Содержание

Введение 5

1. Техническое задание 6

1.1 Назначение разработки и область применения 6

1.2 Технические требования 6

2. Анализ технического задания 7

2.1 Выбор языка программирования 7

2.2 Выбор среды разработки 8

2.3 Выбор среды симулятора 11

2.4 Обзор существующих методов обучения с подкреплением 13

3. Разработка системы на структурном уровне 19

3.1 Общая структурная схема 19

3.2 Схема робота 22

3.3 Принципы работы робота 23

4. Разработка программных средств 25

4.1 Общая функциональная схема 25

4.2 Модуль нейронной сети для обучения с подкреплением 30

4.3 Модуль робота 34

4.4 Модуль формирования карты 39

5. Тестирование системы 47

5.1 Обучение системы 47

5.2 Тестирование обученной системы 48

Заключение 49

Список литературы 50

Приложение А 51

**Введение**

Одной из причин возникновения и развития современной робототехники стала необходимость во внедрении в некоторые сферы деятельности роботов с исключением человека. Эта необходимость обуславливается тем, что роботы более пригодны при выполнении рутинных, энергоемких, опасных операций.

На сегодняшний день робототехника – одно из самых перспективных направлений в области информационных технологий. Спектр применения роботов огромен.

В промышленности их используют на протяжении уже многих десятилетий. Например, сфера производства автомобилей на сегодняшний день не представляется возможной без робототехники. Уже существуют предприятия, где множество процессов по сборке автомобилей осуществляются роботами. Так же во всю развивается внедрение в автомобили и летательные аппараты автопилотов.

В сфере медицины роботы тоже нашли применение. Ведутся разработки в области имплантации и наномедицины. Возможно, в скором будущем роботы смогут лечить даже неизлечимые онкологические заболевания.

В военном деле уже существуют роботы с искусственным интеллектом, предназначенные для ведения боевых действий.

Для детей компания «Lego» выпускает наборы для создания роботов в домашних условиях.

Даже в повседневной жизни нас окружает все больше и больше роботизированных технических систем. Примеров таких систем являются роботы-пылесосы, банкоматы, автоматические двери.

1. **Техническое задание**
   1. **Назначение разработки и область применения**

Разрабатываемая программная система формирования карты помещения на основе обучения с подкреплением предназначена для исследования помещения и его отображения. Для работы системы необходимо заранее подготовить базу готовых карт.

Область применения разрабатываемой системы:

- Использование системы для демонстрации карты помещения;

* 1. **Технические требования**

Так как идет разработка программной системы, то для реализации и проверки была использована симуляция. Рассмотрим требования, предъявляемые разрабатываемой симуляцией:

- В качестве ОС была использована Windows 10

- В качестве симулятора был использован Webots

- В качестве библиотеки для обучения с подкреплением использован TensorFlow;

Рассмотрим функциональность, которой должна обладать симуляция:

1. Робот должен перемещаться по помещению и формировать ее карту.
2. Пользователь должен получать сформированную роботам карту.
3. **Анализ технического задания**
   1. **Выбор языка программирования**

В наше время существуют множество разных языков программирования для разных целей. Например:

1. C++ - язык, предназначенный для создания операционных систем, драйверов для устройств, приложений для серверов и игр. Язык был создан в начале 1980-ых годов Страуструпом. Причина его создания была в том, что язык Cи, имеющийся в то время, был неэффективен для решения задач теории очередей, а язык Симула, который мог решать задачи в этой области был очень медленным. Первым делом Страуструп усовершенствовал язык Си, добавив к нему работу с классами и объектами. В то время язык назвали Си с классами. После были добавлены такие возможности, как перегрузка функций, ссылки, константы и многое другое, и язык был назван C++.
2. C# - язык, предназначенный для разработки приложений под платформу Microsoft .Net, создания игр и веб-сервисов. Разработан Андерсом Хейлсбергом и Скоттом Вильтауматом. Этот язык относят к Си подобным языком по синтаксису. Он имеет такие возможности, как статическая типизация, полиморфизм, делегаты, события, анонимные функции и т. д. Но в отличии от С++ в нем отсутствует множественное наследование.
3. Python – язык, предназначенный для создания веб-приложений, для анализа данных, для машинного обучения и для работы с нейронными сетями. Задумка его реализация сформировалась еще в конце 1980-ых годов, а сама разработка началась 1989 года. Его создавал Гвидо ван Россум, сотрудник голландского института CWI. Язык использует динамическую типизацию, которая позволяет определять переменную только во время исполнения, и циклический сборщик мусора для менеджмента памяти. Он является мультипарадигмальным языком программирования, так как поддерживает императивное, структурное, объектно-ориентированное, функциональное программирование, а также метапрограммирование и частично аспектно-ориентированное программирование за счет декоратором и дополнительных фреймворков.
4. Java – язык, предназначенный для игр и интерфейсных приложений. Он строго типизированный объектно-ориентированный язык программирования общего назначения, созданный компанией Sun Microsystems. Сначала язык создавался Джеймсом Гослингом для бытовых электронных устройств. На данный момент используется для транслирования исходного кода в байт-код.

Язык Python очень хорошо подходит для задачи с обучением. Еще одной причиной для выбора Python послужило знание этого языка.

* 1. **Выбор среды разработки**

У Python есть не мало разных IDE, например:

1. Visual Studio Code – это IDE разработано компанией Microsoft для разных операционных систем (Windows, Linux, MacOS). Является не требовательным редактором кода для кроссплатформенной разработки приложения. Имеет такие функции, как отладка, работа с Git, IntelliSense, средства подсветки синтаксиса и рефакторинга. Он появился в 2015 году и поддерживает многие языки программирования. На рисунке 1 представлен Visual Studio Code.

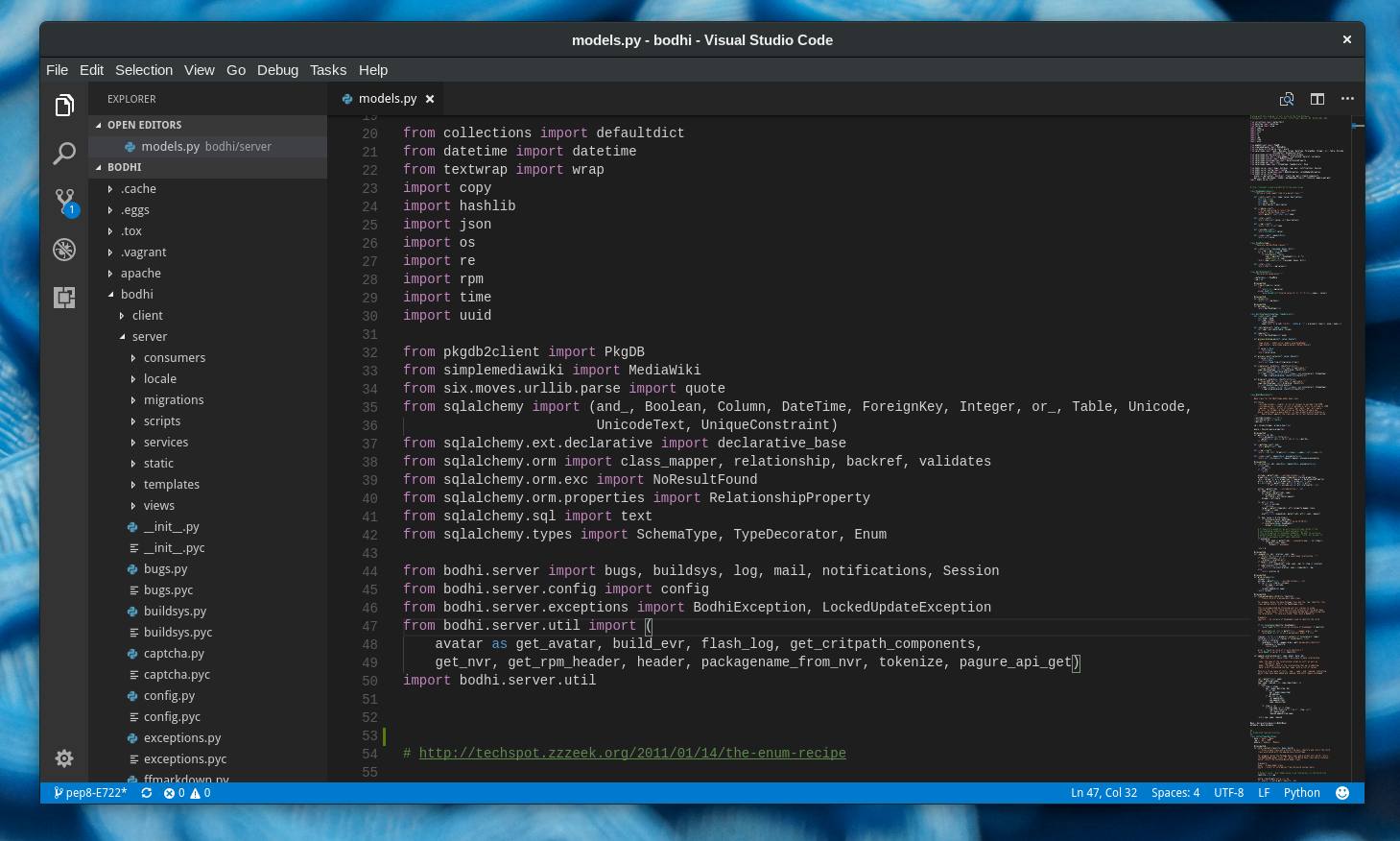


Рисунок Visual Studio Code

1. PyCharm – IDE для Python. В его наличии присутствуют такие инструменты, как анализ кода, графический отладчик, юнит-тесты, и имеет поддержку с Django. Был разработан компанией JetBrains и был выпущен для конкуренции с PyDev. На рисунке 2 представлен интерфейс PyCharm.

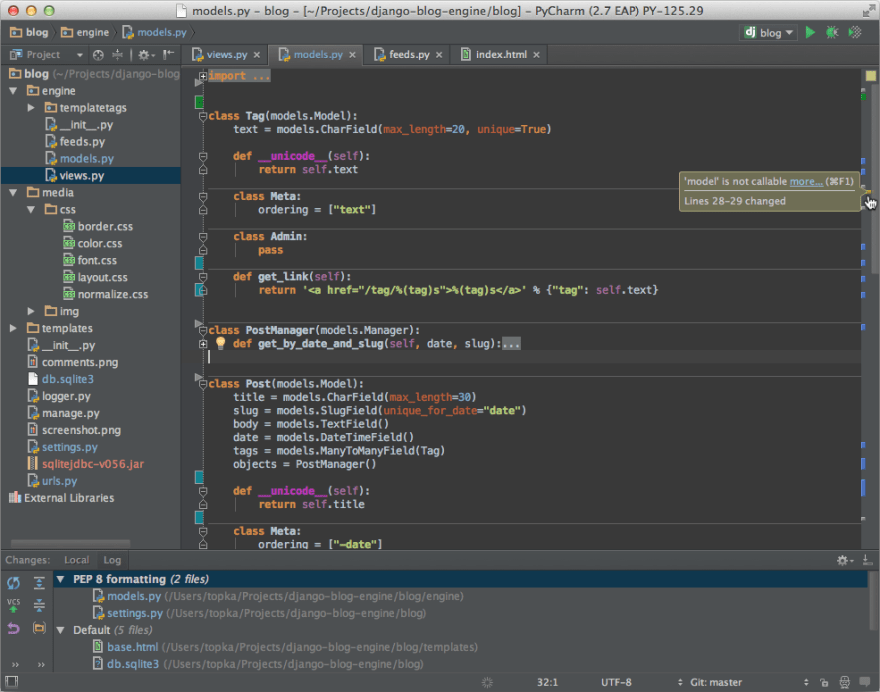


Рисунок Pycharm

1. Jupyter Notebook – IDE, созданная в 2015 году Фернандо Перезом. Представляет собой интерактивная вычислительная среда на основе веб-приложения для создания документов. Этот документ представляет собой JSON-файл, поддерживающий схему, содержащую упорядоченный список входных и выходных ячеек, которые могут содержать код, обычный текст, математические операции, графики и медиа. На рисунке 3 представлен интерфейс Jupyter Notebook

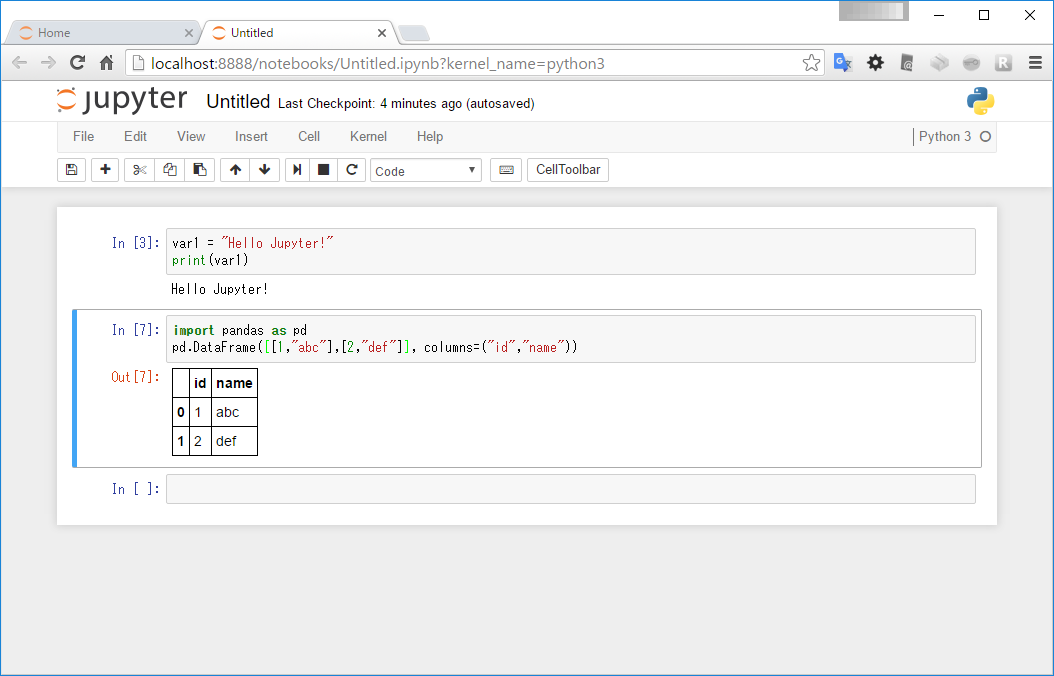


Рисунок Jupyter Notebook

1. Google Colaboratory – IDE, разработанный компанией Google для работы с большими вычислениями, позволяя использовать GPU и TPU сервера, то есть предоставляет их возможности для тех, у кого отсутствует возможности использовать их. На рисунке 4 представлен интерфейс Google Colaboratory, он очень похож на интерфейс Jupyter Notebook

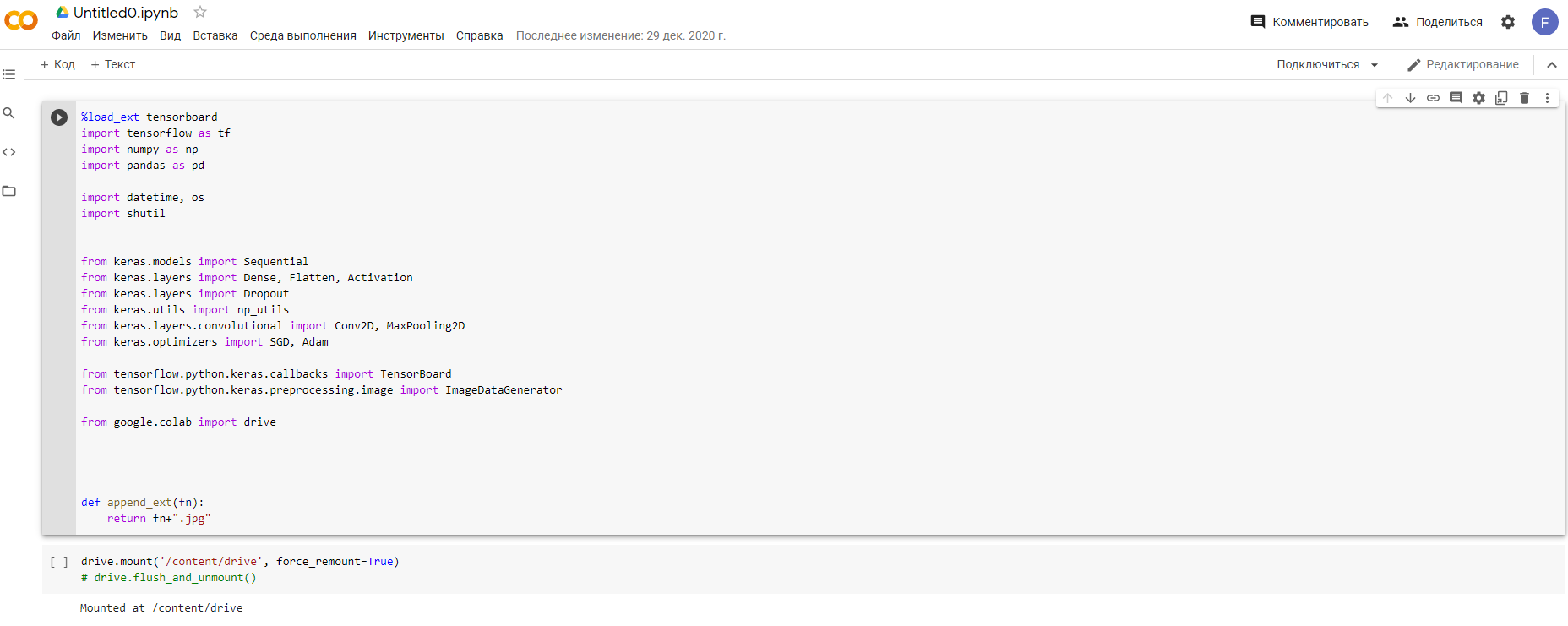


Рисунок Google Colaboratory

В работе будет использоваться Atom (продукт от GitHub) по нескольким причинам:

1. Он имеет лаконичный дизайн.
2. Огромное количество плагинов для удобного написания, запуска и компилирования кода.
3. Имеется быстрый доступ к Git.

Но, конечно, как и любая IDE, Atom имеет несколько минусов. Самый главный из них – производительность, а именно он очень сильно использует ЦП, и на старых компьютерах он может очень долго запускать и работать или не запускаться вообще.

* 1. **Выбор среды симулятора**

Существуют множество симуляторов для программирования роботов, например:

1. V-REP – симулятор роботов, основанный на архитектуре распределенного управления, а именно каждый объект или модель управляется индивидуально с помощью скрипта, узла ROS или удаленного клиента API. Он используется для разработки алгоритма, моделирования автоматизации производства и прототипов, а также проверки и обучения в области робототехники. На рисунке 5 представлен интерфейс V-REP

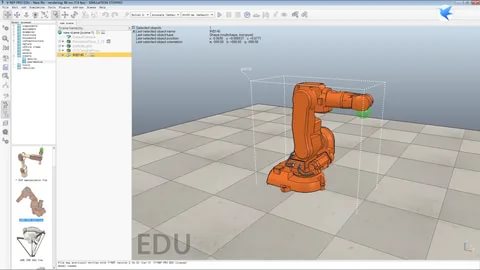


Рисунок V-Rep

1. Microsoft Robotics Developer Studio – windows-ориентированный симулятор роботов, предназначенный для разработки и поддержки аппаратного обеспечения роботов. На рисунке 6 представлен интерфейс Microsoft Robotics Developer Studio

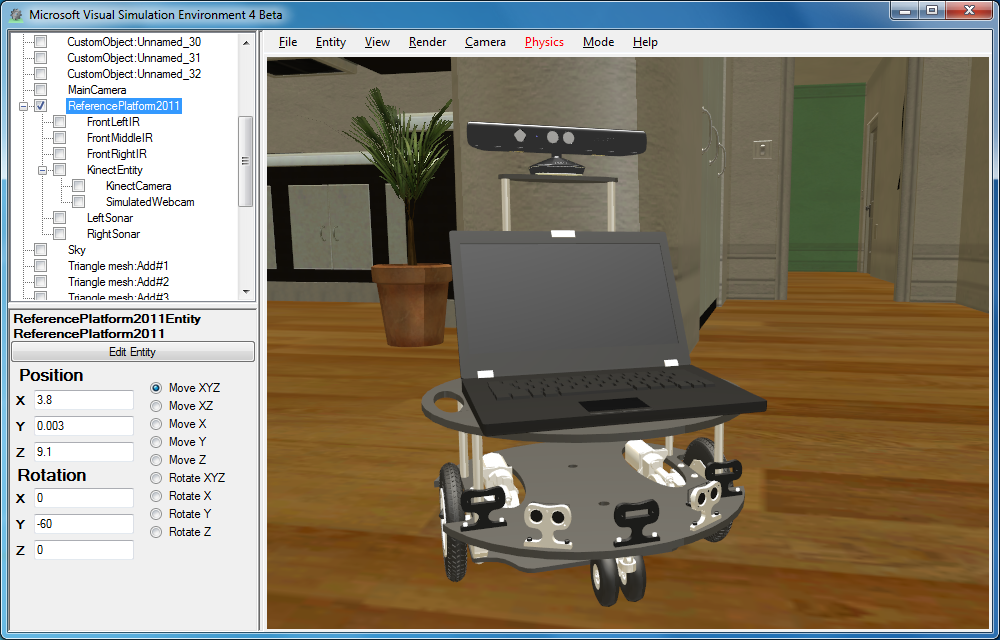


Рисунок 6 Microsoft Robotics Developer Studio

1. Gazebo – симулятор роботов. Он использует физический движок ODE, обеспечивающий визуализацию реальной окружающей среды при помощи высококачественных теней, текстур и освещения, визуализатор OpenGL и поддержка кода для сенсоров и приводов управления. На рисунке 7 представлен интерфейс Gazebo

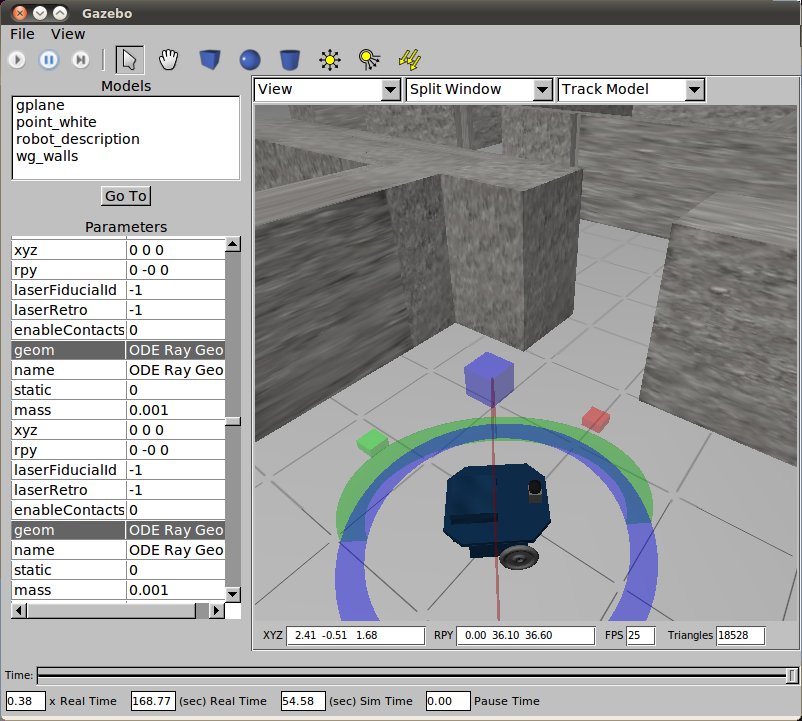


Рисунок Gazebo

1. Webots – симулятор роботов. Он включает большую коллекцию свободно изменяемых моделей роботов, датчиков, исполнительных механизмов и объектов. Кроме того, также можно создавать новые модели с нуля или импортировать их из программного обеспечения 3D CAD. При разработке модели робота пользователь указывает как графические, так и физические свойства объектов. Графические свойства включают форму, размеры, положение и ориентацию, цвета и текстуру объекта. К физическим свойствам относятся масса, коэффициент трения, а также постоянные пружины и демпфирования. В программе присутствует простая гидродинамика. Так же использует физический движок ODE. На рисунке 8 представлен интерфейс Webots

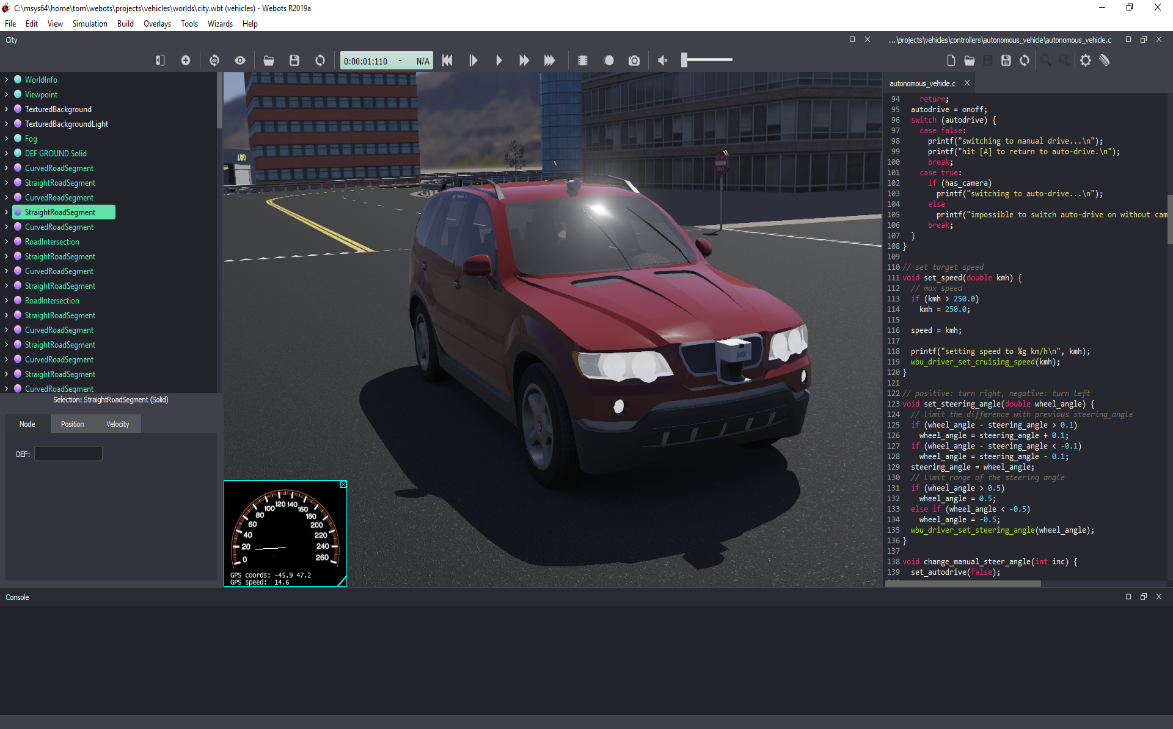


Рисунок Webots

В работе будет использоваться симулятор Webots. Он очень популярен и прост в усвоении.

* 1. **Обзор существующих методов обучения с подкреплением**

Обучение с подкреплением – способ машинного обучения, который является частным случаем обучения с учителем, где учителем является окружающая среда. Обучение с подкреплением применяется в тех случаях, когда нужно выбрать лучший вариант среди многих или достичь сложной цели за множество ходов. Алгоритмы подкрепления, которые включают в себя глубокое обучение, могут победить чемпионов мира в игре Go, начиная с базового понимания правил игры и тренируясь от партии к партии. Он состоит из двух компонентов: агент и окружающая среда. Процесс обучения состоит в следующих шагах:

1. Окружающая среда отправляет свое состояние агенту.
2. На основании отправленных состояний агент совершает действие.
3. Окружающая среда отправляет новое состояние агенту и награду за него.
4. Агент обновит свои знания на основе награды.
5. Повторяется шаг 2.

На данный момент существует множество методов обучения с подкреплением. Одни методы основаны на заданной модели, другие же на свободной (на рисунке 9 можно это посмотреть).

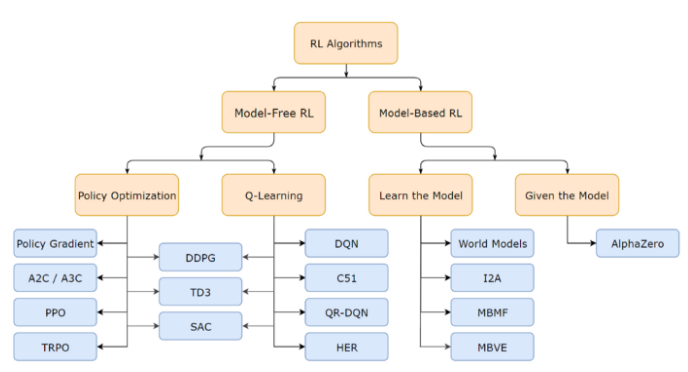


Рисунок 9. Обучение с подкреплением

Разберем некоторые из них:

1. Q-обучение – обучение с подкреплением на свободной модели, основанное на том, что каждому действию в определенном состоянии присваивается свое значения. Было введено в 1989 году, а также его сходимость было доказано Уоткинсом и Даяном в 1992 году. Он основан на алгоритме Бельмана и главная его цель достичь более близкого к истине значению.

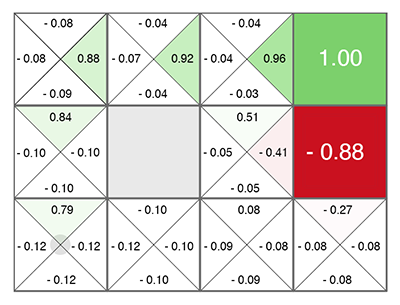


Рисунок 10. Q-обучение

На рисунке 10 представлен пример алгоритма Q-обучение. Цель агента в этом алгоритме максимизировать глобальное вознаграждение. Основываясь на максимальные вознаграждения следующих шагов, выбирает нужное вознаграждения на текущем состоянии. Это вознаграждение представляет собой взвешенную сумму ожидаемых значений вознаграждений за все будущие шаги. Q-обучение является итеративным алгоритмом. Хорошие начальные значения способствует исследованию. Также в процессе обучения можно использовать аппроксимацию функций, что способствует решению более крупных задач.

1. DQN – Q – обучение с использованием глубокой сверточной сети. Нейронная сеть делает обучение с подкреплением расходящимся. Это расходимость возникает из-за корреляции, присутствующей в последовательных наблюдениях. Но в самом алгоритме использовано воспроизведение опыта, который использует случайную выборку предыдущих действий вместо самого последнего действия для продолжения. Но в DQN на вход нейронная сеть подается текущий state (текущая ситуация), а на выходе нейронная сеть предсказывает число Q. А так как на выходе сети перечислены сразу все возможные действия (каждый со своим предсказанным Q), то получается что нейронная сеть в DQN реализует классическую функцию Q(s,a) из Q-learning. На рисунке 11 представлен алгоритм.

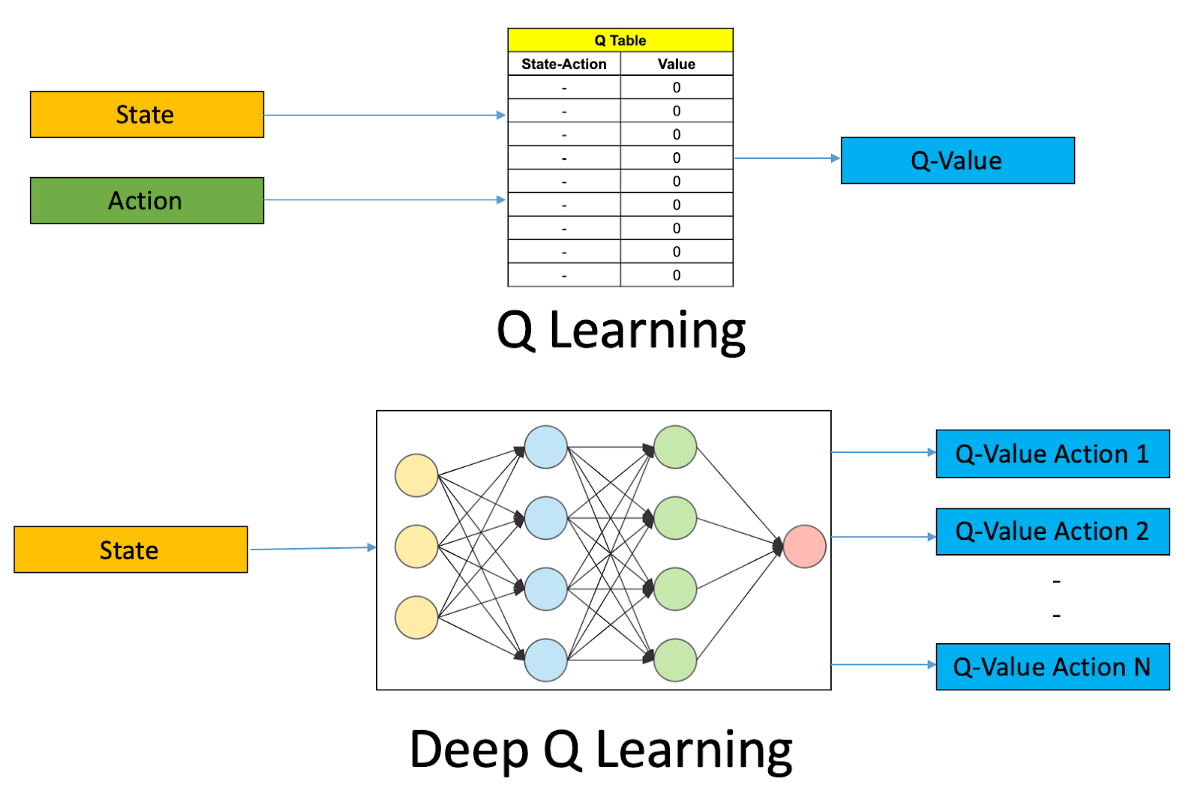


Рисунок 11. DQN

3) Policy-Gradient – метод обучения с подкреплением, предназначенный для нахождения оптимальной стратегии для агента получать оптимальное вознаграждение. Методы policy gradient нацелен на непосредственное моделирование или оптимизации политики. Обычно policy gradient моделируется с помощью параметрической функции θ от которой зависит , которая показывает вероятность определенного действия в определенное состояние. Это классический Policy Gradient. Но у него есть недостаток — надо ждать окончания эпизода, чтобы посчитать кумулятивную награду R, прежде чем изменять веса сети согласно ее изменению. А из преимуществ — гибкая система поощрений и наказаний, которая не только работает в обе стороны, но также зависит от величины награды. Большая награда сильнее поощряет действия, которые к ней привели. На рисунке 12 представлен алгоритм.

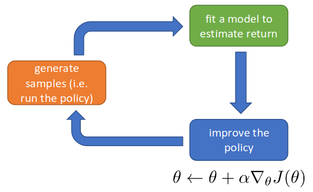


Рисунок 12. Policy Gradient

1. DDPG – обучение с подкреплением, которое совмещает алгоритмы Q-learning и Policy Gradient. Он подразумевает работу двух моделей, а именно Actor и Critic. Его плюс по сравнению с классическим Policy Gradient, что веса сети можно обновлять на каждом шаге, не дожидаясь окончания эпизода. Что ускоряет обучение. Так как она предсказывает напрямую действия actions, то прекрасно работает с continuous действиями. DDPG является прямым continuous конкурентом DQN с его дискретными действиями. На рисунке 13 можно увидеть алгоритм DDPG, где нижняя сеть выходит в роли Actor, а верхняя в роли Critic.

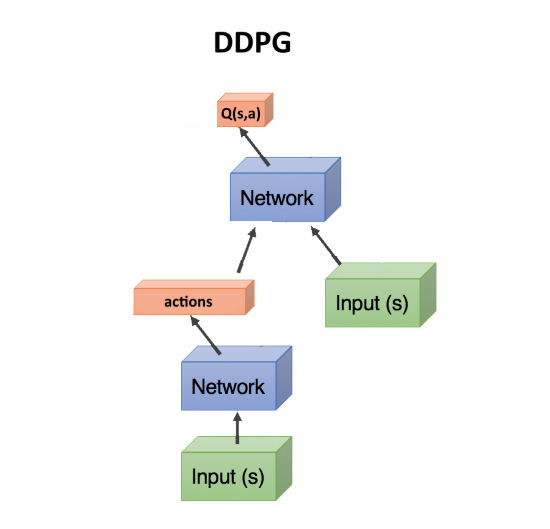


Рисунок 13. DDPG

1. A2C – обучение с подкреплением, при котором находят не только действие, но и значение действия V(s), которое отличается от Q(s,a) тем, что оно показывает лишь относительное преимущество в выборе действия и которое не зависит от действия, а зависит лишь от самого состояния. Это очень близко к тому, что происходит в биологическом мозге. Из экспериментов известно, что максимальный выброс дофамина происходит не во время самого получения удовольствия, а во время ожидания, что скоро получим удовольствие. Впрочем, если ожидания не оправдались, то наступают ужасные последствия, большие чем в обычном случае (в организме присутствует специальная система наказания, обратная системе вознаграждения). На рисунке 14 представлен алгоритм.

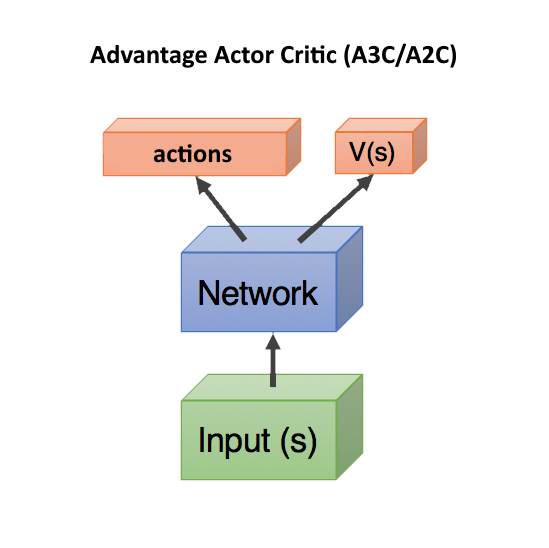


Рисунок 14. A2C

В данной задачи можно использовать метод Policy-Gradient, ведь для передвижения и расчетов нужно будет учитывать препятствия, которые не будут иметь четкого расположения, а будут разбросаны случайно.

1. **Разработка системы на структурном уровне**
   1. **Общая структурная схема**

Систему формирования карты помещения можно разделить на две части, а именно обучение робота и формирование карты. На рисунке 15 представлена общая структурная схема системы.

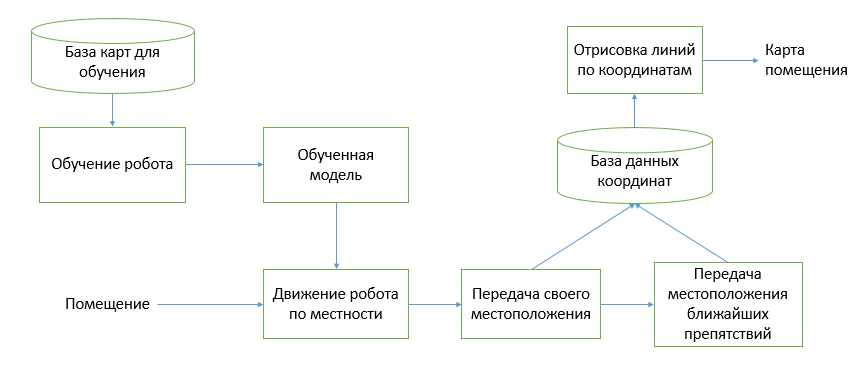


Рисунок 15. Структурная схема системы

На схеме представлены 2 модуля:

1. Работа робота. Она заключается в том, что робот будет работать на базе обучения с подкреплением и иметь карты для самого обучения. После робот, научившийся двигаться, будет отрисовывать карту помещения с помощь своих датчиков расстояния, GPS-навигатора и компаса.
2. Отрисовка карты. Она будет проводится по полученным данным с датчиков робота, а именно его положение, угол поворота, дальность и положение датчиков расстояния. С помощью математических функций поворота 2D тела, проекция робота будет поворачиваться также, как и оригинал.

На рисунках 16, 17, 18, 19 представлена DFD- диаграмма. Она состоит из главного блока, описывающего главную суть программы и изображенного на рисунке 16, блока ID = 1, описывающего блок обучения по модели и изображенного на рисунке 17, блока ID = 2, описывающего блок движения робота по помещению и изображенного на рисунке 18 и блока ID = 3, описывающего отрисовку карты и изображенного на рисунке 19. Также в главном блоке присутствуют такие действующие лица, как робот, supervisor, emitter, receiver, нейросеть, датчики, GPS – навигатор, компас и программа для отрисовки карт и 3 базы данных: база данных эпизодов, веса сети и база карт.



Рисунок 16. Главный блок DFD - диаграммы



Рисунок 17. Блок ID = 1 DFD-диаграммы



Рисунок 18. Блок ID = 2 DFD-диаграммы



Рисунок 19. Блок ID = 3 DFD-диаграммы

* 1. **Схема робота**

На рисунке 20 представлена схема робота

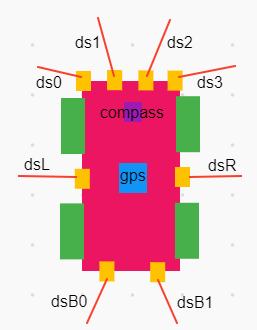


Рисунок 20. Робот

На нем можно увидеть 8 датчиков расстояния (4 передних, 2 задних, 1 левый, 1 правый), компас и GPS-навигатор. Датчики нужны для того, чтобы измерять расстояние до препятствия, компас для распознания угла поворота робота, а GPS-навигатор для передачи положения робота. Датчики настроены на длину до 30 см.

* 1. **Принципы работы робота**

В качестве алгоритма обучения с подкреплением взят Policy Gradient. Как и говорилось в заголовке 2.4 Policy Gradient был выбран по причине того, что он хорошо работает со случайным появлением препятствий, что в нашем случае обычное дело. Его работа представлена на рисунке 21, где:

1. Состояние среды, в котором находится агент на момент передачи этого состояния.
2. Действие агента, которое агент совершает на основе переданного состояния от окружающей среды на данный момент, учитывая вероятности всех действий.
3. Сама окружающая среда, то есть та модель, в которой агент пытается обучится.
4. Следующее состояние среды после предыдущего состояния с учетом сделанного агентом действия.
5. Алгоритм Монте-Карла, основанный на усреднении всех градиентов во время действий агента.

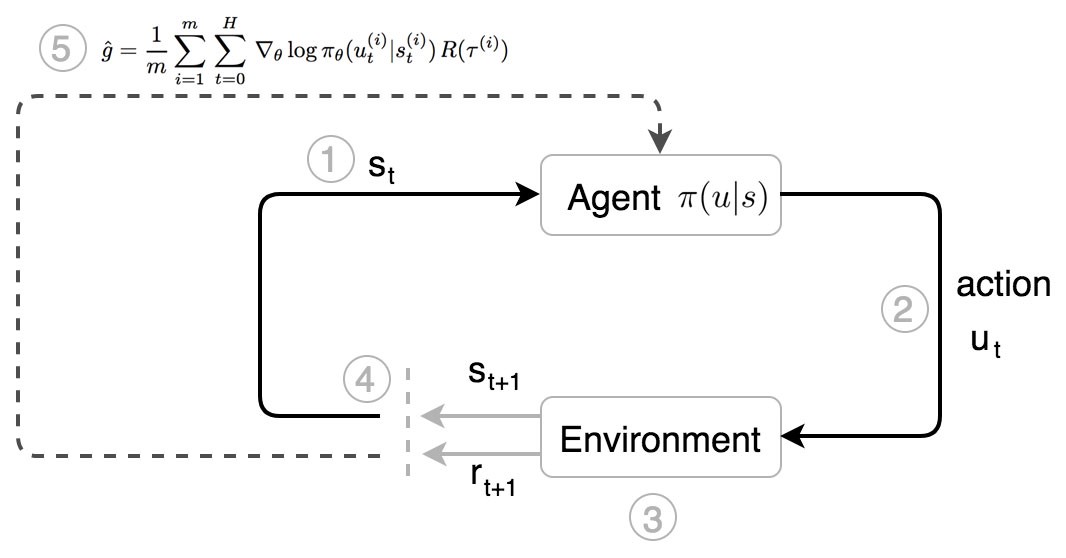


Рисунок 21. Policy Gradient

Этот алгоритм основан на вероятностной политике, где агент за определенное состояние получает не определенное действие, а вероятность действий. Так же подсчитывается градиент для данных по формуле (1) и изменяется θ по формуле (2).

(1)

 (2)

Для формирования карты берется 4 параметра:

1. Габариты робота (длина и ширина). В данном случае все рисуется относительно центра, поэтому берется лишь половина от длины и ширины. На 22 рисунке представлен пример, где r1 и r2 – половина длины и ширины соответственно. Транслируются в реальном времени.

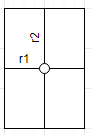


Рисунок 22. Габариты робота

1. Длина и состояние датчиков расстояния. В данном случае поддается массив значений, в котором содержатся показания соответствующих датчиков, а также подается максимальна длина датчика. Транслируются в реальном времени.
2. Показание GPS-навигатора. Поддаются координаты робота, транслируемые в реальном времени.
3. Показание компаса. Поддаются данные робота, транслируемые в реальном времени.

По показаниям GPS-навигатора и габаритам робота закрашивается область на карте, от GPS-навигатора берется положение робота и с помощью его габаритов от центра отрисовывается прямоугольник. Компас нужен для определения угла поворота робота, с помощью которого будет вращать каркас робота на карте, а длина и состояние датчиков для закрашивания области с препятствием или его отсутствием.

1. **Разработка программных средств**
   1. **Общая функциональная схема**

На рисунке 23 представлена функциональная схема, где Agent отвечает за обучение с подкреплением, MyCar за движение робота, SuperVis за вознаграждения и обновление симуляции и Display за зарисовку карты.

Класс Agent содержит такие методы, как:

1. build\_policy\_network() – функция модели сети. Функция возвращает модель нейронной сети. Модель нейронной сети состоит из 4 слоев: входной слой, 2 скрытых слоя и выходной слой. Также определена новая функция потерь, основанная на алгоритме Policy Gradient.
2. choose\_action(obs) – функция для смены действия агента. Эта функция принимает состояние среды и возвращает вероятности действий агента. Состояние поддается на вход обученной нейронной сети, после чего нейронная сеть выдает вероятности каждого из действий, которые выбираются на основе этих вероятностей.
3. store\_transition(obs, action, reward)– функция для накопления действий, состояний и наград. Принимает действия, состояния и награды и запоминает их для дальнейших вычислений
4. learn() – функция для обучения модели. Берутся все состояния, действия и награды. На основе наград вычисляется G, и после вычисления сети на обучение подают эти данные для изменения весов.
5. save\_model() – функция сохранения модели. Сохраняет веса нашей модели, так как архитектура не будет меняться, то её сохранять не обязательно.
6. load\_model() – функция загрузки модели. Загружает веса модели из файла.

Класс Display содержит такие методы, как:

1. drawEmptyCircle(x,z,r) – функция рисования на карте круглого каркаса робота. Функция принимает 2 координаты положения робота и радиус каркаса и возвращает ImageDraw объект.
2. drawEmptyRectangle(x,z,r1,r2,angle,multi) – функция рисования на карте прямоугольного каркаса робота. Функция принимает 2 координаты положения робота, половину ширины и длины каркаса, угол поворота и множитель увеличения карты.
3. angle(x,z,x1,z1) – функция, вычисляющая угол наклона точки относительно вертикальной оси. Она принимает 2 точки, отрезок, относительно которого будем находить наш угол.
4. drawSensorLine(x,z,mass,r1,r2,lenRay,angle,multi) – функция рисования датчиков расстояния, их поворота и перемещения. Она принимает 2 координаты положения робота, список всех датчиков, половину ширины и длины каркаса, дальность датчиков, угол поворота и множитель увеличения карты.
5. draw(x,z,mass,r1,r2,lenRay,angle,multi) – функция рисования прямоугольного робота на карте с датчиками расстояния. Она принимает 2 координаты положения робота, список всех датчиков, половину ширины и длины каркаса, дальность датчиков, угол поворота и множитель увеличения карты.
6. restart(i) – функция, которая сохраняет карту и удаляет её образ в программе, начиная рисовать новую карту.

Класс SuperVis содержит такие методы, как:

1. getStartPosAndRot() – функция, вычисляющая начальную позицию нашего робота. Она возвращает координаты начала движения робота, что помогает в вычислении наград.
2. getFinishPos() – функция, показывающая остановку роботу. Она возвращает координаты финиша, до которого робот должен доехать.
3. getPos() – функция, вычисляющая текущее положение робота. Она возвращает координата текущего положения робота для дальнейшей работы с этими данными.
4. getStartLen() – функция, вычисляющая расстояния от начала до конца пути. В дальнейшем используется для вычисления вознаграждения.
5. reward() – функция, вычисляющая награду, которую надо присвоить роботу. На основе текущей позиции робота и позиции его финиша, вычисляется текущая награда и отправляется роботу.
6. setStartPos(pos, rot) – функция, переносящая робота в заданное положение. Она принимает позицию и угол поворота, куда надо переместить робота и перемещает робота.
7. len(pos1, pos2) – функция, вычисляющая длину между двумя векторами pos1 и pos2.

Класс MyCar содержит такие методы, как:

1. startPos() – функция, запускающая все датчики расстояния, GPS-навигатор, моторы и компас, а также устанавливает позицию моторов робота и их скорость в значение равное 0.
2. finishPos() – функция, отключающая все датчики расстояния, GPS-навигатор, моторы и компас, а также устанавливает позицию моторов робота и их скорость в значение равное 0.
3. getSensorValues() – функция, собирающая информацию с датчиков расстояния. Она смотрит значение каждого включенного датчика, информация записывается в массив.
4. getGPSValues() - функция, собирающая информацию с GPS-навигатора. Берет значения навигатора для дальнейших вычислений.
5. chooseAction() – функция, выбирающая действие робота. Первым делом берет текущее состояние датчиков, оптимизирует их для нейронной сети. Эти показания передаются в нейронную сеть для выяснения вероятностей каждого действия и после на основе случайности делается выбор, учитывая вероятности.
6. getAngle() – функция, собирающая данные с компаса и высчитывающая угол поворота робота. Работает по показаниям компаса.
7. giveReward() – функция получения награды. Робот получает сообщение от среды и переводит это сообщение в число, так как он получает сообщение об награде.
8. takeAction() – функция, анализирующая столкновение робота. Это функция возвращает флаг, отвечающий за столкновения робота с препятствием. Этот флаг формируется с учетом показаний датчиков расстояния, которые робот получает в режиме реального времени.
9. speedCar(action) – функция, присваивающая скорость моторов робота. На вход функции подается предсказанное действие (одно из списка: лево, право, прямо) и за счет этого действия меняет скорости своего мотора для движения по предсказанным действиям.
10. train() – функция обучения робота. В начале очищаются все файлы, связанные с предыдущем обучением. После, если имеется файл с весами, то в нейронную сеть загружаются эти веса, а далее начинается обучение по эпизодам. На каждой итерации цикла эпизодов происходят следующие действия: робот включается, активирует все свои датчики, компас и GPS-навигатор, после передает показания датчиков передаются нейронной сети, которая предсказывает действие робота, после за это действие робот получает определенную награду и данные по награде, действию и состоянию передаются нейронной сети для дальнейшего обучения, далее начинает рисоваться карты и всё это происходит до столкновения с препятствием. Во время столкновения с препятствием робот останавливается, передает сигнал для сохранения полученной карты и обновления пространства для рисования и начинается обучение робота по данным, набранным за эпизод.
11. predict() – функция предсказания робота. Это функция получает от нейронной сети по текущему состоянию предсказанное действия и в зависимости от этого действия моторы робота приобретают нужные скорости для дальнейшего движения робота.

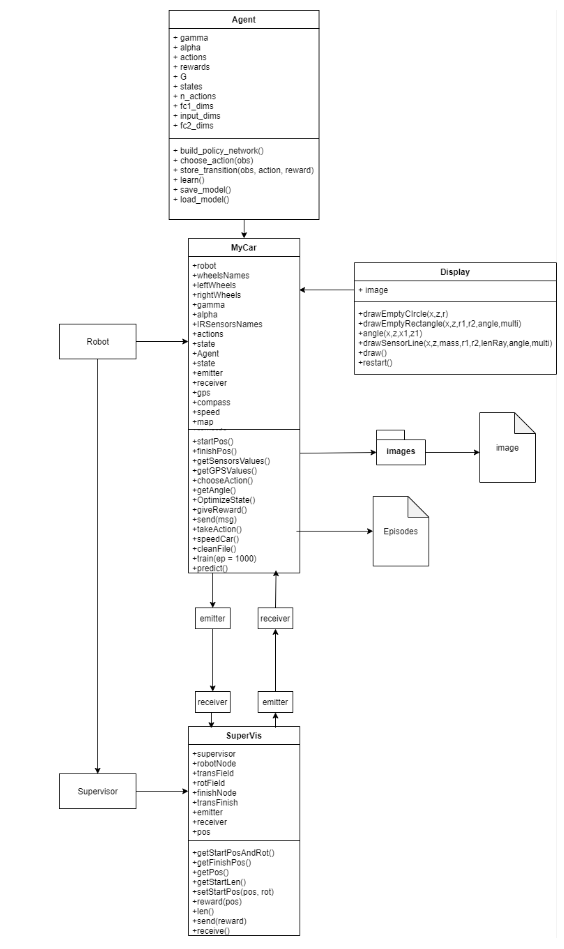
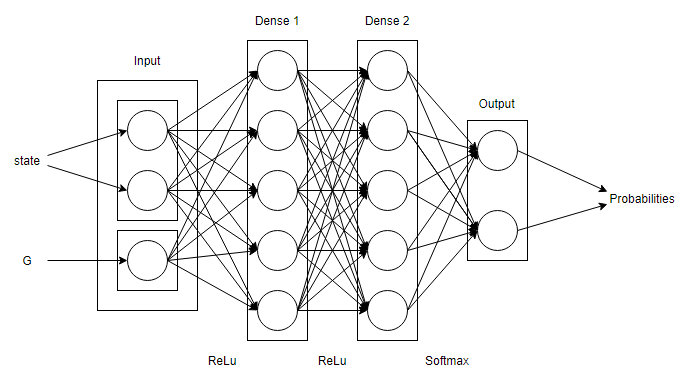


Рисунок 23. Функциональная схема

* 1. **Модуль нейронной сети для обучения с подкреплением**

Для начала нам нужна модель нейронной сети, веса которой будут изменяться по алгоритму Policy Gradient. На рисунке 24 можно увидеть модель нейронной сети. В качестве обучения нейронной сети была взята библиотека TensorFlow.

 Рисунок 24. Модель Нейронной сети

Реализация модели нейронной сети:

def build\_policy\_network(self):

input = Input(shape = (self.input\_dims,))

advantages = Input(shape = [1])

dense1 = Dense(self.fc1\_dims, activation = 'relu')(input)

dense2 = Dense(self.fc2\_dims, activation = 'relu')(dense1)

probs = Dense(self.n\_actions, activation = 'softmax')(dense2)

def custom\_loss(y\_true, y\_pred):

out = K.clip(y\_pred, 1e-8, 1 - 1e-8)

log\_lik = y\_true \* K.log(out)

return K.sum(-log\_lik \* advantages)

policy = Model([input, advantages], [probs])

policy.compile(optimizer = Adam(lr = self.alpha), loss = custom\_loss)

predict = Model([input], [probs])

return policy, predict

На модели видно 4 слоя (Слой входных данных (Input), 2 скрытых слоя (Dense 1, Dense 2) и Выходной слой (Output)). Для входного слоя используется 6 нейронов и функция активации ReLu. Для 2 скрытых слоев используется по 64 нейрона и функция активации ReLu. Для выходного слоя используется 3 нейрона и функция активации SoftMax.Так же использовалась своя функция потерь, которая основана на обучении с подкреплением, где все веса изменяются по формуле (3).

(1), где(3).

В качестве функции оптимизации был выбран Adam, который рассчитан на итерационном обновлении весов. Он объединяет два алгоритма, основанных на стохастическом градиенте, а именно AdaGrad (который обучается по параметру) и RMSProp (основанный на средних значениях последних величин градиентов для изменения весов).

Сама функция вознаграждения рассчитывается по формуле убывания полезность, то есть награда следующего шага в действительности. Это можно увидеть на формуле (4):

(4)

Реализация формулы (4):

G = np.zeros\_like(rewards)

for t in range(len(rewards)):

G\_sum = 0

discount = 1

for k in range(t, len(rewards)):

G\_sum += rewards[k] \* discount

discount \*= self.gamma

G[t] = G\_sum

mean = np.mean(G)

std = np.std(G) if np.std(G) > 0 else 1

self.G = (G - mean) / std

На рисунке 25 представлена блок-схема алгоритма:

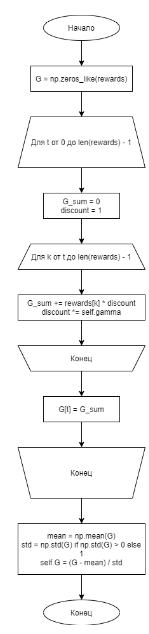


Рисунок Блок-схема G

Рассмотрим функцию активации ReLu. В отличии от сложных функций, которые в добавок являются нелинейными и могут приводить к проблемам с затуханием или увеличением градиентов, простая функция вроде ReLu, которая является выпрямленной линейной функции, в некоторых случаях будет вести себя гораздо лучше. Функция ReLu находится по формуле (5):

График этой функции изображен на рисунке 26.

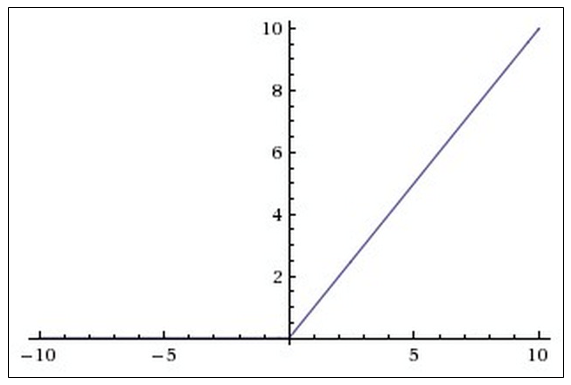


Рисунок 26. Функция ReLu

Как и любой алгоритм функция активации ReLu имеет ряд своих преимуществ и недостатков.

Преимущества:

1. Производная не зависит от параметров функции и равна либо 0, либо 1, что способствует нейтрализации разрастания или затуханию градиентов, ведь при сложных функциях активации градиент бы зависел от параметров самой функции, что может сильно повлиять на разрастание и затухание градиента.
2. Приводит к прореживанию весов.
3. В самой функции используются не ресурсоёмкие операции

Недостатки:

1. Крайне ненадежна, а именно можно изменить веса так, что некоторые нейроны не будут использоваться.
2. Не рассматривает отрицательный выход, что способствует занулению градиента.

Теперь рассмотрим функцию активации Softmax. Её особенность заключается в том, что она используется для обобщения логистической функции для многомерного случая, то есть она работает с вектором таким образом, что возвращает после пересчитывания вектор той же размерности с элементами, значения которых располагаются на интервале [0; 1] и их сумма равна единицы.

Функция активации softmax рассчитывается по формуле (6):

(6)

* 1. **Модуль робота**

В webots для работы контроллера первым делом нужно инициализировать все устройства робота, а именно моторы, датчики, компас и GPS-навигатор

Реализация инициализации устройств:

self.robot = Robot()

self.wheelNames = ['left wheel 1','left wheel 2','right wheel 1','right wheel 2']

self.leftWheels = [self.robot.getDevice(self.wheelNames[i]) for i in range(2)]

self.rightWheels = [self.robot.getDevice(self.wheelNames[i + 2]) for i in range(2)]

self.IRSensorsNames = ['dsL', 'ds0', 'ds1', 'ds2', 'ds3', 'dsR']

self.IRSensors = [self.robot.getDevice(sensor) for sensor in self.IRSensorsNames]

self.gps = self.robot.getDevice("gps")

self.compass = self.robot.getDevice('compass')

После инициализации все устройства необходимо включить. Это делает функция startPos(). Для некоторых устройств необходима указать время обновления их данных по мере движения. Реализация startPos():

self.gps.enable(self.timestep)

self.compass.enable(self.timestep)

for sensor in self.IRSensors:

sensor.enable(self.timestep)

for left, right in zip(self.leftWheels, self.rightWheels):

left.setPosition(float('inf'))

left.setVelocity(0.0)

right.setPosition(float('inf'))

right.setVelocity(0.0)

На рисунке 27 представлена блок-схема функции startPos:

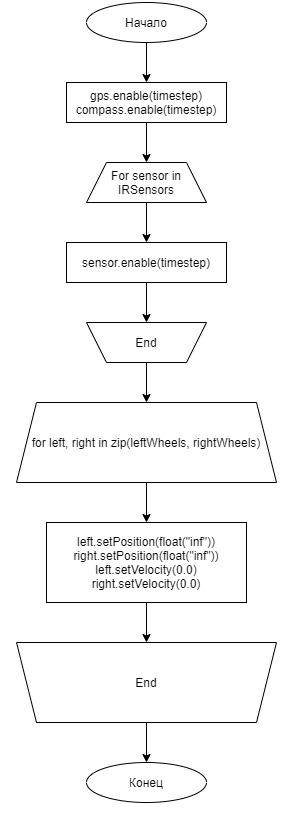


Рисунок Блок-схема startPos()

Также для остановки робота и отключения устройств используется функция finishPos(). Реализация finishPos:

self.gps.disable()

self.compass.disable()

for sensor in self.IRSensors:

sensor.disable()

for left, right in zip(self.leftWheels, self.rightWheels):

left.setPosition(float('inf'))

left.setVelocity(0.0)

right.setPosition(float('inf'))

right.setVelocity(0.0)

На рисунке 28 представлена блок-схема функции finishPos:

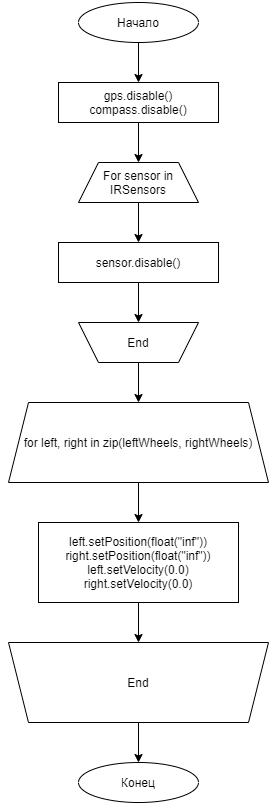


Рисунок Блок-схема G

Для обучения робота используется функция train(ep = 1000), которая на вход принимает количество эпизодов для обучения. Обучение происходит по такому алгоритму:

1. Запускается робот и все его устройства
2. Если есть готовые веса, то они загружаются
3. Робот получает показания GPS – навигатора
4. Робот от окружающей среды получает награду
5. По состоянию окружающей среды робот выбирает направление движения
6. Направление движения предсказывается нейронной сетью
7. По направлению движения роботу назначается скорость
8. Идет проверка встречи препятствия
9. Показания состояния среды, награды и направления движения передаются нейронной сети для обучения
10. Рисование текущего положения робота с его датчиками расстояния
11. Если робот встретил препятствие, то
    1. Обновляется полотно карты
    2. Нейронная сеть обучается по данным, переданным в шаге 8
    3. Обнуляется состояние среды
    4. Робот и все его устройства отключаются
    5. Сохраняется имеющиеся веса нейронной сети

Реализация алгоритма:

def train(self, ep = 100000):

self.cleanFile()

if os.path.exists('model\_weights.h5'):

self.Agent.load\_model()

for i in range(0, ep):

print("Epizode {}".format(i))

score = 0

while self.robot.step(self.timestep) != -1:

self.startPos()

gpsVal = self.getGPSValues()

reward = self.giveReward()

state, action = self.chooseAction()

self.speedCar(action)

self.state = self.takeAction()

if self.state == -1:

reward = -100

self.Agent.store\_transition(state, action, reward)

score += reward

self.map.draw(gpsVal[0], gpsVal[2], self.getSensorsValues(), 0.07, 0.1, 0.3, self.getAngle(), 100)

if self.state == -1:

self.rewards.append(score)

self.map.restart(i)

self.Agent.learn()

avgReward = np.mean(self.rewards[-100:])

s = "round {}: reward {}, avg reward {}".format(i, score, avgReward)

file = open("episodes.txt", 'a')

print(s)

s += '\n'

file.write(s)

file.close()

rewards = []

self.state = 0

self.finishPos()

self.Agent.save\_model()

self.send("1")

time.sleep(1)

break

Также у нашего робота имеется функция предсказания predict(), для работы с обученной сетью. Она работает по следующему алгоритму:

1. Нейронной сети подается показание датчиков расстояния
2. Нейронная сети по показаниям датчиков предсказывает действие робота
3. Робот движется по направлению, предсказанному нейронной сетью

Реализация алгоритма:

def predict(self):

action = self.chooseAction()

self.speedCar(action)

На рисунке 29 представлена блок-схема функции predict:

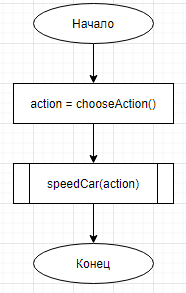


Рисунок Блок-схема predict

* 1. **Модуль формирования карты**

Карта формируется путем зарисовки самого каркаса робота, его положения по GPS-навигатору и показания датчиков расстояния. Модуль получает от робота показания угла поворота робота, посчитанного с помощью компаса.

Реализация подсчета угла поворота робота относительно показаний компаса.

def getAngle(self):

compassVal = self.compass.getValues()

rad = np.arctan2(compassVal[0], compassVal[2])

angle = rad - 1.5708

return angle

После получения показаний GPS-навигатора и угла поворота на карте вырисовывается позиция робота и его каркас. Угол поворота используется для разворота каркаса на определенный угол по принципу разворота 2D фигур вокруг своей оси. Это можно увидеть на формуле (7).

(7)

Реализация алгоритма по формуле (7):

def drawEmptyRectangle(self, x, z, r1, r2, angle, multi):

draw = ImageDraw.Draw(self.image)

x1 = x + (r1) \* np.cos(angle) - (r2) \* np.sin(angle)

y1 = z + (r2) \* np.cos(angle) + (r1) \* np.sin(angle)

x2 = x + (-r1) \* np.cos(angle) - (r2) \* np.sin(angle)

y2 = z + (r2) \* np.cos(angle) + (-r1) \* np.sin(angle)

x3 = x + (-r1) \* np.cos(angle) - (-r2) \* np.sin(angle)

y3 = z + (-r2) \* np.cos(angle) + (-r1) \* np.sin(angle)

x4 = x + (r1) \* np.cos(angle) - (-r2) \* np.sin(angle)

y4 = z + (-r2) \* np.cos(angle) + (r1) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

x3 = 250 + x3 \* multi

x4 = 250 + x4 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

y3 = 250 + y3 \* multi

y4 = 250 + y4 \* multi

draw.polygon(xy = ((x1,y1), (x2,y2), (x3,y3), (x4,y4)), fill = "brown", outline="brown")

return draw

На рисунке 30 представлена блок-схема функции drawEmptyRectangle

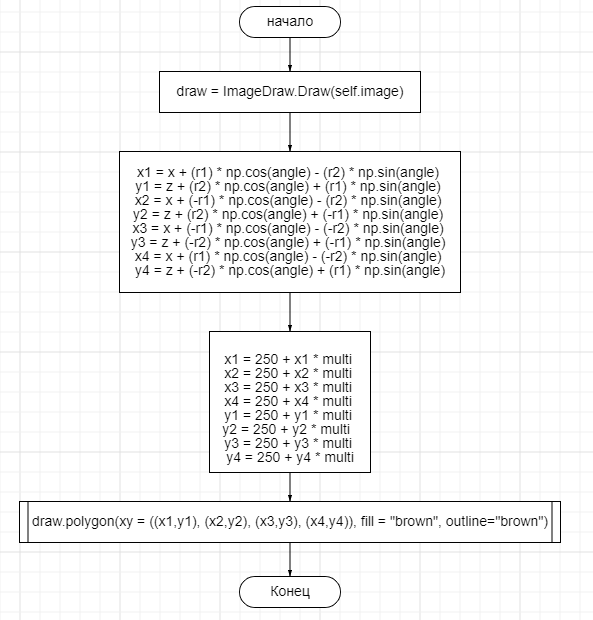


Рисунок Блок-схема drawEmptyRectangle()

На рисунке 31 изображен поворот прямоугольника относительно точки A( на угол , где B(0,0) – начала координат.

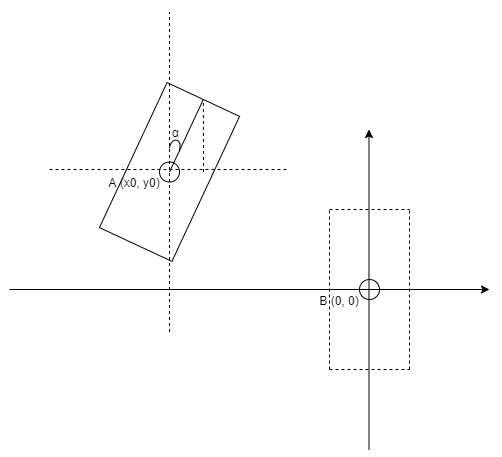


Рисунок 31. Поворот 2D фигуры относительно точки A(x0, y0)

В нашей зарисовке карты на данный момент используются лишь два датчика: левый и правый (dsL и dsR, представленные на рисунке 3). Датчики дают показания до 1000 единиц (где 1000 единиц – 30 см (предельное значение датчика)). Если датчик показывает все 1000 единиц, то можно утверждать, что в его пределе нет препятствий, если же датчик показывает меньше 1000, то значит он задетектировал препятствие, расстояния до которого можно просчитать по следующей формуле:

На рисунке 32 представлена блок-схема функции drawSensorLine:

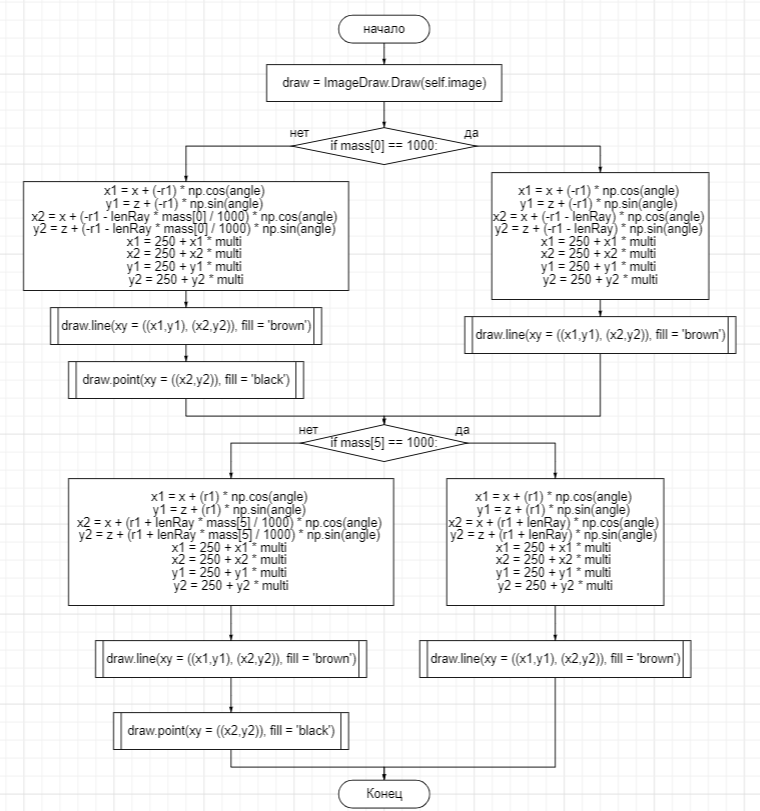


Рисунок Блок-схема drawSensorLine()

Реализация этого алгоритма:

def drawSensorLine(self, x, z, mass, r1, r2, lenRay, angle, multi):

draw = ImageDraw.Draw(self.image)

if mass[0] == 1000:

x1 = x + (-r1) \* np.cos(angle)

y1 = z + (-r1) \* np.sin(angle)

x2 = x + (-r1 - lenRay) \* np.cos(angle)

y2 = z + (-r1 - lenRay) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

else:

x1 = x + (-r1) \* np.cos(angle)

y1 = z + (-r1) \* np.sin(angle)

x2 = x + (-r1 - lenRay \* mass[0] / 1000) \* np.cos(angle)

y2 = z + (-r1 - lenRay \* mass[0] / 1000) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

draw.point(xy = ((x2,y2)), fill = 'black')

if mass[5] == 1000:

x1 = x + (r1) \* np.cos(angle)

y1 = z + (r1) \* np.sin(angle)

x2 = x + (r1 + lenRay) \* np.cos(angle)

y2 = z + (r1 + lenRay) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

else:

x1 = x + (r1) \* np.cos(angle)

y1 = z + (r1) \* np.sin(angle)

x2 = x + (r1 + lenRay \* mass[5] / 1000) \* np.cos(angle)

y2 = z + (r1 + lenRay \* mass[5] / 1000) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

draw.point(xy = ((x2,y2)), fill = 'black')

Функция draw(self, x, z, mass, r1, r2, lenRay, angle, multi) использует две верхние формулы, где:

1. x – координата робота по оси X на полотне.
2. z - координата робота по оси Y на полотне.
3. mass – массив датчиков расстояния
4. r1 – половина ширины робота
5. r2 – половина длины робота
6. lenRay – максимальное расстояние датчиков расстояния
7. angle – угол поворота робота
8. multi – увеличение изображения в несколько раз

Реализация алгоритма

def draw(self, x, z, mass, r1, r2, lenRay, angle, multi):

self.drawEmptyRectangle(x, z, r1, r2, angle, multi)

self.drawSensorLine(x, z, mass, r1, r2, lenRay, angle, multi)

На рисунке 33 изображена блок-схема функции draw:

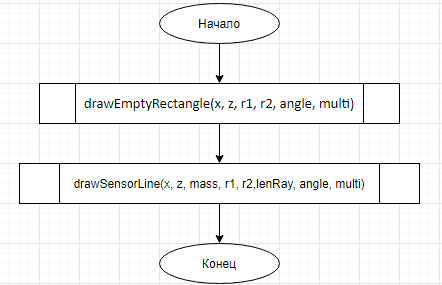


Рисунок Блок-схема draw()

Функция restart(self, i) принимает на вход число i, которое определяет имя файла для рисунка, который будет сохранен. Функция сохраняет изображение в файл test\_i.png и обновляет полотно, делая его абсолютно чистым.

Реализация алгоритма:

def restart(self, i):

self.image.save("images\\test\_{}.png".format(i), "PNG")

self.image = Image.new("RGB", (500,500), (155,155,155))

На рисунке 34 представлена блок-схема функции restart:

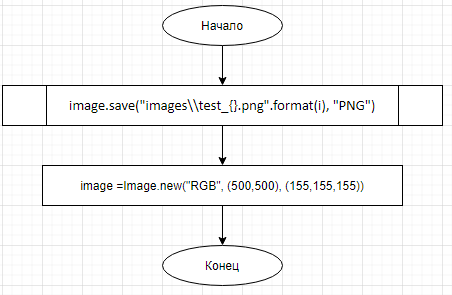


Рисунок Блок-схема restart()

Функция angle(x,z,x1,z1) принимает на вход 2 точки, начала и конца, и находит угол между вертикальной оси и отрезком, соединяющем 2 точки поддающиеся на вход. Для вычисления используется формула (8) угла между векторами

(8)

На рисунке 35 представлено нахождение угла между двумя векторами.

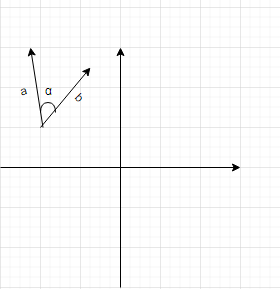


Рисунок Угол между двумя векторами

На рисунке 36 представлена блок-схема функции angle:

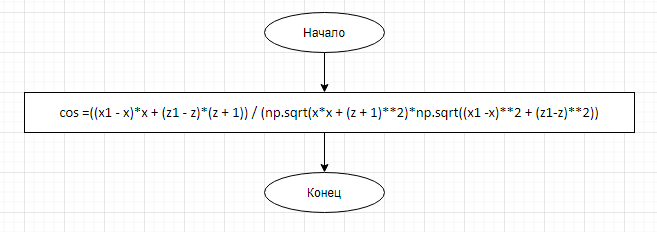


Рисунок Блок-схема angle()

Реализация функции angle():

def angle(self, x, z, x1, z1):

cos = ((x1 - x)\*x + (z1 - z)\*(z + 1)) / (np.sqrt(x\*x + (z + 1)\*\*2)\*np.sqrt((x1 - x)\*\*2 + (z1-z)\*\*2))

return np.arccos(cos)

1. **Тестирование системы**
   1. **Обучение системы**

На данный момент робот обучался около 3000-4000 эпох и показал результат, представленный на рисунке 37, а на рисунке 38 – оригинал.



Рисунок 37. Карта, нарисованная во время обучения робота

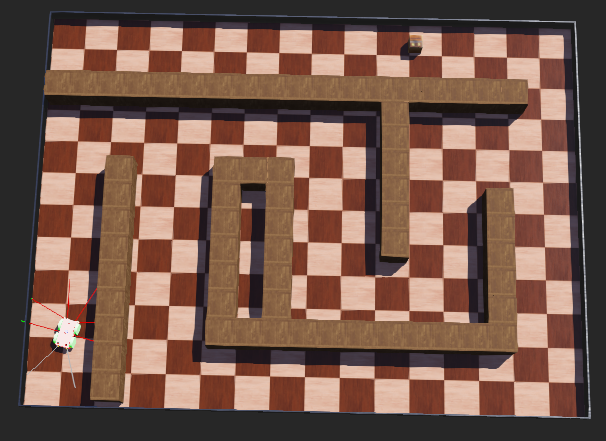


Рисунок 38. Реальная карта, на которой проходило обучение

Как видим, робот нарисовал около 90% карты, что есть неплохой результат для нейронной сети в целом.

* 1. **Тестирование обученной системы**

Обученная система была проверена на случайном помещении, представленном на рисунке 39. А на рисунке 40 изображено то, что робот нарисовал по мере движения.

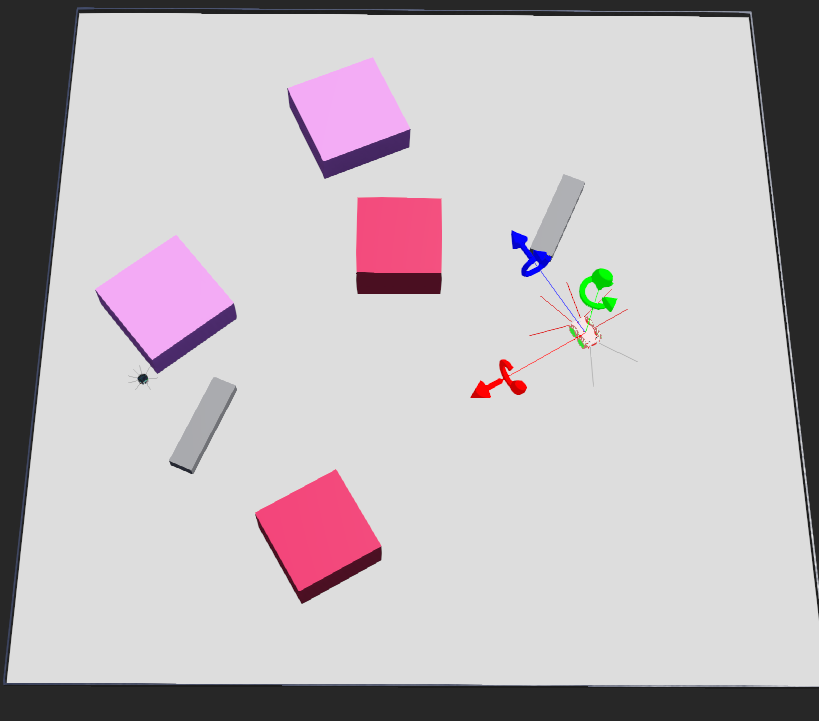


Рисунок 39. Тестируемое помещение

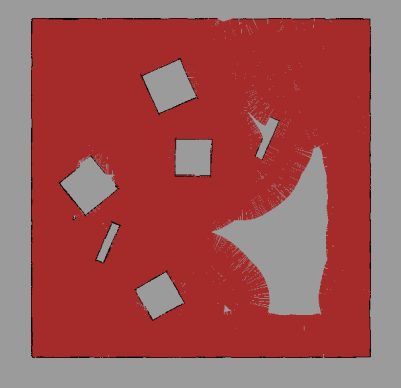


Рисунок 40. Карта тестируемого помещения

Как видим, робот нарисовал около 80-85%, что есть очень хороший результат для обучения.

**Заключение**

Было изучено обучение с подкреплением. После изучения были рассмотрены несколько известных алгоритмов обучения. Был выбран алгоритм Policy Gradient в качестве алгоритма для обучения робота рисовать карту помещения. Алгоритм был успешно реализован и результаты обучения составили около 80%, что на самом деле довольно неплохой вариант, но почему же всего лишь 80%? На этот вопрос можно ответить таким образом:

1. Первая причина – это использование нейронной сети, то есть в наше время сложно найти нейронную сеть с точностью в 100%,
2. Вторая причина - малое количество эпизодов для обучения. Этому способствовало время обучения и технические характеристики ПК, ведь нейронная сеть сама по себе трудоемкая и очень много берет ресурсов для себя.
3. Третья причина - использование не очень сложной архитектуры из-за все той же технической характеристики ПК.
4. Четвертая причина - использование простой функции активации ReLu, при которой некоторое нейроны могли быть не использованы из-за зануления градиента при изменении весов.

**Список литературы**

1. Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W.(1996). "Reinforcement Learning: A Survey”
2. van Otterlo, M.; Wiering, M. (2012). Reinforcement learning and markov decision processes. Reinforcement Learning. Adaptation, Learning, and Optimization. 12. pp. 3–42. doi:10.1007/978-3-642-27645-3\_1. ISBN 978-3-642-27644-6.
3. Gosavi, Abhijit (2003). Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement. Operations Research/Computer Science Interfaces Series. Springer. ISBN 978-1-4020-7454-7.
4. Burnetas, Apostolos N.; Katehakis, Michael N. (1997), "Optimal adaptive policies for Markov Decision Processes", Mathematics of Operations Research, 22: 222–255, doi:10.1287/moor.22.1.222
5. Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992): 229-256.
6. Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems. 2000.
7. Policy Gradients. CS 294-112: Deep Reinforcement Learning, Sergey Levine.

**Приложение А**

Код проекта состоит из 3 моделей: reinforce\_keras.py, map.py, testCar.py, а именно:

1. reinforce\_keras.py:

import tensorflow as tf

from tensorflow.keras.layers import Dense, Activation, Input

from tensorflow.keras.models import Model, load\_model

from tensorflow.keras.optimizers import Adam

import tensorflow.keras.backend as K

from tensorflow.python.framework.ops import disable\_eager\_execution

import numpy as np

disable\_eager\_execution()

class Agent(object):

def \_\_init\_\_(self, alpha, gamma, n\_actions = 4, layer1\_size = 16, layer2\_size = 16, input\_dims = 128, fname = 'model\_weights.h5'):

self.gamma = gamma

self.alpha = alpha

self.G = 0

self.input\_dims = input\_dims

self.fc1\_dims = layer1\_size

self.fc2\_dims = layer2\_size

self.n\_actions = n\_actions

self.states = []

self.actions = []

self.rewards = []

self.policy, self.predict = self.build\_policy\_network()

self.action\_space = [i for i in range(n\_actions)]

self.model\_file = fname

def build\_policy\_network(self):

input = Input(shape = (self.input\_dims,))

advantages = Input(shape = [1])

dense1 = Dense(self.fc1\_dims, activation = 'relu')(input)

dense2 = Dense(self.fc2\_dims, activation = 'relu')(dense1)

probs = Dense(self.n\_actions, activation = 'softmax')(dense2)

def custom\_loss(y\_true, y\_pred):

out = K.clip(y\_pred, 1e-8, 1 - 1e-8)

log\_lik = y\_true \* K.log(out)

return K.sum(-log\_lik \* advantages)

policy = Model([input, advantages], [probs])

policy.compile(optimizer = Adam(lr = self.alpha), loss = custom\_loss)

predict = Model([input], [probs])

return policy, predict

def choose\_action(self, obs):

state = np.array([obs])

# print("state = ", state)

probabilities = self.predict.predict(state)[0]

# print("prob = ", probabilities)

action = np.random.choice(self.action\_space, p = probabilities)

# print("action = ", action)

return action

def store\_transition(self, obs, action, reward):

self.states.append(obs)

self.actions.append(action)

self.rewards.append(reward)

def learn(self):

actions = np.array(self.actions)

states = np.array(self.states)

rewards = np.array(self.rewards)

actions1 = np.zeros([len(actions), self.n\_actions])

actions1[np.arange(len(actions)), actions] = 1

G = np.zeros\_like(rewards)

for t in range(len(rewards)):

G\_sum = 0

discount = 1

for k in range(t, len(rewards)):

G\_sum += rewards[k] \* discount

discount \*= self.gamma

G[t] = G\_sum

mean = np.mean(G)

std = np.std(G) if np.std(G) > 0 else 1

self.G = (G - mean) / std

cost = self.policy.train\_on\_batch([states, self.G], actions1)

self.states = []

self.actions = []

self.rewards = []

def save\_model(self):

self.policy.save\_weights(self.model\_file)

def load\_model(self):

self.policy.load\_weights(self.model\_file)

1. map.py:

from PIL import Image, ImageDraw

import numpy as np

class Display():

def \_\_init\_\_(self):

self.image = Image.new("RGB", (500,500), (155,155,155))

def drawEmptyCircle(self, x, z, r):

draw = ImageDraw.Draw(self.image)

draw.ellipse((250 + x - r, 250 - z - r, 250 + x + r, 250 - z + r), fill = "brown", outline="brown")

return draw

def drawEmptyRectangle(self, x, z, r1, r2, angle, multi):

draw = ImageDraw.Draw(self.image)

x1 = x + (r1) \* np.cos(angle) - (r2) \* np.sin(angle)

y1 = z + (r2) \* np.cos(angle) + (r1) \* np.sin(angle)

x2 = x + (-r1) \* np.cos(angle) - (r2) \* np.sin(angle)

y2 = z + (r2) \* np.cos(angle) + (-r1) \* np.sin(angle)

x3 = x + (-r1) \* np.cos(angle) - (-r2) \* np.sin(angle)

y3 = z + (-r2) \* np.cos(angle) + (-r1) \* np.sin(angle)

x4 = x + (r1) \* np.cos(angle) - (-r2) \* np.sin(angle)

y4 = z + (-r2) \* np.cos(angle) + (r1) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

x3 = 250 + x3 \* multi

x4 = 250 + x4 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

y3 = 250 + y3 \* multi

y4 = 250 + y4 \* multi

draw.polygon(xy = ((x1,y1), (x2,y2), (x3,y3), (x4,y4)), fill = "brown", outline="brown")

return draw

def angle(self, x, z, x1, z1):

cos = ((x1 - x)\*x + (z1 - z)\*(z + 1)) / (np.sqrt(x\*x + (z + 1)\*\*2)\*np.sqrt((x1 - x)\*\*2 + (z1-z)\*\*2))

return np.arccos(cos)

def drawSensorLine(self, x, z, mass, r1, r2, lenRay, angle, multi):

draw = ImageDraw.Draw(self.image)

if mass[0] == 1000:

x1 = x + (-r1) \* np.cos(angle)

y1 = z + (-r1) \* np.sin(angle)

x2 = x + (-r1 - lenRay) \* np.cos(angle)

y2 = z + (-r1 - lenRay) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

else:

x1 = x + (-r1) \* np.cos(angle)

y1 = z + (-r1) \* np.sin(angle)

x2 = x + (-r1 - lenRay \* mass[0] / 1000) \* np.cos(angle)

y2 = z + (-r1 - lenRay \* mass[0] / 1000) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

draw.point(xy = ((x2,y2)), fill = 'black')

if mass[5] == 1000:

x1 = x + (r1) \* np.cos(angle)

y1 = z + (r1) \* np.sin(angle)

x2 = x + (r1 + lenRay) \* np.cos(angle)

y2 = z + (r1 + lenRay) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

else:

x1 = x + (r1) \* np.cos(angle)

y1 = z + (r1) \* np.sin(angle)

x2 = x + (r1 + lenRay \* mass[5] / 1000) \* np.cos(angle)

y2 = z + (r1 + lenRay \* mass[5] / 1000) \* np.sin(angle)

x1 = 250 + x1 \* multi

x2 = 250 + x2 \* multi

y1 = 250 + y1 \* multi

y2 = 250 + y2 \* multi

draw.line(xy = ((x1,y1), (x2,y2)), fill = 'brown')

draw.point(xy = ((x2,y2)), fill = 'black')

def draw(self, x, z, mass, r1, r2, lenRay, angle, multi):

self.drawEmptyRectangle(x, z, r1, r2, angle, multi)

self.drawSensorLine(x, z, mass, r1, r2, lenRay, angle, multi)

def restart(self, i):

self.image.save("images\\test\_{}.png".format(i), "PNG")

self.image = Image.new("RGB", (500,500), (155,155,155))

1. testCar.py:

"""testCar controller."""

from controller import Robot

import numpy as np

import map

import os

import shutil

from reinforce\_keras import Agent

import time

class MyCar():

def \_\_init\_\_(self, timestep, speed, alpha = 0.0002, gamma = 0.95):

self.robot = Robot()

self.wheelNames = ['left wheel 1','left wheel 2','right wheel 1','right wheel 2']

self.leftWheels = [self.robot.getDevice(self.wheelNames[i]) for i in range(2)]

self.rightWheels = [self.robot.getDevice(self.wheelNames[i + 2]) for i in range(2)]

self.timestep = timestep

self.alpha = alpha

self.gamma = gamma

self.IRSensorsNames = ['dsL', 'ds0', 'ds1', 'ds2', 'ds3', 'dsR']

self.actions = ['left', 'front', 'right']

self.IRSensors = [self.robot.getDevice(sensor) for sensor in self.IRSensorsNames]

self.Agent = Agent(alpha = alpha, input\_dims = 6, gamma = gamma, n\_actions = 3, layer1\_size = 64, layer2\_size = 64)

self.state = 0

self.emitter = self.robot.getDevice("emitter")

self.receiver = self.robot.getDevice("receiver")

self.gps = self.robot.getDevice("gps")

self.compass = self.robot.getDevice('compass')

self.speed = speed

self.map = map.Display()

self.rewards = []

def startPos(self):

self.receiver.enable(self.timestep)

self.gps.enable(self.timestep)

self.compass.enable(self.timestep)

for sensor in self.IRSensors:

sensor.enable(self.timestep)

for left, right in zip(self.leftWheels, self.rightWheels):

left.setPosition(float('inf'))

left.setVelocity(0.0)

right.setPosition(float('inf'))

right.setVelocity(0.0)

def finishPos(self):

self.receiver.disable()

self.gps.disable()

self.compass.disable()

for sensor in self.IRSensors:

sensor.disable()

for left, right in zip(self.leftWheels, self.rightWheels):

left.setPosition(float('inf'))

left.setVelocity(0.0)

right.setPosition(float('inf'))

right.setVelocity(0.0)

def getSensorsValues(self):

IRSensorValues = []

# print(self.gps.getValues())

for sensor in self.IRSensors:

IRSensorValues.append(sensor.getValue())

return IRSensorValues

def getGPSValues(self):

gpsVal = self.gps.getValues()

return gpsVal

def getAngle(self):

compassVal = self.compass.getValues()

rad = np.arctan2(compassVal[0], compassVal[2])

angle = rad - 1.5708

return angle

def optimizeState(self, vector):

max = 800

max2 = 600

for i, vec in enumerate(vector):

if i == 2 or i == 3:

vector[i] = 1 if vec > max else 0

else:

vector[i] = 1 if vec > max2 else 0

return vector

def chooseAction(self):

state = self.getSensorsValues()

state = self.optimizeState(state)

action = self.Agent.choose\_action(state)

# print("sens :", sens)

return state, action

def giveReward(self):

# print("len = ", self.receiver.getQueueLength())

if self.receiver.getQueueLength() > 0:

msg = self.receiver.getData().decode("utf-8")

reward = int(msg)

self.receiver.nextPacket()

self.receiver.nextPacket()

return reward

def send(self, msg):

msg1 = msg.encode("utf-8")

self.emitter.send(msg1)

def takeAction(self):

sensors = self.getSensorsValues()

# print(sensors)

# print("sensors: ", sensors)

if sensors[1] < 70 or sensors[2] < 150 or sensors[3] < 150 or sensors[4] < 70:

state = -1

else:

state = 0

return state

def speedCar(self, action):

if action == 0:

leftSpeed = 0

rightSpeed = self.speed

elif action == 2:

leftSpeed = self.speed

rightSpeed = 0

else:

leftSpeed = self.speed

rightSpeed = self.speed

for left, right in zip(self.leftWheels, self.rightWheels):

left.setVelocity(leftSpeed)

right.setVelocity(rightSpeed)

def cleanFile(self):

file = open("episodes.txt", 'w')

file.close()

file = open("thetas.txt", 'w')

file.close()

shutil.rmtree("images")

os.mkdir("images")

def train(self, ep = 100000):

self.cleanFile()

if os.path.exists('model\_weights.h5'):

self.Agent.load\_model()

for i in range(0, ep):

print("Epizode {}".format(i))

self.startPos()

score = 0

while self.robot.step(self.timestep) != -1:

self.startPos()

gpsVal = self.getGPSValues()

reward = self.giveReward()

state, action = self.chooseAction()

self.speedCar(action)

self.state = self.takeAction()

if self.state == -1:

reward = -100

self.Agent.store\_transition(state, action, reward)

score += reward

self.map.draw(gpsVal[0], gpsVal[2], self.getSensorsValues(), 0.07, 0.1, 0.3, self.getAngle(), 100)

# print(reward)

if self.state == -1:

self.rewards.append(score)

self.map.restart(i)

self.Agent.learn()

avgReward = np.mean(self.rewards[-100:])

s = "round {}: reward {}, avg reward {}".format(i, score, avgReward)

file = open("episodes.txt", 'a')

print(s)

s += '\n'

file.write(s)

file.close()

rewards = []

self.state = 0

self.finishPos()

self.Agent.save\_model()

self.send("1")

time.sleep(1)

break

def predict(self):

action = self.chooseAction()

self.speedCar(action)

robot = MyCar(64, 2)

robot.train(1000)