canvas):
        (p1, p2) = self.forward_kinematics(self.alpha, self.beta)

        p1x, p1y = int(p1[0]), int(p1[1])
        p2x, p2y = int(p2[0]), int(p2[1])

        cv2.line(canvas, (self.x0, self.y0), (p1x, p1y), self.color, 1)
        cv2.line(canvas, (p1x, p1y), (p2x, p2y), self.color, 1)

    def forward_kinematics(self, alpha, beta):
        x1 = self.x0 + math.cos(alpha) * self.l1
        y1 = self.y0 + math.sin(alpha) * self.l1

        x2 = x1 + math.cos(alpha + beta) * self.l2
        y2 = y1 + math.sin(alpha + beta) * self.l2

        return ((x1, y1), (x2, y2))

    def inverse_kinematics(self, x2_ref, y2_ref, pnum):
        alpha_opt, beta_opt = 0, 0
        d_min = math.sqrt(WIND_X**2 + WIND_Y**2)

        alphas = np.linspace(0, 2 * math.pi, pnum)
        betas  = np.linspace(0, math.pi, pnum)

        for a in alphas:
            for b in betas:
                _, p2 = self.forward_kinematics(a, b)

                d = math.sqrt((p2[0] - x2_ref)**2 + (p2[1] - y2_ref)**2)

                if (d < d_min):
                    d_min = d
                    alpha_opt = a
```

```
60                      beta_opt = b
61
62          return alpha_opt, beta_opt, d_min
63
64  refresh_flag = True
65
66  def refresh_flag_up(_):
67      global refresh_flag
68
69      refresh_flag = True
70
71  alpha_opt, beta_opt, d_min = 0, 0, 0
72
73  canvas_ref = np.ones((WIND_Y, WIND_X, 3), np.uint8) * GRAY
74
75  cv2.namedWindow("canvas")
76  cv2.createTrackbar("pnum", "canvas", 233, 300, refresh_flag_up)
77
78  m = Manipulator(WIND_X // 2, WIND_Y // 2, 390, 300, 1, 1, (120, 40, 240))
79
80  pnum = 0
81
82  for i in range(int(2 * math.pi * 50 * 30)):
83      x_target, y_target = int(WIND_X * 0.75)  +
84      int(200 * math.sin(0.05 * i)), WIND_Y // 2 +
85      int(300 * math.cos(0.07 * i))
86
87      cv2.circle(canvas_ref, (x_target, y_target), 3, (70, 90, 10), -1)
88
89  i = 0
90
91  while(True):
92      canvas = copy.deepcopy(canvas_ref)
93
94      x_target, y_target = int(WIND_X * 0.75)  +
95              int(200 * math.sin(0.05 * i)),
96              WIND_Y // 2 + int(300 * math.cos(0.07 * i))
97
98      i += 1
99
100     cv2.circle(canvas, (x_target, y_target), 3, (120, 230, 10), -1)
101
```

```
#     if (refresh_flag == True):
    pnum = cv2.getTrackbarPos("pnum", "canvas")

    alpha_opt, beta_opt, d_min = m.inverse_kinematics(x_target,
                                                y_target, pnum)

#         refresh_flag = False

    #window_name = 'Image'
    #canvas = cv2.putText(canvas, str(pnum) + "    " + str(d_min)[:5],
                        (100, 50), cv2.FONT_HERSHEY_SIMPLEX,
    #                   1, (255, 0, 0), 2, cv2.LINE_AA)

    m.set_angles(alpha_opt, beta_opt)
    m.draw(canvas)

    #print(alpha_opt, beta_opt)

    cv2.imshow("canvas", canvas)

    key = cv2.waitKey(40) & 0xFF

    if (key == ord('q')):
        break

cv2.destroyAllWindows()
cv2.waitKey(10)

```

9.4 Стерео

Рассмотрим устройство цифровых камер, которые применяются в современных автономных роботах, бинокулярного зрения, обработки стереоизображений.

Начнем с камеры, которую мы будем называть «обычной». Это камера, работающая в оптическом диапазоне с трехканальной картинкой на выходе. То есть выдающая массив из чисел скажем 1920 на 1080 на 3, где 3 - это красный, зеленый, синий, 30 раз в секунду. Такие камеры стоят в ноутбуках, веб-камеры тоже обычно такие, и во многих любительских роботах применяются именно они.

Они работают как пассивные датчики, то есть регистрируют свет, приходящий извне, с помощью массива светочувствительных элементов. Камеры, которые снимают на пленку, тут бы не подошли, а цифровые камеры переводят свет в электрические сигналы, и именно это позволяет использовать их в робототехнике. Светочувствительные элементы

для всех трех базовых цветов используются одинаковые, но над ними расположены светофильтры в таком вот шахматном порядке. Элементов, чувствительных к зеленому, тут в два раза больше, и поэтому чувствительность самая высокая именно в этой части спектра.

Мы уже говорили о цветовых пространствах, и тут можно сказать, что используется цветовое пространство RGB. Значение сигнала в каждом из каналов пропорционально мощности приходящего света.

С камерами бывают разные чудеса. Можно этого не замечать, но обычные пользовательские камеры постоянно поднастраиваются, чтобы картинка была сбалансированной по яркости белого в кадре, фокусируются. Лучше всего это заметно, если поднести ладонь близко к веб-камере ноутбука. Для звонков по скайпу это замечательно, а вот если некоторый фильтр однажды был настроен, то после такого срабатывания одному изготовителю известного алгоритма его снова нужно будет настраивать. Но автоматическую настройку всевозможных параметров обычно можно программно отключить, например с помощью opencv или usb драйвера v4l2.

Однажды я купил несколько дешевых камер в переходе, чтобы с ними экспериментировать. И заметил, что при некоторых условиях на картинке появлялись белые пятна, а если держать камеру рукой, то они пропадали. Оказалось, что поскольку камера была очень низкого качества, пластик корпуса был полупрозрачным, и свет попадал внутрь не только через объектив, но еще и с другой стороны. В итоге я обклеил камеру пластырем, и это помогло.

У камеры есть много важных параметров, включая и баланс белого, и фокусное расстояние, и выдержку, но поговорим сейчас про те, на которые нужно особенно обращать внимание при выборе камеры. Баланс белого можно настроить, фокусное расстояние зачастую в принципе тоже можно, выдержку можно регулировать, а вот три следующих характеристики останутся неизменны. Это разрешение, углы обзора и тип затвора. Поговорим про них в отдельности.

С разрешением проще всего. Чем больше разрешение, тем выше детализация объектов на изображении и тем больше данных нужно кодировать, передавать и хранить. Если очень грубо уподобить человеческий глаз камере, то разрешение его будет порядка 100 мегапикселей. Понятно, что тонкостей масса, и плотность расположения колбочек и палочек на сетчатке непостоянна, и сигналы на самом деле передаются не от каждой светочувствительной клетки в отдельности, но порядок величины можно почувствовать. Угловое разрешение глаза, то есть минимальный угол между объектами, при котором они воспринимаются как различные, составляет около трех сотых градуса и близок к таковому у хороших современных камер.

Тут нужно понимать, что чудес не бывает и что можно сделать разрешение камеры хоть пятьсот мегапикселей, но чтобы это имело смысл, нужна соответствующая оптика. В конце концов, есть дифракционный предел, да и коэффициент преломления в материале линзы на самом деле зависит от длины волны.

И о скорости обработки данных тоже не стоит забывать: бывает такое, что робототехники вынуждены ограничивать разрешение потока с камеры, чтобы поддерживать достаточную частоту обработки кадров. В одном из роботов на основе смарт-камеры,

то есть компьютера в сущности достаточно маломощного, мы в погоне за скоростью и экономией памяти использовали разрешение в шестнадцать раз меньше максимально возможного.

Еще одна важная характеристика камеры, а если более конкретно, то оптики, - это углы обзора. Чем шире обзор, тем большая часть сцены попадает в поле зрения. С другой стороны, снижается уровень детализации. У широкоугольных линз масса достоинств, в частности и для применения в автономной робототехнике. Когда на наших роботах оптика поменялась с обычной на широкоугольную, робот смог гораздо меньше крутить головой, а видел он при этом больше, чем раньше. Когда он стоит в центре поля, он может одновременно наблюдать все четыре стойки ворот. Есть и оборотная сторона - картинка становится сильно искаженной. Этот эффект рыбьего глаза еще можно увидеть на камерах GoPro и широкоугольных камерах на телефонах. На краю картинки плотность пикселей на квадратный сантиметр сцены сильно ниже, чем в центре.

Если на роботе используются нейросети, при их обучении должна учитываться вариативность в том, как может выглядеть один и тот же объект в разных частях кадра.

Менее очевидная, но тоже очень важная характеристика камеры - это тип затвора, или шаттера, в обиходе он так называется. В обычных камерах используется так называемый бегущий (или сканирующий) затвор, или rolling shutter. Снятие данных со светочувствительной матрицы происходит строка за строкой, и это занимает какое-то время, обычно порядка миллисекунд. Если за это время объект успевает сдвинуться, то на снимке он будет искажен.

Есть так называемый глобальный затвор, или global shutter. В этом случае информация со всей матрицы считывается одномоментно. Тут есть свои минусы: больший размер каждого пикселя, большее энергопотребление и как следствие более сильный нагрев. Обычно у global shutter камер небольшое разрешение, а стоят они в сравнении с обычными очень дорого. Но на динамичных роботах, включая беспилотные машины, обычно применяются именно они, поскольку допускать искажения линий разметки за счет особенностей затвора тут нельзя.

Поговорим о том, как происходит формирование изображения при съемке. Есть сцена, то есть трехмерные объекты в мире, с поверхности которых в сторону камеры излучаются фотоны. Нас интересует, в каких пиксельных координатах, то есть где на изображении, окажется каждый из видимых объектов. То есть хочется построить функцию, которая отображает три координаты в мире в две координаты на изображении. Оказывается, что при построении полноценной модели камеры необходимо учесть следующие факторы.

Во-первых, это внутренние параметры камеры. Давайте разберемся, что они из себя представляют. В массивах нумерация обычно начинается с нуля, а на изображении удобно отсчитывать пиксели, откладывая их от оптической оси. Помимо этого, камеры не бывают идеальными, и на самом деле оптическая ось обычно проходит не через середину матрицы. c_x и c_y - это пиксельные координаты оптической оси относительно угла матрицы. А f_x и f_y - это фокусное расстояние, выраженное в пикселях.

Второй фактор при построении изображения - это так называемая дисторсия, возникающая за счет особенностей конкретной оптической системы. Она бывает положительной и отрицательной, наиболее привычной будет скорее всего эффект рыбьего глаза, или fish

eye. Соответствующие преобразования описываются вот таким образом. Коэффициенты k_1, k_2 и так далее находят в процессе калибровки камеры, как и перечисленные выше внутренние параметры.

Для калибровки используются несколько десятков фотографий калибровочного шаблона, обычно это шахматная доска, с разных ракурсов. На них сначала находятся углы, а потом запускается итерационный процесс, который находит и коэффициенты дисторсии, и внутренние параметры камеры, и положения точек съемки.

В конечном итоге все эти вещи нужны для того, чтобы из пиксельных координат, которые возвращает модуль зрения, получать информацию о том, где в мире находятся объекты. Из внутренних параметров камеры можно понять, как в мире расположен луч, проходящий через фокус и через конкретный пиксель матрицы. Но для определения расстояния до объекта еще нужно понимать, где на этом луче объект находится, так что по одной камере нельзя выяснить положение объектов без какой-либо дополнительной информации о сцене.

Часто бывает, что это самое дополнительное предположение о сцене состоит в том, что объекты находятся на плоскости. В частности, такое предположение в ходу в робофутболе, и это оправданно. Получается, что луч, проходящий через пиксель матрицы и фокус, в одной точке пересекает плоскость земли, и именно в этой точке по логике вещей находится объект, найденный модулем зрения.

Когда сенсоры и вычислительные мощности позволяют, используется представление объектов как групп точек в трехмерном пространстве. Стереозрение позволяет увидеть в объеме настоящие наблюдения, не предсказания, и поэтому у него вообще говоря шире область применения, но разумеется для построения работающей системы стереозрения нужно решить тысячу технических проблем: стереокалибровку, синхронизацию камер, ну и канал для передачи данных должен быть достаточно широким.

Бинокулярное зрение основывается на том, что для разных точек съемки один и тот же объект будет выглядеть чуть-чуть по-разному. И чем ближе объект к камере, тем больше расхождение между изображениями. Например, собственный нос человек видит с двух разных сторон без пересечения. Еще одна забавная вещь в том, что пока человеку не напомнишь, что он видит нос, он его не видит.

Расхождение в пиксельных координатах на двух изображениях называется диспаратностью. Для стереопары, то есть двух изображений одного и того же, снятых одновременно с разных точек, можно построить карту диспаратности. В ней для каждого пикселя будет находиться расхождение пиксельных координат между снимками. Для простоты стереопары обычно делают горизонтальными или приводят к горизонтальным. В них диспаратность есть только по оси x, и при этом объект на правой картинке всегда левее.

Казалось бы, для того чтобы с двумя глазами стереозрение работало, у них должны пересекаться области видимости. Но вот у голубей например они практически не пересекаются. Однако эти птицы прекрасно ориентируются в пространстве. В полете точка съемки для каждого глаза меняется достаточно быстро и так просто за счет линейного перемещения, а на земле они двигают головой вперед-назад, чтобы постоянно обновлять карту диспаратности.

Диспаратность сама по себе не очень интересна, это какие-то пиксели, на которые сдвинут каждый объект. Но ее можно перевести в карту глубины, и тогда каждому пикселю будет соответствовать его глубина в сцене. Для такого преобразования нужны будут описанные выше внутренние параметры камер и информация об их взаимном расположении.

Поговорим о том, какими бывают стереокамеры. Они могут быть с активной подсветкой и без. Сначала поговорим о камерах вроде Intel RealSense или Microsoft Kinect, которые подсвечивают сцену в невидимой для человека части спектра, а именно в инфракрасном диапазоне. В RealSense например есть обычная камера оптического диапазона, то есть с тремя каналами, есть инфракрасный проектор, который накладывает специальный шаблон на окружающие предметы, и две инфракрасные камеры, снимающие этот шаблон с разных точек. При этом вся обработка происходит на устройстве, и для пользователя realsense выглядит как волшебная камера, которая снимает объекты в объеме. Электрическая мощность у нее при этом около 5 ватт она может питаться через type-c кабель, и эта камера - частый выбор для установки на автономного робота. Помимо всего прочего, у нее есть модификации с global shutter-ом.

Такие камеры работают в темноте, в отличие от пассивных датчиков.

В робофутболе сенсоры с активной подсветкой запрещены, поскольку роботы должны быть достаточно похожи на людей. Остается только использовать стерео второго типа, то есть на обычных пассивных камерах. Вопрос: как можно для пикселя с левого кадра найти, где он на правом? Понятно, что нужно построить описание, желательно уникальное для этого пикселя, да еще и такое, которое было бы удобно сравнивать с описаниями пикселей с другого кадра.

Мы рассмотрим две задачи. Первая - это нахождение соответствия между двумя пикселями, относящимися к одному и тому же объекту в общем случае, то есть когда окрестность может быть и повернута, и растянута. Это может пригодиться, если хочется найти сдвиг и поворот камеры между двумя кадрами. А вторая - это построение полной карты глубины для выровненных друг относительно друга картинок.

В принципе достаточно хорошее описание пикселя - это его окрестность. Но проблема здесь в том, что окрестность размера один, то есть только сам пиксель, в принципе не уникальна, а окрестности большего размера не инвариантны к поворотам. Проще говоря, окрестность может выглядеть по-разному на разных снимках. Так что такие описания можно использовать для решения второй задачи - для построения карты глубины по выровненной стереопаре.

Так называемое блочное сопоставление, или block matching, основывается на том, чтобы для окрестности пикселя с одного изображения найти такой сдвиг, при котором на другой снимок эта окрестность накладывается наилучшим образом. Мера несоответствия изображений - это или сумма модулей разностей, или сумма квадратов разностей. То есть происходит следующее. Берется окрестность каждого пикселя с левого снимка. Для нее перебираются разные сдвиги и с такими сдвигами окрестность сравнивается с тем, что находится на другом изображении. Ответом считается сдвиг, на котором достигается минимальное различие.

У блочного сопоставления есть много настроечных параметров, и зачастую к резуль-

тату бывает нужно применять фильтрацию, сглаживание и другую постобработку.

А вот если нужно построить преобразование, переводящее первую точку съемки во вторую, можно находить соответствия не для всех пикселей, а только для небольшого их числа, зато для очень информативных. Обычно на изображениях бывают не просто россыпи разноцветных пикселей, а пиксели, объединенные в группы по признаку принадлежности к некоторому объекту. И это сопряжено с пространственной близостью на кадре. Нужно искать соответствие между углами и другими примечательными точками на двух кадрах, а не между всеми подряд.

Давайте разберемся, почему так. Если посмотреть на равномерно залитую неоновым светом однотонную стену, глазу зацепиться будет не за что, потому что все пиксели стены плюс-минус одинаковые. Если на стене нарисована вертикальная контрастная линия, то можно, по крайней мере, удерживать внимание на ней. А если их две и они пересекаются, то такая точка становится отличным ориентиром для нахождения на двух изображениях сразу. Примерно из таких соображений точки для сопоставления на двух изображениях и выбираются. Они еще называются ключевыми точками, или key points. Точки на прямой имеют большое значение численно найденного градиента, то есть грубо говоря перепада яркости, а для пересечения прямых это верно в двух направлениях. Хорошая ключевая точка - это к примеру угол.

После того как были найдены ключевые точки, нужно построить для них описания. Чтобы описания можно было удобно сопоставлять между собой, для них нужно определить меру различия, или метрику. Оказывается, что достаточно хорошо работают описания в виде векторов, для которых можно находить обычные евклидовы расстояния.

Поговорим более подробно о том, как строится одно из самых широко используемых векторных описаний окрестности пикселя - SIFT, или Scale-Invariant Feature Transform. В 16 квадратах 4 на 4 вокруг точки строятся распределения направлений градиентов, и затем они записываются в единый стадвадцативосьмимерный вектор с разными нормализациями, чтобы сделать векторы инвариантными к поворотам и аффинным преобразованиям, насколько это возможно.

Ну и после этого можно за два вложенных цикла найти для каждой точки пару.

У представления мира в трехмерном формате, будь то карта глубины или облако точек, есть много разных применений. Приведем некоторые из них.

Например, использование облаков точек сильно упрощает детектирование противников в робофутболе. Обход противников был важной составляющей нашей тактики в чемпионате 2021 года, где мы заняли первое место. По точкам в трехмерном пространстве, которые видит робот, можно построить приближение плоскости земли. А все то, что в этой плоскости не находится и при этом достаточно высокое, - это препятствия, ну или противники.

Одно из занятных применений карты глубины - это интерполяция кадра для двух точек съемки. Предположим, что у нас есть два изображения одной и той же сцены при разных положениях камеры. Проще говоря, есть стереопара. Так вот, если карта диспаратности - это грубо говоря на сколько позиций нужно сдвинуть пиксели левой картинки, чтобы получить правую, то можно домножить ее на 0.5 и получить то, на сколько позиций нужно сдвинуть пиксели левой картинки, чтобы получить картинку,

снятую с виртуальной камеры, находящейся между двумя реальными.

9.5 Визуализация трехмерных объектов

```
import numpy as np
import cv2
import time
import math
import copy

axes = {"x" : 0, "y" : 1, "z" : 2}

class Figure:
    def __init__(self):
        pass

    def draw(self, canvas, emitter):
        pass

    def move(self):
        pass

class Light_emitter:
    def __init__(self, x0, y0, z0, color):
        self.x = x0
        self.y = y0
        self.z = z0
        self.color = color

    def get_coords(self):
        return np.array([self.x, self.y, self.z], np.float64)

    def get_color(self):
        return self.color

class Circle(Figure):
    def __init__(self, x0, y0, z0, r, color):
        Figure.__init__(self)
        self.x = x0
        self.y = y0
        self.z = z0

```

```
39         self.r = r
40         self.color = color
41 
42     def draw(self, canvas, emitter):
43         canvas.draw_3d_circle((self.x, self.y, self.z), self.r, self.color)
44 
45     def move(self):
46         self.z += 0.1
47 
48 class Line(Figure):
49     def __init__(self, x10, y10, z10, x20, y20, z20, color):
50         Figure.__init__(self)
51         self.x1 = x10
52         self.y1 = y10
53         self.z1 = z10
54         self.x2 = x20
55         self.y2 = y20
56         self.z2 = z20
57 
58         self.color = color
59 
60     def draw(self, canvas, emitter):
61         canvas.draw_3d_line((self.x1, self.y1, self.z1), (self.x2,
62                             self.y2, self.z2), self.color)
63 
64     def move(self):
65         self.z1 += 0.1
66         self.z2 += 0.04
67 
68 class Triangle(Figure):
69     def __init__(self, p1, p2, p3, color):
70         self.p1 = np.array(p1, np.float64)
71         self.p2 = np.array(p2, np.float64)
72         self.p3 = np.array(p3, np.float64)
73 
74         self.color      = color
75         self.curr_color = self.color
76 
77     def draw(self, canvas, emitter, shift = np.zeros((3), np.float64)):
78         self.calc_lighting(emitter, shift)
79 
80         canvas.draw_3d_triangle(self.p1, self.p2, self.p3,
```

```
                    self.curr_color, shift)

    def get_midpoint(self):
        return (self.p1 + self.p2 + self.p3) / 3

    def calc_lighting(self, emitter, shift):
        # normal vector calculation
        A = self.p2 - self.p1
        B = self.p3 - self.p1

        Nx = A[1] * B[2] - A[2] * B[1]
        Ny = A[2] * B[0] - A[0] * B[2]
        Nz = A[0] * B[1] - A[1] * B[0]

        N = np.array([Nx, Ny, Nz], np.float64)
        l_N = np.linalg.norm(N)

        #print(N)
        #print(l_N)

        N /= l_N

        # vector from triangle to emitter
        from_tr_to_em = emitter.get_coords() - self.get_midpoint() - shift
        l_ftr = np.linalg.norm(from_tr_to_em)
        from_tr_to_em /= l_ftr

        # lighting calculation
        cosine = np.dot(N, from_tr_to_em)

        #print(cosine)

        lighting = max(0, cosine)

        em_color = emitter.get_color()

        self.curr_color = (self.color[0] * em_color[0] / 255 * lighting,
                           self.color[1] * em_color[1] / 255 * lighting,
                           self.color[2] * em_color[2] / 255 * lighting)

        #print(self.curr_color)

```

```
    def _rotate_point(self, p, theta, axis = "x"):
        rot_mat = np.array([[math.cos(theta), -math.sin(theta)],
                            [math.sin(theta), math.cos(theta)]])

        vec = np.zeros((2), np.float64)

        ind = 0

        for i in range(3):
            if (i != axes[axis]):
                vec[ind] = p[i]
                ind += 1

        rotated = np.dot(rot_mat, vec)

        ind = 0

        for i in range(3):
            if (i != axes[axis]):
                p[i] = rotated[ind]
                ind += 1

        return p

    def rotate(self, theta, axis):
        self.p1 = self._rotate_point(self.p1, theta, axis)
        self.p2 = self._rotate_point(self.p2, theta, axis)
        self.p3 = self._rotate_point(self.p3, theta, axis)

    #def move(self):
    #    self.rotate(0.12, "x")

class Triangle_mesh(Figure):
    def __init__(self, center, rot_vel = {}):
        Figure.__init__(self)

        self.triangles = []

        self.center = np.array(center, np.float64)

        self.rot_vel = rot_vel

```

```
165     def add_triangle(self, new_triangle):
166         self.triangles.append(new_triangle)
167 
168     def generate_triangulation(self):
169         pass
170 
171     def draw(self, canvas, emitter):
172         for tr in sorted(self.triangles,
173                 key = lambda tr: - tr.get_midpoint()[2]):
174             tr.draw(canvas, emitter, self.center)
175 
176     def rotate(self, theta, axis):
177         for tr in self.triangles:
178             tr.rotate(theta, axis)
179 
180     def move(self):
181         for k, v in self.rot_vel.items():
182             self.rotate(v, k)
183 
184 class Two_dim_func(Triangle_mesh):
185     def __init__(self, center, w, h, color, pnum, z0, func):
186         Triangle_mesh.__init__(self, center)
187 
188         self.w = w
189         self.h = h
190 
191         self.color = color
192         self.pnum  = pnum
193 
194         self.z0 = z0
195 
196         self.func = func
197 
198         self.generate_triangulation()
199 
200     def generate_triangulation(self):
201         xvals = np.linspace(- self.w / 2, self.w / 2, self.pnum)
202         yvals = np.linspace(- self.h / 2, self.h / 2, self.pnum)
203 
204         for i in range(self.pnum - 1):
205             for j in range(self.pnum - 1):
206                 p1 = (xvals[i], yvals[j], self.z0 +
```

```
207                     self.func(xvals[i], yvals[j]))
208                 p2 = (xvals[i], yvals[j + 1], self.z0 +
209                     self.func(xvals[i], yvals[j + 1]))
210                 p3 = (xvals[i + 1], yvals[j], self.z0 +
211                     self.func(xvals[i + 1], yvals[j]))
212                 p4 = (xvals[i + 1], yvals[j + 1], self.z0 +
213                     self.func(xvals[i + 1], yvals[j + 1]))
214 
215                 self.add_triangle(Triangle(p1, p3, p2, self.color))
216                 self.add_triangle(Triangle(p2, p3, p4, self.color))
217 
218 class Cylinder(Triangle_mesh):
219     def __init__(self, center, r, h, color, pnum, rot_vel):
220         Triangle_mesh.__init__(self, center, rot_vel)
221 
222         self.r = r
223         self.h = h
224 
225         self.color = color
226         self.pnum  = pnum
227 
228         self.generate_triangulation()
229 
230     def generate_triangulation(self):
231         yvals = np.linspace(0, self.h, self.pnum)
232         avals = np.linspace(0, 2 * math.pi, self.pnum)
233 
234         for i in range(self.pnum - 1):
235             for j in range(self.pnum - 1):
236                 p1 = (self.r * math.cos(avals[i]), yvals[j],
237                     self.r * math.sin(avals[i]))
238                 p2 = (self.r * math.cos(avals[i]), yvals[j + 1],
239                     self.r * math.sin(avals[i]))
240                 p3 = (self.r * math.cos(avals[i + 1]), yvals[j],
241                     self.r * math.sin(avals[i + 1]))
242                 p4 = (self.r * math.cos(avals[i + 1]), yvals[j + 1],
243                     self.r * math.sin(avals[i + 1]))
244 
245                 self.add_triangle(Triangle(p1, p2, p3, self.color))
246                 self.add_triangle(Triangle(p2, p4, p3, self.color))
247 
248 class Ellipsoid(Triangle_mesh):
```

```
249     def __init__(self, center, r, h, color, pnum, rot_vel):
250         Triangle_mesh.__init__(self, center, rot_vel)
251 
252         self.r = r
253         self.h = h
254 
255         self.color = color
256         self.pnum  = pnum
257 
258         self.generate_triangulation()
259 
260     def _rand_color(self):
261         return (np.random.randint(255), np.random.randint(255),
262             np.random.randint(255))
263 
264     def generate_triangulation(self):
265         yvals = np.linspace(- self.h, self.h, self.pnum)
266         avals = np.linspace(0, 2 * math.pi, self.pnum)
267 
268         for i in range(self.pnum - 1):
269             for j in range(self.pnum - 1):
270                 r_curr = math.sqrt(self.h**2 - yvals[j]**2) *
271                     self.r / self.h
272                 r_next = math.sqrt(self.h**2 - yvals[j + 1]**2) *
273                     self.r / self.h
274 
275                 p1 = (r_curr * math.cos(avals[i]),     yvals[j],
276                     r_curr * math.sin(avals[i]))
277                 p2 = (r_next * math.cos(avals[i]),     yvals[j + 1],
278                     r_next * math.sin(avals[i]))
279                 p3 = (r_curr * math.cos(avals[i + 1]), yvals[j],
280                     r_curr * math.sin(avals[i + 1]))
281                 p4 = (r_next * math.cos(avals[i + 1]), yvals[j + 1],
282                     r_next * math.sin(avals[i + 1]))
283 
284                 if (j != 0):
285                     self.add_triangle(Triangle(p1, p2, p3,
286                         self._rand_color()))
287 
288                 if (j != self.pnum - 2):
289                     self.add_triangle(Triangle(p2, p4, p3,
290                         self._rand_color()))
```

```
class Radially_symmetrical_obj(Triangle_mesh):
    def __init__(self, center, h, color, pnum, rot_vel, func):
        Triangle_mesh.__init__(self, center, rot_vel)

        self.h = h

        self.color = color
        self.pnum  = pnum

        self.func = func

        self.generate_triangulation()

    def _rand_color(self):
        return (np.random.randint(255), np.random.randint(255),
            np.random.randint(255))

    def generate_triangulation(self):
        yvals = np.linspace(- self.h, self.h, self.pnum)
        avals = np.linspace(0, 2 * math.pi, self.pnum)

        for i in range(self.pnum - 1):
            for j in range(self.pnum - 1):
                r_curr = self.func(yvals[j] / self.h) * self.h
                r_next = self.func(yvals[j + 1] / self.h) * self.h

                p1 = (r_curr * math.cos(avals[i]),     yvals[j],
                    r_curr * math.sin(avals[i]))
                p2 = (r_next * math.cos(avals[i]),     yvals[j + 1],
                    r_next * math.sin(avals[i]))
                p3 = (r_curr * math.cos(avals[i + 1]), yvals[j],
                    r_curr * math.sin(avals[i + 1]))
                p4 = (r_next * math.cos(avals[i + 1]), yvals[j + 1],
                    r_next * math.sin(avals[i + 1]))

                if (j != 0):
                    self.add_triangle(Triangle(p1, p2, p3,
                        self._rand_color()))

                if (j != self.pnum - 2):
                    self.add_triangle(Triangle(p2, p4, p3,
```

```
333                             self._rand_color()))
334
335 # szx - width of the whole "world"
336 # center (0, 0) is in WIND_X // 2, WIND_Y // 2
337
338 class Canvas:
339     def __init__(self, WIND_X, WIND_Y, color, szx, szy, szz = 1,
340                  name = "canvas"):
341         self.WIND_X = WIND_X
342         self.WIND_Y = WIND_Y
343
344         self.szx = szx
345         self.szy = szy
346         self.szz = szz
347
348         self.color = color
349         self.name = name
350
351         self.canvas = np.zeros((WIND_Y, WIND_X, 3), np.uint8)
352
353         self.clear_canvas()
354
355     def clear_canvas(self):
356         self.canvas[:, :, 0] = self.color[0]
357         self.canvas[:, :, 1] = self.color[1]
358         self.canvas[:, :, 2] = self.color[2]
359
360     def draw_canvas(self):
361         cv2.imshow(self.name, self.canvas)
362
363     def project_point(self, p3d):
364         x, y, z = p3d
365
366         x2d = ((x / self.szx / z + 0.5) * WIND_X)
367         y2d = ((y / self.szy / z + 0.5) * WIND_Y)
368
369         return x2d, y2d
370
371     def draw_3d_circle(self, p3d, r, color):
372         xc, yc = self.project_point(p3d)
373
374         cv2.circle(self.canvas, (int(xc), int(yc)),
```

```
            int(r / p3d[2]), color, 3)

    def draw_3d_line(self, p1, p2, color, th = 1):
        x1, y1 = self.project_point(p1)
        x2, y2 = self.project_point(p2)

        cv2.line(self.canvas, (int(x1), int(y1)),
            (int(x2), int(y2)), color, th)

    def draw_3d_triangle(self, p1, p2, p3, color, shift, th = -1):
        x1, y1 = self.project_point(p1 + shift)
        x2, y2 = self.project_point(p2 + shift)
        x3, y3 = self.project_point(p3 + shift)

        contours = [np.array([[x1, y1], [x2, y2], [x3, y3]], np.int32)]

        cv2.fillPoly(self.canvas, contours, color)

class Manager:
    def __init__(self, WIND_X, WIND_Y, color, szx, szy):
        self.canvas = Canvas(WIND_X, WIND_Y, color, szx, szy)
        self.emitter = Light_emitter(0, -3, 0, (255, 255, 255))

        self.exit_flag = False

        self.objects = []

    def add_object(self, new_object):
        self.objects.append(new_object)

    def draw(self):
        self.canvas.clear_canvas()

        for obj in self.objects:
            obj.draw(self.canvas, self.emitter)

        self.canvas.draw_canvas()

    def move(self):
        for obj in self.objects:
            obj.move()

```

```
417     def handle_keyboard(self):
418         key = cv2.waitKey(100) & 0xFF
419 
420         if (key == ord('q')):
421             self.exit_flag = True
422 
423     def exit(self):
424         return self.exit_flag
425 
426 class Randomized_surface(Triangle_mesh):
427     def __init__(self, center, w, h, color, pnum, z0, rot_vel = {}):
428         Triangle_mesh.__init__(self, center, rot_vel)
429 
430         self.w = w
431         self.h = h
432 
433         self.color = color
434         self.pnum  = pnum
435 
436         self.z0 = z0
437 
438         self.generate_triangulation()
439 
440     def generate_triangulation(self):
441         xvals = np.linspace(- self.w / 2, self.w / 2, self.pnum)
442         yvals = np.linspace(- self.h / 2, self.h / 2, self.pnum)
443 
444         heights = np.zeros((self.pnum, self.pnum), np.float64)
445 
446         for i in range(int(self.pnum)):
447             for j in range(int(self.pnum)):
448                 heights[i, j] = max((self.pnum / 2 - min(i,
449                     self.pnum - i)) / 25, (self.pnum / 2 -
450                     min(j, self.pnum - j)) / 25)
451 
452         for k in range(int(math.log2(self.pnum))):
453             step = 2**k
454 
455             print(step)
456 
457             for i in range(int(self.pnum / step)):
458                 for j in range(int(self.pnum / step)):
```

```
                    heights[int(i * step) : int((i + 1) * step),
                            int(j * step) : int((j + 1) * step)] +=
                            np.random.rand() / 3 * math.sqrt(k)

        heights = cv2.blur(heights, (3, 3))

        for i in range(self.pnum - 1):
            for j in range(self.pnum - 1):
                p1 = (xvals[i],     yvals[j],
                    self.z0 + heights[i, j])
                p2 = (xvals[i],     yvals[j + 1],
                    self.z0 + heights[i, j + 1])
                p3 = (xvals[i + 1], yvals[j],
                    self.z0 + heights[i + 1, j])
                p4 = (xvals[i + 1], yvals[j + 1],
                    self.z0 + heights[i + 1, j + 1])

                self.add_triangle(Triangle(p1, p2, p3, self.color))
                self.add_triangle(Triangle(p2, p4, p3, self.color))

color = (60, 60, 60)
WIND_X, WIND_Y = 1000, 700

manager = Manager(WIND_X, WIND_Y, color, 3, 2.25)

#manager.add_object(Circle(1, 1, 2, 50, (10, 230, 120)))
#manager.add_object(Circle(1, -1, 2, 50, (10, 230, 120)))
#manager.add_object(Circle(-1, 1, 2, 50, (10, 230, 120)))
#manager.add_object(Circle(-1, -1, 2, 50, (10, 230, 120)))

#manager.add_object(Line(-1, -1, 1, 2, 3, 1, (10, 230, 120)))

# tr1 = Triangle([-2, -1, 1], [-2, 0, 1], [-1, -1, 1], (10, 230, 120))
# tr2 = Triangle([-1, -1, 1], [-1, 1, 1], [1, -0.5, 1], (210, 20, 12))

# tr_mesh = Triangle_mesh((1, 1, 3))

# tr_mesh.add_triangle(tr1)
# tr_mesh.add_triangle(tr2)

# def two_dim_func(x, y):
#     return (x**2 + 1.5 * y**2) / 12
```

```
501
502 # plane = Two_dim_func((2, 1.4, 3), 4.4, 4.4, (230, 120, 180), 25,
503 #    0, two_dim_func)
504
505 # cylinder = Cylinder((0.1, 0.3, 4), 1.7, 2.2, (230, 120, 180), 25,
506 #                      rot_vel = {"x" : 0.13, "y" : 0.15, "z" : 0.17})
507
508 # sphere = Sphere((0.1, 0.3, 10), 5.7, 3, (230, 120, 180), 25,
509 #                      rot_vel = {"x" : 0.03, "y" : 0.05, "z" : 0.07})
510
511
512 symm = Radially_symmetrical_obj((0.1, 0.3, 10), 5.7, (230, 120, 180), 25,
513             rot_vel = {"x" : 0.03, "y" : 0.05, "z" : 0.07}, func = profile)
514
515 #rand_surf = Randomized_surface((0, 0.0, 5), 6.4, 6.4,
516 #    (18, 220, 225), 64, 0, rot_vel = {"y" : 0.03})
517
518 #rand_surf.rotate(- math.pi / 2, "x")
519
520 manager.add_object(symm)
521
522 #manager.add_object(rand_surf)
523
524 while(not manager.exit()):
525     manager.draw()
526     manager.move()
527
528     manager.handle_keyboard()
529
530 cv2.destroyAllWindows()
531 cv2.waitKey(10)
532
533 import matplotlib.pyplot as plt
534
535 params = np.linspace(-1, 1, 55)
536
537 def profile(y):
538     result = math.sqrt(1 - y**2) * (0.2 * math.sin(10 * y) + 0.5)
539     #result = math.sqrt(1 - y**2)
540
541     return result
542
```

```
plt.plot(params, [profile(y) for y in params])
plt.show()
```