Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки: 01.04.02 Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А.А. Замышляева
«____»_____2019 г.

Алгоритм обучения с подкреплением для управления движением робота

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.04.02.2019.509.ПЗ ВКР

Руководитель работы, к.ф.-м.н.,
доцент кафедры ПМиП
_____/С.М. Елсаков
«____»_____2019 г.
Автор работы
студент группы ЕТ-222
_____/М.П. Майоров
«____»_____2019 г.
Нормоконтролер, ассистент
_____/Н.С. Мидоночева
«____»_____2019 г.

Челябинск
2019

АННОТАЦИЯ

Майоров М.П. Алгоритм обучения с подкреплением для решения задачи движения робота. – Челябинск: ЮУрГУ, ЕТ-222, 52 с., 28 ил., библиогр. список – 23 наим., 1 прил.

Целью данной работы является разработка алгоритма обучения с подкреплением для решения задачи движения робота с реализацией в виде компьютерной программы.

В первом разделе был проведен анализ предметной области, исследован объект управления. Проведен обзор существующих методов управления роботом, рассмотрены основные конфигурации, используемые в робототехнике для решения задачи. Рассмотрены виды обучения с подкреплением. Проанализирован алгоритм обучения с подкреплением, основанный на методе обучения агента Q-learning.

Во втором разделе описаны модель объекта управления, модель виртуальной среды, модель процесса обучения интеллектуального агента. Описывается функция вознаграждения агента, используемая при обучении с подкреплением.

Третий раздел посвящен разработке и тестированию алгоритма обучения с подкреплением. Приведена информация об особенностях реализации среды, агента, процесса взаимодействия между ними и обучения. Также в данном разделе представлены результаты и сравнение алгоритма обучения с подкреплением с тривиальным алгоритмом движения вдоль линии.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Окружающий нас мир очень сложная открытая система, развивающаяся и меняющаяся во времени и пространстве. Автоматизация и роботизация встречается на каждом шагу. Автоматические устройства выполняют рутинные, сложные, рискованные и другие задачи, подстраиваясь и адаптируясь к существующему миру для достижения наилучших результатов. Адаптивность информационной системы – это ее способность изменяться для сохранения своих эксплуатационных показателей в заданных пределах при изменениях внешней среды, также – это одно из качеств, которые уменьшают разрыв между понятием информационной системой и понятием живого организма. Адаптивность информационной системы сегодня – это залог успеха системы, завтра это – необходимое условие выживаемости. В информационной среде уже проходит отбор систем по критерию адаптивности, какая система быстрее сможет приспособиться к новым условиям существования. Современные способы построения адаптивных информационных систем в совокупности с системным подходом позволяют создать информационную систему, удовлетворяющую критерию адаптивности.

Робототехника – прикладная наука, занимающаяся разработкой автоматизированных технических систем. Робот действует по заранее заложенной программе, получая информацию о внешнем мире от датчиков, он самостоятельно осуществляет производственные и иные операции, обычно выполняемые человеком (либо животными). При этом робот может, как иметь связь с оператором (получать от него команды), так и действовать автономно. Робототехника опирается на такие дисциплины, как электроника, механика, информатика, радиотехника, электротехника и другие. Выделяют основные виды робототехники: строительная, промышленная, бытовая, авиационная, военная, космическая и подводная. Практически в каждом виде

роботов выделяют задачу движения, которая является одной из главных подзадач, требуемых для реализации проекта.

Для решения задач на стыке машинного обучения и оптимального управления была создана область обучения с подкреплением. В системах обучения с подкреплением выделяют важную проблему выбора между эксплуатацией текущих знаний для получения наибольшей награды и исследованием окружения с целью получения знаний о структуре окружения. В данной работе ставилась задача получения стратегия исследования окружений с непрерывными пространствами состояний. В качестве основы системы обучения с подкреплением была выбрана модель глубокого - обучения. Была предложена модификации существующей стратегии исследования в рамках задачи движения робота.

Существует множество готовых конфигураций роботов, механизмов и исполнителей, в основе которых лежат различные датчики, позволяющие осуществлять навигацию и передвижение в пространстве. А также существуют конструкторы, которые позволяют создавать макеты и прототипы роботов, используя практически любые конфигурации и формы исполнителей.

Цель работы: разработать и реализовать на языке высокого уровня алгоритм обучения с подкреплением для решения задачи движения робота по заданной линии.

Таким образом, для достижения цели были поставлены следующие задачи.

1. Провести анализ предметной области движения робота.

2. Изучить алгоритмы обучения с подкреплением.

3. Выполнить программную реализацию алгоритма обучения с подкреплением для решения задачи движения робота по линии.

4. Провести тестирование работы алгоритма в виртуальной среде.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Исследование объекта управления

Робототехника уже сейчас стала частью нашей жизни. С ростом уровня автоматизации увеличивается разнообразие механизмов, облегчающих труд.

Автомат – это устройство, которое может работать по заложенной в него программе без участия человека [1].

Робот – это автомат, который способен ориентироваться в окружающей среде и обладает элементами искусственного интеллекта, то есть он может принимать «самостоятельные решения», например управлять автомобилем вместо водителя [1].

Существуют различные модели готовых решений, отличающиеся ценой, предустановленным программным обеспечением, запасом хода и другими параметрами. Также на рынке присутствуют конструкторы, из которых можно собрать прототип решения, гибко изменять параметры натурной модели, например расположение и количество индикаторов. Основные Arduino [2], Raspberry pi [3], intel nuc [4], LEGO Mindstorms [5] некоторые примеры на базе этих платформ представлены на рисунке 1.1. Конструкторы хорошо подходят для проверки идей и новых решений, а также создания прототипов.

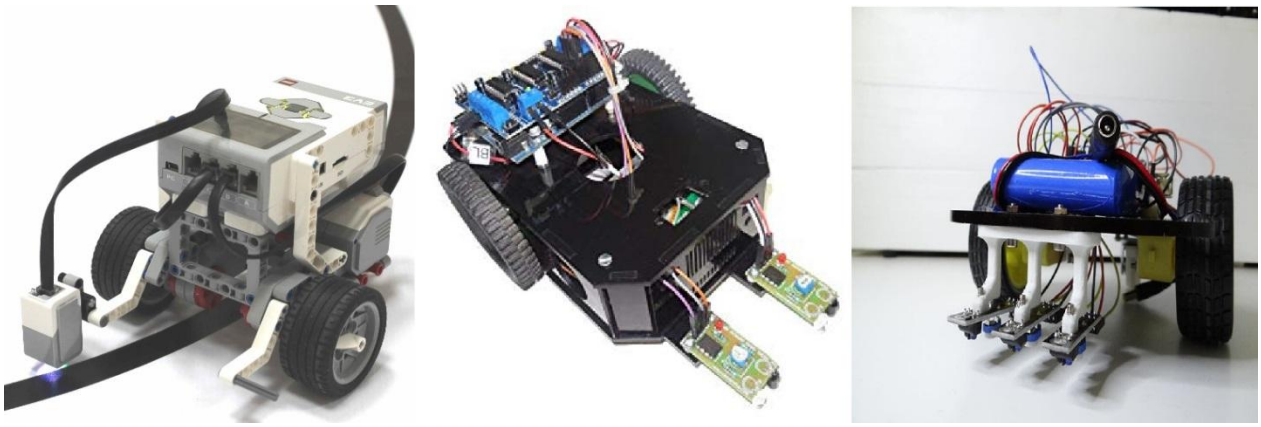


Рисунок 1.1 – Примеры роботов

Прогресс локальной навигации роботов во многом будет определяться развитием систем машинного зрения. Но пока не появятся высокоэффективные решения, сохранится интерес к технологиям локальной навигации по сигналам внешних маяков. Эти маяки обретут интеллектуальные свойства и состыкуются с GPS-службой (первые эксперименты показали высокую результативность такого подхода) [6]. Хорошо известные на рынке роботы и относительно дешевые наборы для их создания, предлагаемые фирмой Evolution Robotics, основываются на коммерчески доступных одометрах и системах машинного зрения [7].

Решения, использующие более точные лазерные дальномеры, пока дороги (около 5 тыс. долл.) [8]. Технологии машинного зрения (включая «зрение» с помощью сонаров и лазеров), обработки сигналов глобальных навигационных систем типа GPS и ориентации по маякам считаются сегодня ключевыми в задачах навигации роботов.

Значительно более перспективно формирование точной геометрической модели окружающего пространства и распознавания. Но для этого надо иметь максимально детальную информацию об окружающей среде, а предоставить ее могут устройства визуального наблюдения (машинного зрения) высокого разрешения и хорошие системы распознавания объектов.

Такие системы пока либо очень дороги, либо существуют только в экспериментальных версиях, а базирующиеся на них роботы уверенно двигаются только в простой обстановке.

Во многих случаях карта может быть подготовлена заранее, но тогда возникает проблема определения собственного местоположения на ней, что весьма сложно в случаях, когда отсутствуют различимые особенности местности. Вариант, когда робот самостоятельно строит карту, а затем, сопоставляя ее с исходной фиксированной версией, уточняет и корректирует. Но задача такого сопоставления для компьютера также очень трудоемка. Ведь трехмерные цифровые карты местности требуют для хранения во много

раз больше памяти, нежели двумерные. Соответственно и для обработки таких объемов данных нужны приличные вычислительные ресурсы.

В дополнение к задаче определения координат устройству необходимо уметь строить план движения по такой карте. Существующие алгоритмы позволяют делать это достаточно точно, однако они пока не учитывают скорость движения аппарата по маршруту – а она может внести серьезные коррективы в теоретический план.

Тем не менее, уже сейчас некоторые роботы способны за секунду создавать вполне точный трехмерный образ окружающего пространства при нахождении внутри зданий типовых конструкций с жестко предопределенной геометрией [9].

У строительных роботов, у которых предъявляются повышенные требования к точности отработки траектории движения, на тактическом уровне используют алгоритмы управления по скорости и положению. Их особенностью является формирование для исполнительного уровня вектора управляющих сигналов, пропорциональных обобщенным скоростям, рассчитанным на основе динамических моделей с учетом погрешностей отработки траектории движения на предыдущем шаге управления.

Тенденция к упрощению процесса управления привела к разработке алгоритма управления по вектору силы. В нем имитационно реализуется управление по заданному направлению. При этом методе управления следящие приводы развивают совокупность обобщенных сил, которые динамически эквивалентны имитационным силам, обеспечивающим при приложении силы к рабочему органу переход в заданное положение по определенной траектории [10].

В зависимости от задач и ограничений число датчиков цвета может варьироваться. На рисунке 1.2 представлены модели роботов с различной конфигурацией датчиков цвета. На моделях варьируется количество и расположение сенсоров.

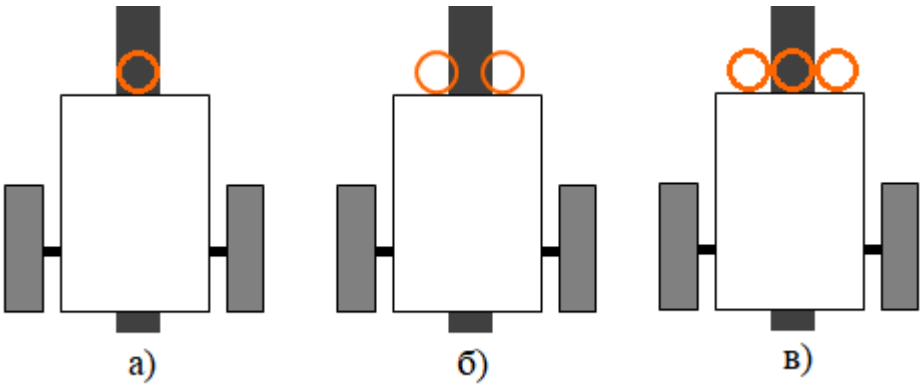


Рисунок 1.2 – Модели роботов, вид сверху

Вне зависимости от аппаратной платформы модель робота можно представить в виде нескольких материальных точек, которые соответствуют центру и датчикам.

С увеличением числа датчиков, увеличивается стоимость конечного изделия, увеличивается потребление энергии, однако до определенного момента увеличение положительно влияет на качество прохождения траектории.

Задача движения по заранее определенной линии является классической. Существует соревнование робототехников по созданию алгоритмов и механизмов, способных передвигаться вдоль определенной дороги [11].

На рисунке 1.3 представлен один из примеров трассы, используемой на соревнованиях. Робот должен набрать максимальное количество очков, двигаясь по линии заданной траектории от зоны старта до зоны финиша. Очки начисляются за пройденную дистанцию. Если во время попытки робот съедет с линии, т.е. окажется всеми колесами или другими деталями, соприкасающимися с полем, с одной стороны линии, то робот останавливается членами жюри и получает очки, заработанные до этого момента.

Если во время попытки робот станет двигаться неконтролируемо, то попытка завершается. В начале попытки робот выставляется в зоне старта

так, чтобы все касающиеся поля части робота находились перед стартовой линией.

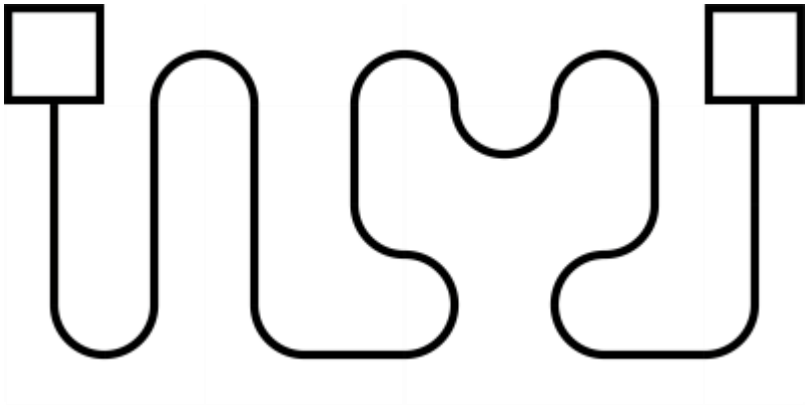


Рисунок 1.3 – Пример траектории,
вид сверху

Поле состоит из секций, на которых отмечена траектория, по которой должен следовать робот. Траектория может отмечаться чёрной линией на белом фоне, либо белой линией на чёрном фоне.

Игровое поле может состоять из различных элементов: отрезков, ломанных линий, угловых элементов. На рисунке 1.4 представлены эти элементы без учета поворота.

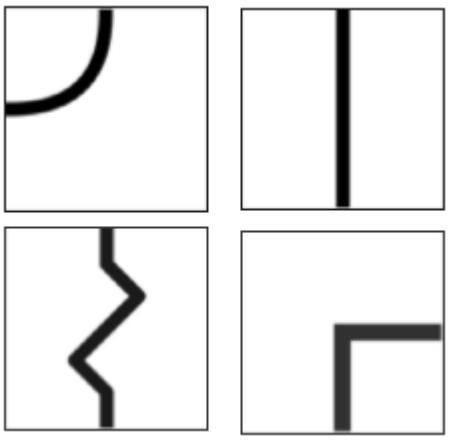


Рисунок 1.4 – Элементы
игрового поля, вид сверху

На рисунке 1.5 представлена трасса, составленная из допустимых элементов. Элементы стыкуются стык в стык, не допустимо использовать самопересечения, непредвиденно обрывать линию, а также не указывать точку старта и конца движения.

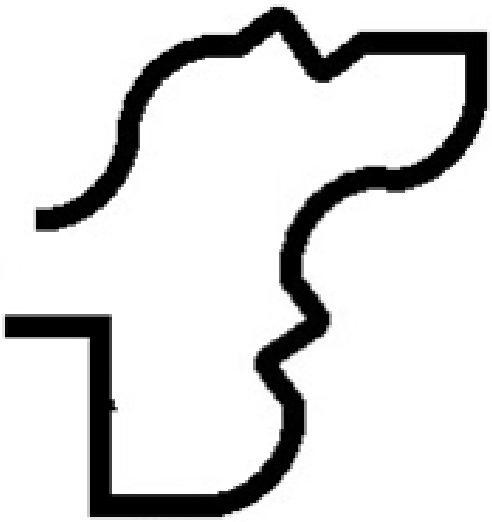


Рисунок 1.5 – Трасса из
допустимых элементов

1.2 Обзор существующих алгоритмов движения

Существует множество подходов для решения задачи следования по линии [12, 13]. Выбор одного из них зависит от конкретной конструкции робота, от количества сенсоров, их расположения относительно колёс и друг друга.

Разберем решение задачи движения по заданной траектории на примере робота с одним датчиком. Действие алгоритма основано на том, что в зависимости от степени перекрытия, пучка подсветки датчика чёрной линией, возвращаемые датчиком показания дискретно варьируются. Робот сохраняет положение датчика света на границе чёрной линии.

Преобразовывая входные данные от датчика света, система управления формирует значение скорости поворота робота. Движение робота напоминает раскачивание. Датчик освещенности следит за левой или правой границей линии: чуть робот съедет на белую часть поля – регулятор возвращает робота на границу, начнет датчик перемещаться вглубь черной линии – регулятор выправляет его обратно [14]. Схема движение робота представлена на рисунке 1.6.

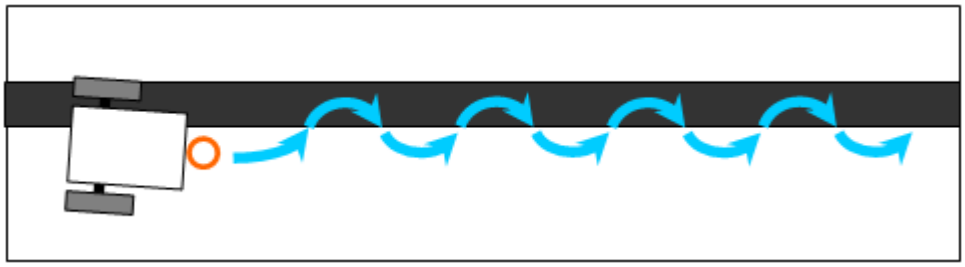


Рисунок 1.6 – Движение робота с одним датчиком, вид сверху

У данного подхода есть некоторый недостаток – если робот следит за левой границей линии, то на правых поворотах он не сразу определяет искривления траектории и, как следствие, тратит большее время на поиск линии и поворот. Причем, чем, круче поворот, тем дольше по времени происходит этот поиск. На рисунке 1.7 представлена проблемная ситуация, когда робот может сбиться с пути

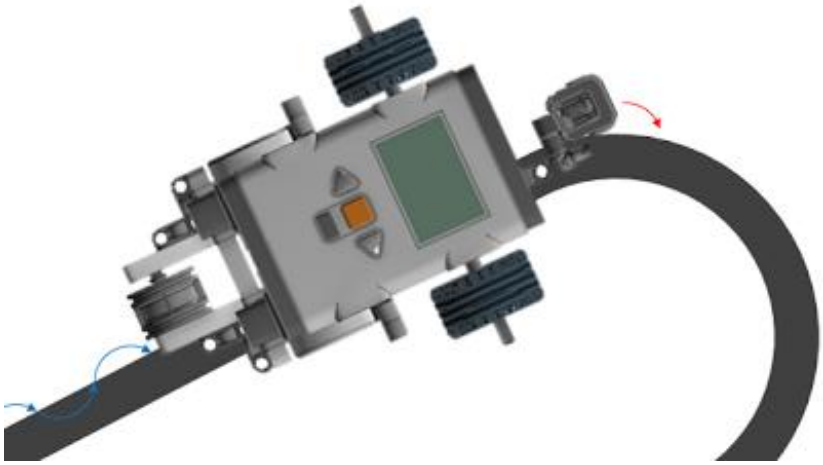


Рисунок 1.7 – Робот с одним датчиком на крутом повороте

Рассмотрим движение по линии с использованием двух датчиков цвета. Датчики нужно установить таким образом, чтобы черная линия проходила между ними. Как показано на рисунке 1.8.

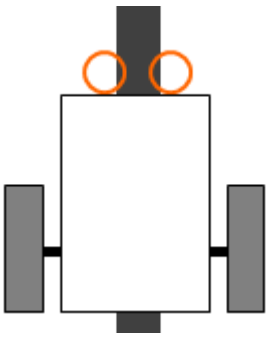


Рисунок 1.8 – Положение робота
с двумя датчиками, вид сверху

Опишем тривиальный алгоритм движения робота. Если оба датчика видят белый цвет – двигаемся вперед. Если один из датчиков видит белый, а другой черный – поворачиваем в сторону черного. Если оба датчика видят черный цвет – мы на перекрестке (например, остановимся). Для реализации алгоритма нам потребуется отслеживать показания обоих датчиков, и только после этого задавать движение роботу [15]. На рисунке 1.9 представлен робот на базе lego mindstorms ev3 с двумя датчиками цвета.

С помощью ресурсных наборов состоящих из огромного количества элементов LEGO Technic, например, колес, датчиков, шестерен, поворотных и крепежных деталей появилась возможность создавать и прототипировать различные конструкции роботов. Таким образом нарастить еще один датчик цвета на платформу не составляет особого труда.

На рисунке 1.9 представлен прототип робота с двумя псервоприводами, позволяющими совершать движение, двумя светочувствительными датчиками, с помощью которых робот получает информацию для дальнейшего анализа и принятия решения.

Прототип построен на базе платформы Lego Mindstorms ev3, эта платформа является достаточно гибкой и содержит большое количество разнообразных элементов и деталей. Например, датчики температуры, датчики давления, датчики света, ультразвуковой датчик, измеряющий расстояние, гироскопические датчики, позволяющие измерять движение вращения робота, а также отслеживать изменения его положения и углы, датчики касания, дает возможность отслеживать нажатия на кнопку.

Рисунок 1.9 – Робот с двумя датчиками

У различных платформ существуют свои особенности и практики программирования, так, например, основные способы написания программ для платформы lego mindstorms ev3 является визуальный редактор, в котором процесс реализации алгоритма сводится к перетягиванию функциональных блоков, пример программы с использованием визуального редактора представлен на рисунке 1.10.

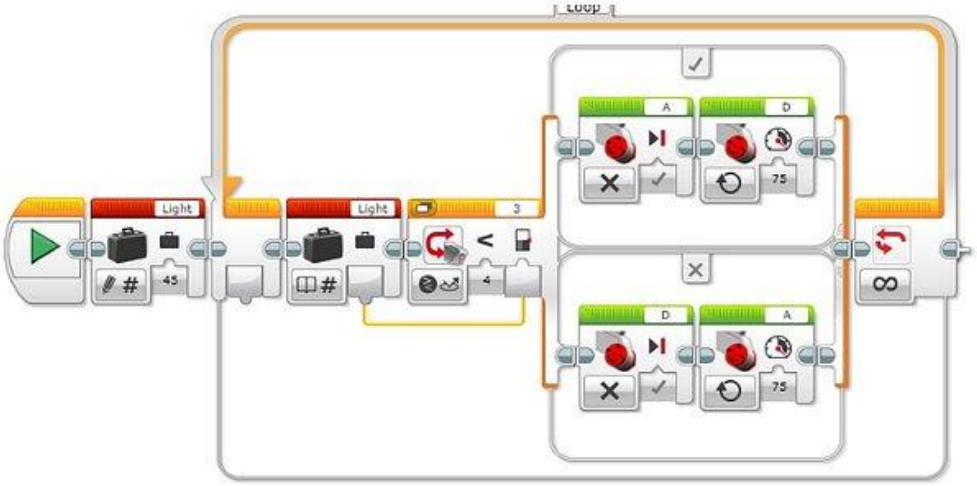


Рисунок 1.10 – Программа для lego mindstorms ev3

Помимо визуального редактора практически на всех платформах существует возможность написания текста на языках высокого уровня. Например, для Lego доступны языки EV3 BASIC, Python и другие, для систем на базе Raspberry Pi доступны языки Scratch, Python, Java и другие, для Arduino доступен C/C++ и Python [16].

1.3 Обзор алгоритмов обучения с подкреплением

Искусственный интеллект – очень широкая область исследований, посвященная когнитивным способностям машин: обучение определенному поведению, упреждающее взаимодействие с окружающей средой, способность к логическому выводу и дедукции, компьютерное зрение, распознавание речи, решение задач, представление знаний, восприятие действительности и многое другое.

Менее формально под искусственным интеллектом понимается любая ситуация, в которой машины имитируют интеллектуальное поведение, считающееся присущим человеку.

Искусственный интеллект заимствует методы исследования из информатики, математики и статистики. Машинное обучение – отрасль искусственного интеллекта, посвященная тому, как обучать компьютеры решению конкретных задач без программирования того, как именно надо решить задачу.

Основная идея машинного обучения заключается в том, что можно создавать алгоритмы, способные обучаться на данных и впоследствии давать предсказания.

Различают три основных типа машинного обучения: обучение с учителем, обучение без учителя и обучение с подкреплением.

Классификация основных видов машинного обучения показана на рисунке 1.11.

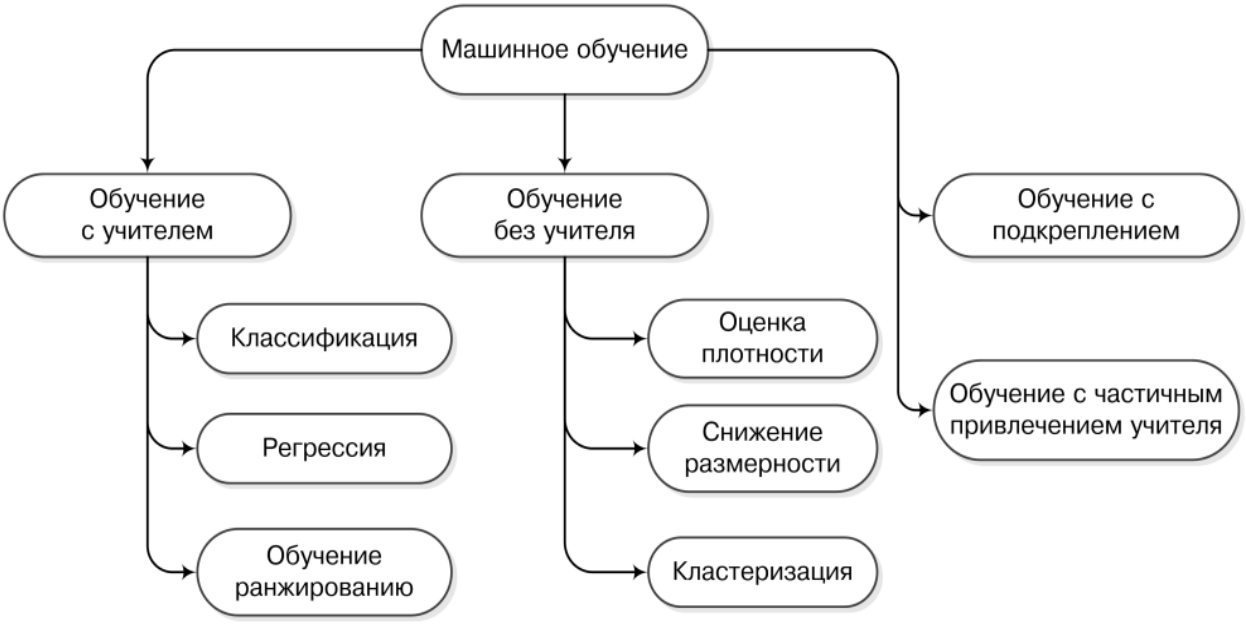


Рисунок 1.11 – Классификация машинного обучения

Цель машинного обучения – научить машину (точнее, программу) решать задачу, предъявив ей несколько примеров (с правильными и неправильными решениями) [17].

Обучение с учителем (Supervised learning) – один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов – пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества. Основная задача обучения с учителем состоит в том, чтобы на маркированных тренировочных данных извлечь модель, которая позволяет делать прогнозы о ранее не встречавшихся или будущих данных. Под учителем понимается либо сама обучающая выборка, либо тот, кто указал на заданных объектах правильные ответы [18].

В случае обучения с учителем машине предъявляются данные и правильные результаты, а цель состоит в том, чтобы машина обучилась на этих примерах и смогла выдавать осмысленные результаты для данных, которые раньше не видела. В случае обучения без учителя машине предъявляются только сами данные, а она должна выявить структуру без постороннего вмешательства. В случае обучения с подкреплением машина ведет себя как агент, который взаимодействует с окружающей средой и обучается находить варианты поведения, приносящие вознаграждение [19].

Обучение без учителя (Unsupervised learning) – один из разделов машинного обучения. Изучает широкий класс задач обработки данных, в которых известны только описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами [19].

Обучение без учителя часто противопоставляется обучению с учителем, когда для каждого обучающего объекта задаётся «правильный ответ», и требуется найти зависимость между объектами и ответами.

Задача обучения с подкреплением состоит в выработке системы (агента), которая улучшает свое качество на основе взаимодействий со средой. Поскольку информация о текущем состоянии среды, как правило, содержит и так называемый сигнал вознаграждения, обучение с подкреплением можно представить как область, имеющую отношение к обучению с учителем. Однако в обучении с подкреплением эта обратная связь является не меткой или значением, раз и навсегда определенными в результате прямых наблюдений, а мерой того, насколько хорошо действие было оценено функцией вознаграждения. В ходе взаимодействия со средой агент на основе разведочного подхода путем проб и ошибок или совещательного планирования может использовать обучение с подкреплением для вычленения серии действий, которые максимизируют это вознаграждение [18].

В обучении с учителем, когда мы тренируем модель, мы знаем правильный ответ заранее, а в обучении с подкреплением мы определяем меру вознаграждения за выполненные агентом отдельно взятые действия. В обучении без учителя мы имеем дело с немаркированными данными или данными с неизвестной структурой. Используя методы обучения без учителя, мы можем разведать структуру данных с целью выделения содержательной информации без контроля со стороны известной результирующей переменной или функции вознаграждения.

Обучение с подкреплением – это обучение тому, что надо делать, как следует отображать ситуации в действия, чтобы максимизировать некоторый сигнал поощрения (вознаграждения), принимающий числовые значения. Обучаемому не говорят, какое действие следует предпринять, как это имеет место в большинстве подходов к обучению машин. Вместо этого он, пробуя выполнять различные действия, должен найти, какие из них принесут ему наибольшее вознаграждение. В наиболее интересных и важных случаях действия могут влиять не только на вознаграждение, получаемое немедленно, но также и на возникающую ситуацию, а через нее на все последующие поощрения. Эти две характеристики поиск методом проб и ошибок, а также отсроченные поощрения представляют собой две наиболее важные отличительные черты обучения с подкреплением. Определение обучения с подкреплением дается не через описание методов обучения, а путем выявления характерных свойств задачи обучения [20].

Одна из наиболее серьезных проблем, возникающих в обучении с подкреплением и отсутствующих в других видах обучения, это проблема поиска компромисса между изучением и применением. Чтобы получить большее вознаграждение, агент, обучающийся с подкреплением, должен предпочитать действия, которые он уже проверил в прошлой своей деятельности и обнаружил, что они эффективны с точки зрения получения поощрения. Однако, чтобы обнаруживать их, надо пробовать выполнять такие действия, которые еще не выполнялись ранее. Агент должен применять

те действия, про которые уже известно, что они позволяют получить вознаграждение, но он должен также и изучать новые действия, чтобы иметь возможность делать лучший выбор в будущем. Проблема состоит в том, что нельзя только использовать уже проверенные действия или только искать новые эффективные действия, поскольку это ведет к провалу попыток решения задачи. Агент должен пытаться предпринимать разнообразные действия и благоприятствовать тем из них, которые окажутся лучшими. В задачах стохастического характера каждое из действий должно быть повторено многократно, чтобы добиться получения надежной оценки ожидаемого вознаграждения. Все агенты, обучающиеся с подкреплением, имеют явным образом выраженные цели, могут воспринимать особенности среды, а также выбирать действия, влияющие на эту среду. Более того, обычно с самого начала предполагается, что агент должен действовать, несмотря на существенную неопределенность в среде.

В теории обучения с подкреплением существует дилемма между исследованием окружающей среды и использованием уже накопленных знаний. В момент принятия решения объект может использовать уже накопленные в процессе обучения знания об окружающей среде и выбрать оптимальное, по его мнению, действие. Однако накопленные знания агента часто не совсем точно отражают реальную динамику внешней среды. Возможно, существует действие, которое приведет к еще большей награде в будущем, т. е. агенту иногда следует выбрать действие наугад, чтобы обнаружить действительно оптимальную политику. Однако слишком частые исследования будут приносить гораздо меньше награды, чем простое использование накопленных знаний. Описанные проблемы ярко проявляются при обучении управлению агентом в виртуальных средах больших размерностей [20].

Параметром, определяющим степень исследовательского поведения агента, является параметр стохастичности. Когда в обучение с подкреплением вовлекается планирование, приходится предпринимать

усилия для обеспечения взаимодействия между планированием и выбором действий в реальном масштабе времени, а также для поиска способов получения и улучшения моделей среды. Примеры прикладных задач:

- управление технологическими процессами;
- управление роботами;
- персонализация показов рекламы в Интернете;
- управление ценами и ассортиментом в сетях продаж;
- игра на бирже;
- маршрутизация в телекоммуникационных сетях;
- маршрутизация в беспроводных сенсорных сетях;
- логические игры (шашки, нарды, и т. д.).

Обучение с подкреплением можно считать частным случаем обучения с учителем, где учителем является среда или ее модель. Формально простейшая модель обучения с подкреплением включает следующие элементы:

- множества состояний окружения;
- множества действий;
- множества скалярных значений «выигрышей».

В произвольный момент времени агент характеризуется состоянием и множеством возможных действий. Выбирая действие, он переходит в следующее состояние и получает выигрыш. Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию, которая максимизирует величину выигрыша.

Для формирования функции оценивания состояния используются различные подходы, например метод Монте-Карло. Методы Монте-Карло требуют только наличия опытных данных – последовательностей выборок состояний, действий и вознаграждений, получаемых путем непосредственного или смоделированного взаимодействия с окружающей средой.

Обучение на основе опытных данных, получаемых и используемых непосредственно в процессе взаимодействия системы со средой, дает поразительно хорошие результаты, поскольку не требует предварительного знания динамики окружающей среды, но, как и методы на основе модели среды, позволяет достигать оптимального поведения [19].

Обучение на основе смоделированных опытных данных также достаточно эффективно. Несмотря на то, что при таком подходе к обучению по-прежнему требуется модель, она необходима лишь для формирования переходов из одного состояния в другое, а не полного распределения вероятностей всех возможных переходов, которое требуется в методах динамического программирования [21].

Существует удивительно много случаев, в которых достаточно легко сформировать элемент набора опытных данных согласно желаемым распределениям вероятностей, но невозможно получить такие распределения в явном виде. Методы Монте-Карло позволяют решать задачу обучения с подкреплением, основываясь на выгоде, осредненной на некоторой выборке. Для обеспечения уверенности в том, что выгода будет вполне определенной, методы Монте-Карло используются только для заданий, состоящих из эпизодов, т. е. предполагается, что опытные данные разбиваются на эпизоды, а все эпизоды, в конечном счете, завершаются вне зависимости от выбираемых действий. Только по завершении эпизода происходит оценивание значения ценности и изменение стратегии. Таким образом, методы Монте-Карло реализуют процессы инкрементного типа [19].

Суть всех методов, основанных на данном подходе, состоит в том, что с ростом числа наблюдений, дающих значение выгоды, среднее значение выгоды сходится к ожидаемой величине. Пусть появление одного из состояний в эпизоде называется посещением. Монте-Карло метод всех посещений оценивает награду, как среднее значение выгод, соответствующих всем посещениям в некоторой совокупности эпизодов.

При отсутствии модели намного полезнее оценивать ценности действий, нежели состояний. Если модель имеется, то ценностей состояний вполне достаточно для определения стратегии: надо просто выполнить один шаг вперед и выбрать действие, которое приводит к наилучшей комбинации вознаграждения и следующего состояния. Задача оценивания стратегии на основе ценностей действий состоит в вычислении ожидаемой выгоды при определенном начальном состоянии, выполнении действия и следовании в дальнейшем некоторой стратегии. При таком подходе в обучении с подкреплением возникает проблема не посещенных состояний или не исследованных действий.

Для любой функции ценности действия, соответствующей жадной стратегии является стратегия, которая для каждого состояния выбирает значение с максимальной ожидаемой выгодой по формуле:

, (1)

где – состояние системы,

– действие,

– функция ожидаемой выгоды.

Улучшение стратегии может быть осуществлено за счет построения каждой стратегии как жадной по отношению к функции ожидаемой выгоды. В этом случае к стратегиям можно применить теорему об улучшение стратегии [19].

Данная теорема гарантирует, что каждая следующая стратегия обязательно лучше, чем предыдущая, либо равна ей в случае, если обе стратегии оптимальны. Это, в свою очередь, гарантирует, что весь процесс сходится к оптимальной стратегии и оптимальной функции ценности.

Пошаговая реализация, имеет дело со случаем простого или арифметического усреднения, в котором каждая выгода имеет одинаковый вес. Предположим, что вместо этого необходимо реализовать взвешенное усреднение, в котором каждая выгода имеет вес, и требуется вычислить величину выгоды. Тогда величина награды вычисляется по формуле:

, (2)

где – величина награды на шаге ,

– вес награды шага ,

– награда, полученная на шаге.

В методах Монте-Карло обычно происходит чередование операций оценивания и улучшения от эпизода к эпизоду. По завершении каждого эпизода полученная выгода используется для оценивания стратегии, после чего данная стратегия улучшается во всех состояниях данного эпизода, посещение которых имело место.

Другой подход к обучению с подкреплением – это обучение на основе временных различий. Обучение на основе временных различий представляет собой сочетание идей метода Монте-Карло и динамического программирования [21, 22, 23].

Как и в методах Монте-Карло, в методах на основе временных различий процесс обучения может основываться непосредственно на получаемом опыте без предварительных знаний о модели поведения окружающей среды. Как и в динамическом программировании, методы обновляют расчетные оценки, основанные отчасти на других полученных оценках, не дожидаясь окончательного результата.

Методы на основе временных различий и методы Монте-Карло для решения задачи предсказания используют имеющийся опыт. При наличии некоторого опыта следования стратегии оба указанных метода корректируют свои оценки функции выгоды. Если имеет место посещение нетерминального состояния в некоторый момент времени, то оба метода корректируют свои оценки, основываясь на том, что случилось после этого посещения [22]. Формула имеет следующий вид:

, (3)

где – фактическая выгода, полученная после момента времени *t,*

–длинна шага,

t – момент времени.

Преимуществом метода над методами Монте-Карло является то, что первые по своей сути являются интерактивными, полностью инкрементными методами [21].

В случае с методами Монте-Карло каждый раз необходимо ждать завершения эпизода так, как только тогда становится известна выгода, тогда как в случае с методами, основанными на временных различиях, необходимо дождаться лишь следующего временного шага. Достаточно часто данная особенность имеет решающее значение. В некоторых ситуациях эпизоды могут быть настолько продолжительными, что задержки процесса обучения, связанные с необходимостью завершения эпизодов, будут слишком велики. В других ситуациях имеются непрерывные задачи, а эпизоды отсутствуют как таковые.

В случае с методами, основанными на временных различиях, необходимо дождаться только следующего временного шага. Непосредственно в момент времени формируется целевое значение оценки, после чего производится необходимая корректировка с учетом уже имеющегося вознаграждения и оценки состояния по формуле:

, (4)

где – имеющееся вознаграждение,

–длина шага,

–коэффициент приведения.

Методы на основе временных различий строят свои оценки, частично основываясь на других оценках. Они делают предположения, основываясь на предположениях – они самонастраиваются [19]. Очевидно, что одним из преимуществ является то, что они не требуют знания модели окружающей среды с ее вознаграждениями и вероятностным распределением состояний. Также метод не требует дожидаться окончания эпизода, что улучшает обучающую способность и ускоряет получение обратного отклика.

Одним из наиболее важных достижений в обучении с подкреплением стало развитие управления по методу с разделенной оценкой ценности

стратегий, известного как Q-обучение [19]. Его простейшая форма, одношаговое Q-обучение, определяется следующим образом:

, (5)

где Q – искомая функция действия ценности,

–длинна шага,

–коэффициент приведения.

t – момент времени,

– состояние системы,

– действие.

Стратегия определяет то, какие пары состояния и действия посещаются и корректируются. Тем не менее для обеспечения сходимости необходимо лишь, чтобы все пары продолжали корректироваться.

1.4 Выводы по разделу

Изучение роботов и робототехники, показало, что задача движения робота по заданной линии популярна и вынесена в отдельное соревнование, проводимое по всему миру, данное направление называется «Траектория» или «Line Follower». Многие исследователи пытаются решить эту задачу, используя различные алгоритмы, приемы, различные конфигурации и модели роботов, отличающиеся набором датчиков, двигателей, а также внешней формой, динамикой движения.

Решение задачи позволит усовершенствовать текущие производственные процессы, связанные с транспортировкой грузов и логистикой, охранной деятельностью, улучшит пользовательский опыт при обращении к роботам помощникам.

Одним из преимуществ обучения с подкреплением, является гибкость и слабая связность с задачей, данный метод можно применить к различным условиям и ограничениям.

Были рассмотрены и исследованы разделы обучения с подкреплением: обучение на основе временных разностей, обучение, основанное на методах Монте-Карло и Q-обучение.

Важным выводом является то, что для решения конкретной задачи движения робота по линии можно применить метод обучения с подкреплением, который не противоречит условиям задачи.

Таким образом, для исследуемой задачи необходимо разработать и настроить алгоритм обучения с подкреплением, для решения задачи движения робота по линии.

2 РАЗРАБОТКА АЛГОРИТМА УПРАВЛЕНИЯ РОБОТОМ

2.1 Модель объекта управления

Объектом управления выступает робот с двумя датчиками цвета. В работе используется компьютерная виртуальная модель физического устройства, называемая агент.

Агент представлен в виде нескольких материальных точек, характеризующих следующие элементы:

- центр робота;
- левый сенсор;
- правый сенсор;
- передняя граница корпуса, для определения направления.

Агенту доступно три базовых действия:

- движение вперед;
- поворот против часовой стрелки относительно центра робота;
- поворот по часовой стрелке относительно центра робота.

В работе виртуальный агент взаимодействует с виртуальной средой через вызовы программных функций. Левый и правый светочувствительные сенсоры располагаются по левую и правую сторону от центра робота. Все три основных элемента агента лежат на одной прямой, как изображено на рисунке 2.1.

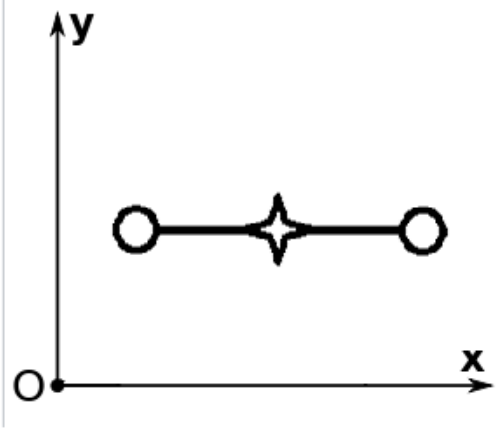


Рисунок 2.1 – Модель агента

Кругами обозначены сенсоры, а звезда посередине – это центр робота. Базовые действия робота осуществляются с помощью линейных преобразований координат. При движении вперед происходит параллельный перенос всех координат агента на вектор направления.

При повороте совершении агентом действия поворота происходит преобразование координат, и поворот вокруг центральной точки агента как показано на рисунке 2.2. Пунктиром изображено новое положение сенсоров после совершения действия поворота.

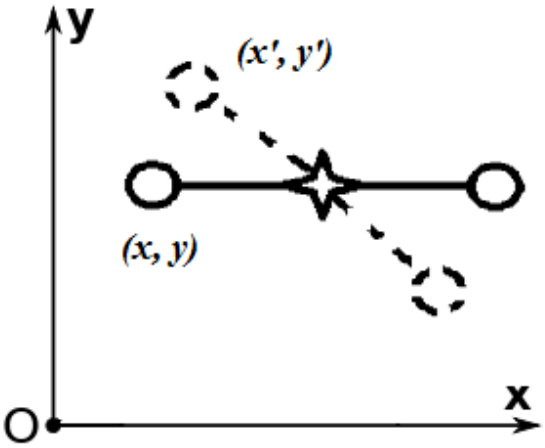


Рисунок 2.2 – Поворот агента

Поворот сенсоров вокруг центра агента осуществляется с помощью линейного преобразования координат. В работе реализованы функции для преобразования и совершения действий над координатами, а именно параллельный перенос, реализованный по формуле 6, поворот относительно начала координат по формуле 7

, (6)

где x, y – координаты точки,

d – величина шага.

Для определения новых координат сенсоров и центра робота используется матрица поворота.

, (7)

где – угол поворота.

2.2 Модель среды и взаимодействия с роботом

Физически робот взаимодействует с полем, на котором нанесена разметка. И для определения уровня засветки может использоваться дискретный датчик, который определяет уровень освещенности.

Например, на рисунке 2.3 представлен цифровой датчик, который может определять цвет или яркость света, поступающего в небольшое окошко на лицевой стороне. Этот датчик может работать в трех разных режимах: в режиме «Цвет», в режиме «Яркость отраженного света» и в режиме «Яркость внешнего освещения».

Рисунок 2.3 – Датчик цвета
EV3 45506

Для определения уровня засветки требуется посчитать засвеченную область. Расчет производится по формуле.

(8)

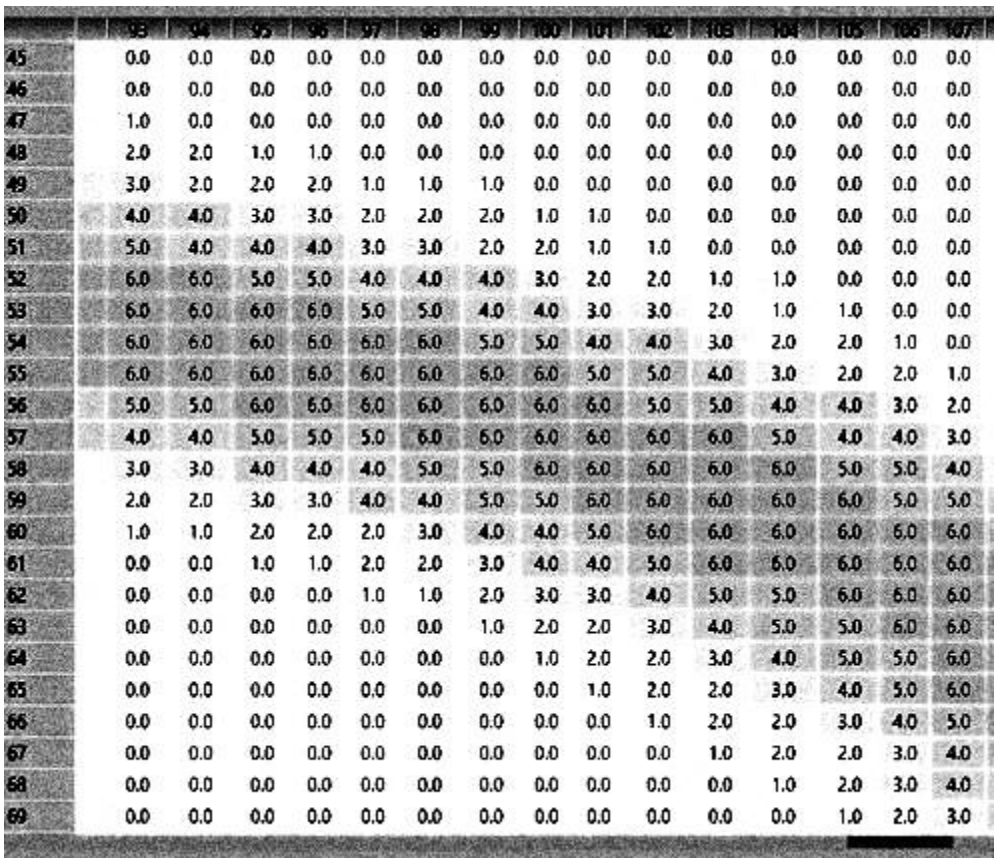
где – координаты карты.

В исследовании используется виртуальная модель поля, представленная в виде числовой матрицы. В ячейках сетки расставлены числа, которые показывают, нанесена ли разметка на заданный сектор или нет. Пример части виртуального поля представлен на рисунке 2.4.

	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
49	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
51	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
53	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
54	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
55	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0
56	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0
57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
58	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
59	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
60	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
61	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
62	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
63	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
65	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0
66	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
67	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

Рисунок 2.4 – Виртуальное поле

Так как датчик, считывающий информацию с поля, имеет возможность различать интенсивность, то исходное виртуальное поле преобразуется, чем ближе к центру линии, тем больше значение уровня засветки. Пример изображен на рисунке 2.5. Благодаря такому подходу к решению задачи появляется возможность моделировать получение данных в виртуальной среде таким образом, как будто данные получены с сенсора ev3 45506. Датчик цвета EV3 предназначен для того, чтобы различать цвета. Датчик цвета различает 7 цветов и может определить отсутствие цвета. Он может работать как датчик освещенности. Измеряет отраженный красный свет и окружающее освещение – от полной темноты до яркого солнечного света. Способен определять различия между белым и черным или цветами: синим, зеленым, желтым, красным, белым и коричневым. Автоматически идентифицируется программным обеспечением EV3.



	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	2.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
49	3.0	2.0	2.0	2.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	4.0	4.0	3.0	3.0	2.0	2.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
51	5.0	4.0	4.0	4.0	3.0	3.0	2.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
52	6.0	6.0	5.0	5.0	4.0	4.0	4.0	3.0	2.0	2.0	1.0	1.0	0.0	0.0	0.0
53	6.0	6.0	6.0	6.0	5.0	5.0	4.0	4.0	3.0	3.0	2.0	1.0	1.0	0.0	0.0
54	6.0	6.0	6.0	6.0	6.0	6.0	5.0	5.0	4.0	4.0	3.0	2.0	2.0	1.0	0.0
55	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	5.0	5.0	4.0	3.0	2.0	2.0	1.0
56	5.0	5.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	5.0	5.0	4.0	4.0	3.0	2.0
57	4.0	4.0	5.0	5.0	5.0	6.0	6.0	6.0	6.0	6.0	6.0	5.0	4.0	4.0	3.0
58	3.0	3.0	4.0	4.0	4.0	5.0	5.0	6.0	6.0	6.0	6.0	6.0	5.0	5.0	4.0
59	2.0	2.0	3.0	3.0	4.0	4.0	5.0	5.0	6.0	6.0	6.0	6.0	6.0	5.0	5.0
60	1.0	1.0	2.0	2.0	2.0	3.0	4.0	4.0	5.0	6.0	6.0	6.0	6.0	6.0	6.0
61	0.0	0.0	1.0	1.0	2.0	2.0	3.0	4.0	4.0	5.0	6.0	6.0	6.0	6.0	6.0
62	0.0	0.0	0.0	0.0	1.0	1.0	2.0	3.0	3.0	4.0	5.0	5.0	6.0	6.0	6.0
63	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	3.0	4.0	5.0	5.0	6.0	6.0
64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	3.0	4.0	5.0	5.0	6.0
65	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	3.0	4.0	5.0	6.0
66	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	3.0	4.0	5.0
67	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	3.0	4.0
68	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	3.0	4.0
69	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	3.0

Рисунок 2.5 – Карта интенсивности

Карта интенсивности показывает уровень засветки и моделирует показание датчика, полученного в ходе движения робота по линии.

2.3 Модель процесса обучения

Процесс обучения построен на базе алгоритма Q-learning. На основе получаемого от среды вознаграждения агент формирует функцию полезности Q, что в последствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-learning состоит в том, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя модели окружающей среды.

Выбранный метод основан на введении функции , отражающей ценность каждого возможного действия агента для текущего состояния, в котором сейчас находится симуляция. Эта функция задает оценку агентом той награды, которую он может получить, совершив в определенный ход

определенное действие. А также она включает в себя оценку того, какую награду агент может получить в будущем. Процесс обучения представляет из себя итерационное уточнение значения функции на каждом шаге агента. В первую очередь следует определить величину награды, которую агент получит в этот ход. Выбор действия определяется следующим образом:

, (9)

где – текущее состояние системы;

– доступные действия;

– количество доступных действий.

Обучение агента происходит эпохами, каждая эпоха задает новое окружение для агента. Однако накопленный опыт агента сохраняется и с увеличением числа эпох обучения опыт увеличивается. На рисунке 2.6 представлен алгоритм обучения и процесс обновления параметров.

Рисунок 2.6 – Алгоритм Q-learning

Каждая эпоха представляет проход робота по карте с корректировкой матрицы весовых коэффициентов Q. Стартовое положение робота задается следующим образом, на линии, вдоль которой должен пройти робот, выбираются все точки, с которых можно совершить шагов вперед, и для каждой эпохи случайным образом берется одна из них, после этого к каждой координате прибавляется случайное целое число из диапазона . Это делается для того, чтобы увеличить число начальных состояний и препятствовать процессу переобучения.

2.4 Функция вознаграждения

В качестве вознаграждения за совершенное действие рассчитывается расстояние от центра робота до ближайшей не посещенной точки, находящейся на центральной линии. Чем ближе агент, тем большую награду он получит, чем дальше, тем меньше награды.

Для определения ближайшего расстояния используется алгоритм поиска в ширину [14]. Для того, чтобы определить, где находится центр, на начальном этапе производится сжатие, как показано на рисунке 2.7. Далее проставляются временные метки.

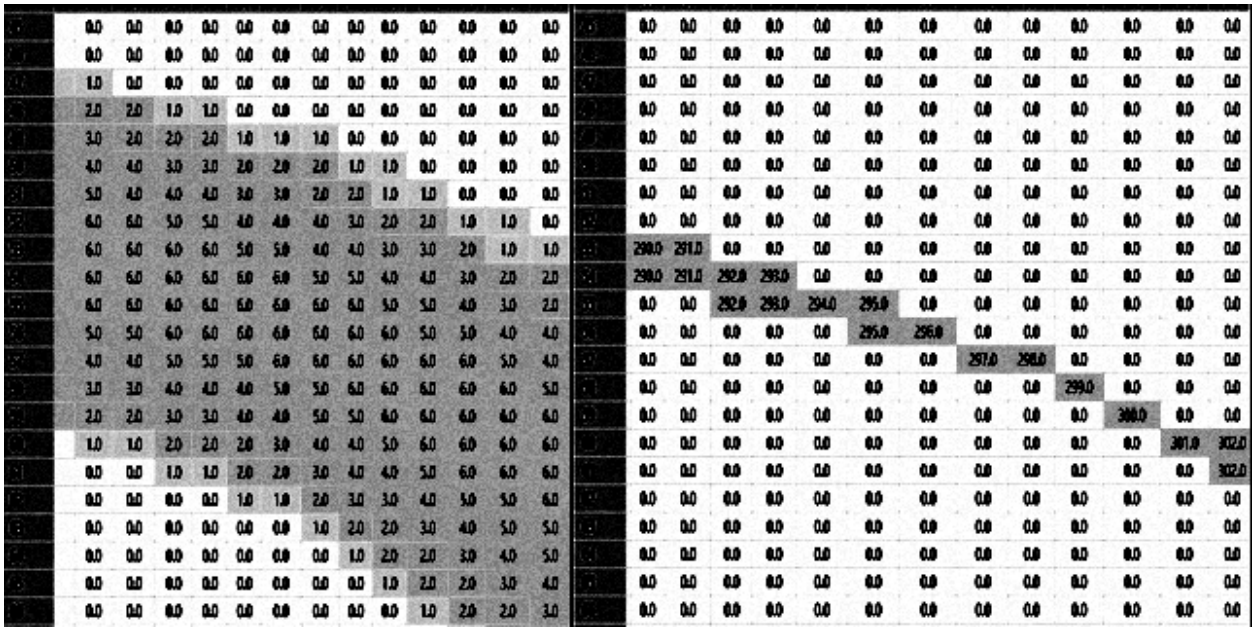


Рисунок 2.7 – Центр линии

После сжатия исходной линии и определения ее центральных позиций, происходит процесс расстановки количества шагов, требуемых для достижения центров от начала движения. Дистанция вычисляется как минимальная длина пути.

(10)

(11)

где , – координаты в порядке обхода поиском в ширину.

Определение уровня засветки считается по формулам ниже.

(12)

(13)

где – ширина зазора между сенсорами.

Центральная точка считается посещенной, если агент либо посетил ее или расстояние между агентом и точкой меньше .

Этот шаг требуется для того, чтобы обучать агента двигаться вперед к следующей центральной позиции. Без введения этой информации возможны ситуации, когда агент начинает срезать путь, не посещая все обязательные элементы линии.

2.5 Выводы по разделу

Для решения поставленных задач была разработана модель объекта управления, включающая определения доступных агенту действий и состояний. Была определена виртуальная среда, позволяющая изучить поведение робота и проводить симуляции обучения. Была разработана модель обучения на основе алгоритма Q-learning. Введена функция вознаграждения, основанная на расстояния от агента до центра линии, использующаяся в процессе обучения.

3 ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

3.1 Интерфейс взаимодействия

Вопросы интеграции приложений предприятия с внешним модулем очень важны, так как простой интерфейс позволяет расширять функционал существующих систем и объединять несколько сложных проектов без внесения большого количества изменений в существующую кодовую базу.

Сторонним разработчикам предоставляется API (программный интерфейс приложения, интерфейс прикладного программирования) для использования во внешних программных продуктах. В качестве интеграционной возможности реализован программный интерфейс на языке высокого уровня Python.

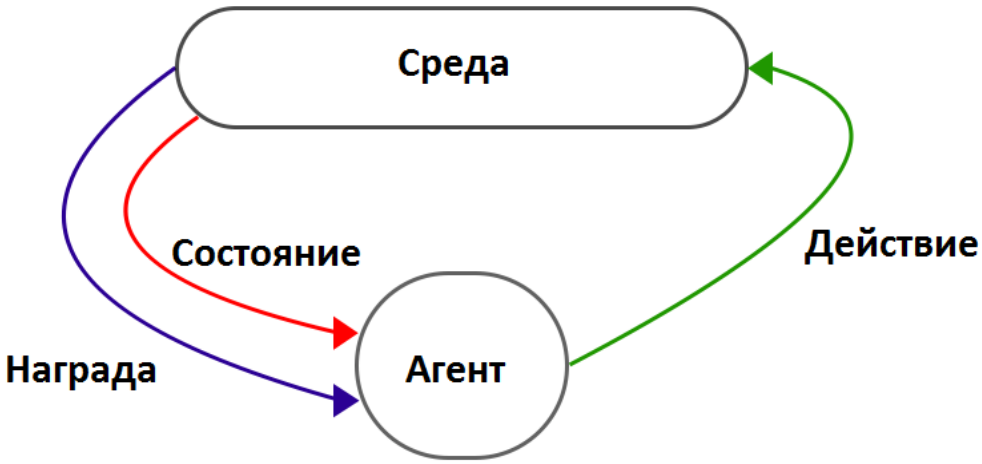


Рисунок 3.1 – Взаимодействие агента и среды

От пользователя программы требуется реализовать следующее:

- класс, унаследованный от базового класса среды выполнения;
- симуляцию взаимодействие между средой и агентом.

Базовый класс среды выполнения требует реализации следующих методов:

- получение награды;
- получение текущего состояния среды, доступной агенту;
- получения количества допустимых действий в текущем состоянии;

• обработка выбранного агентом действия.

Uml диаграмма абстрактного класса представлена на рисунке 3.2.

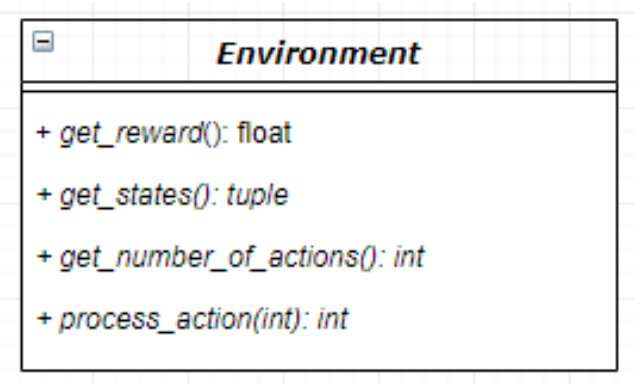


Рисунок 3.2 Диаграмма класса

Разработка взаимодействия между средой и агентом реализуется пользователем программного модуля. Для разработчика возможны различные варианты симуляции работы модуля обучения с подкреплением, например некоторые из них:

• моделирование фиксированного числа шагов;

• достижение определенной суммы наград;

• проверка целевого состояния системы.

3.2 Реализация среды

В работе была реализована виртуальная среда позволяющая осуществлять симуляции и процесс обучения. Среда представляет следующие обязательные функции, реализованные на языке Python:

• получить текущее состояние;

• получить текущую награду;

• получить список доступных действий;

• обработать выбранное действие.

Текущее состояние среды определяется последними данными, считанными с датчиков цвета. Так как робот оснащен двумя сенсорами, то данных, получаемых от среды в два раза больше.

Пример вектора состояний, полученных от среды, представлен на формуле (14). Так как в работе используется два датчика, то количество элементов вектора состояний равно удвоенному числу последних данных, полученных с сенсоров, как показано на формуле (15).

Тогда мощность множества всех возможных состояний, которые может получить агент в ходе работы выражается экспоненциально по формуле (16).

, (14)

, (15)

, , (16)

где – величина дискретизации;

– состояния системы;

– множество всех состояний системы;

– глубина запоминания.

Текущая награда агента определяется как расстояние от центра робота до ближайшей еще не посещенной центральной точки. Для определения награды используется алгоритм, основанный на поиске в ширину [14].

Поиск в ширину алгоритмов поиска в графе. Особенностью этого алгоритма является то, что он находит все вершины на расстоянии до того, как начинает искать любую из вершин на расстоянии .

Одним из хороших способов визуализации поиска в ширину будет представление о том, что он строит дерево, по одному уровню за раз. Алгоритм добавляет всех детей начальной вершины до того, как обнаруживает любого из её внуков.

Суть алгоритма поиска в ширину в том, что мы обходим связный граф таким образом, что сначала мы рассматриваем родителя, потом по очереди рассматриваем его предков, потом рассматриваем предков его предков и так далее, пока не посетим все вершины в графе. Пример обхода графа представлен на рисунке 3.3.

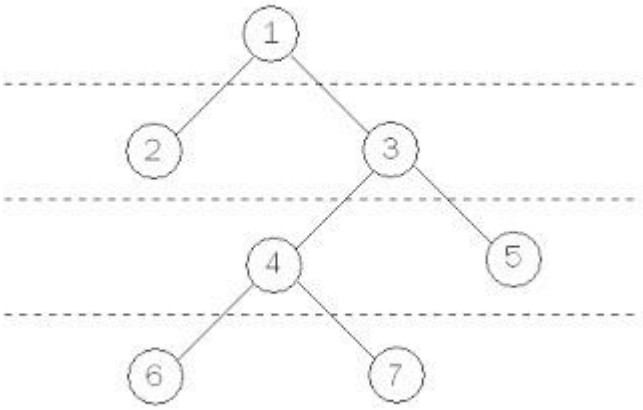


Рисунок 3.3 – Обход графа

Первой вершине (родителю всех остальных вершин) приписываем метку 1. Рассматриваем все смежные с ней вершины и приписываем им метку 2. Дальше рассматриваем окружение вершин с меткой 2 и присваиваем метку 3 всем вершинам, кроме самой главной (родителя всех вершин).

Так как в работе используется карта с отмеченными показаниями, то и кратчайшие расстояния ищутся по ней.

Список действий, доступных агенту определен заранее:

- движение вперед на одну условную единицу;
- поворот против часовой стрелки и шаг вперед;
- поворот по часовой стрелки и шаг вперед.

3.3 Реализация агента

Агент реализует следующие базовые функции, необходимые для работы обучения с подкреплением:

- обработка состояния и награды, полученные от среды;
- получение выбранного действия на основе анализа состояния;
- получение накопленного опыта, представленного в виде матрицы Q.

Обучение агента производится с помощью алгоритма Q-learning, в основе которого лежит функций полезности – Q-функция. Для вычисления ее значений используется матрица Q, представленная на рисунке 3.4.

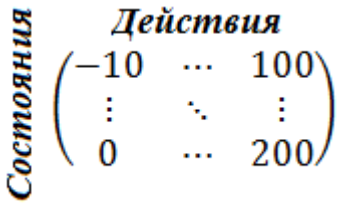


Рисунок 3.4 – Матрица
полезности Q

Прямоугольная матрица Q по одному измерению содержит всевозможные состояния, в которые может попасть агент обучения, а по второму измерению номера действий, которые может предпринять агент. В ячейках матрицы находится накопленная награда, полученная из конкретного состояния по определенному действию.

3.4 Организация вычислительного эксперимента

Эксперимент с роботом состоял из нескольких этапов, приведенных ниже.

1. Этап обучения.
2. Этап контрольной оценки обученного агента.
3. Оценка работы тривиального алгоритма.
4. Сравнение результатов.

Этап обучения происходил следующим образом, было определено 10000 эпох, на каждой эпохе случайным образом выбиралась одна из 20 заранее подготовленных трасс, состоящих минимум из одного допустимого элемента.

На основе случайно выбранной трассы, на каждой эпохе обучения конструировалась новая среда.

Для сохранения результатов обучения агент оставался прежним, сохранял внутреннее состояние, использовал существующую матрицу , и накапливал опыт.

В ходе обучения агент мог предпринимать недопустимые действия, например движение за пределами виртуальной среды, набирал большую дистанцию и отдалялся от линии. В таких случаях агент получал штрафные очки, которые могут быть компенсированы только возвращением на верный курс.

При получении слишком большого штрафа эпоха досрочно заканчивалась, без возможности исправить ситуацию.

Примеры карт, на которых проходило обучение агента, представлены на рисунке 3.5. Обучающий набор отличается количеством элементов, комбинацией различных элементов и их поворотами.

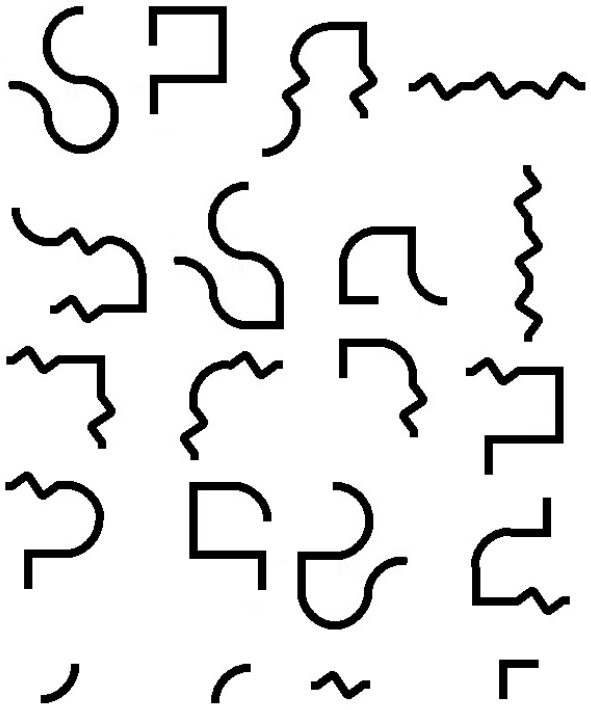


Рисунок 3.5 – Обучающие трассы

Для этапа контрольной проверки было создано 20 карт, на которых проводилась оценка качества обучения и оценка параметров вычисленной матрицы . В тестовой выборке элементов, из которых составлена трасса больше. По количеству элементов тестовые трассы отличаются от

обучающей выборки, так как каждая карта составлена из минимум шести элементов. Пример тестовых карт представлен на рисунке 3.6.

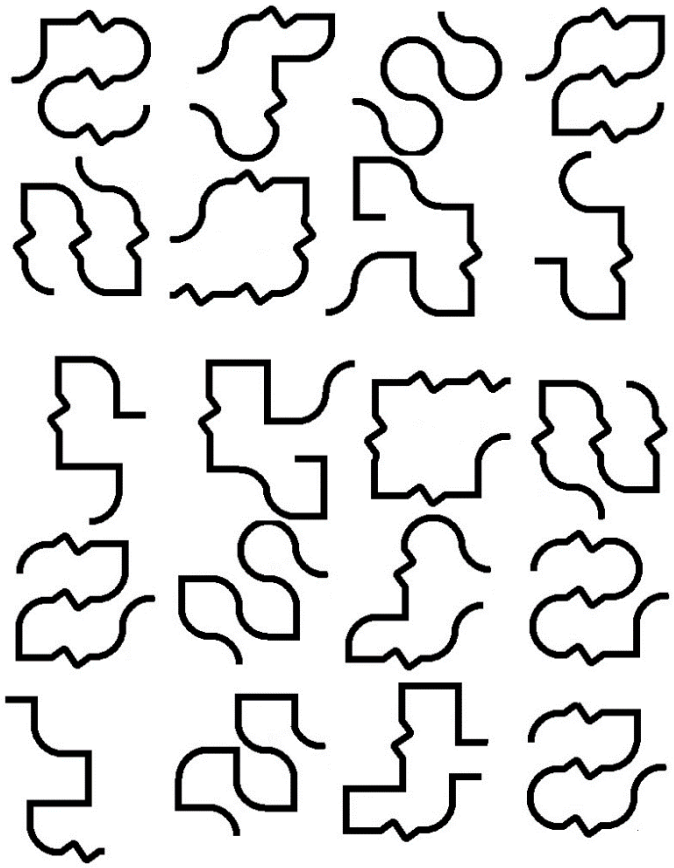


Рисунок 3.6 – Контрольные трассы

По результатам работы алгоритма обучения с подкреплением и тривиального алгоритма проводилась оценка следующих критериев.

1. Количество полностью пройденных трасс.
2. Отношение длины пройденного пути к величине трассы.
3. Максимальная удаленность центра робота от линии по всем картам.
4. Средняя удаленность центра робота от линии.

3.5 Результаты обучения

Оценка результатов проводилась через каждые 1000 эпох обучения. Ниже представлены графики полученных результатов.

На рисунке 3.7 представлен график зависимости пройденных трасс от количества эпох обучения. Обучение длилось приблизительно 2 часа. На графике изображены данные, полученные в результате обучения с подкреплением и тривиального алгоритма.

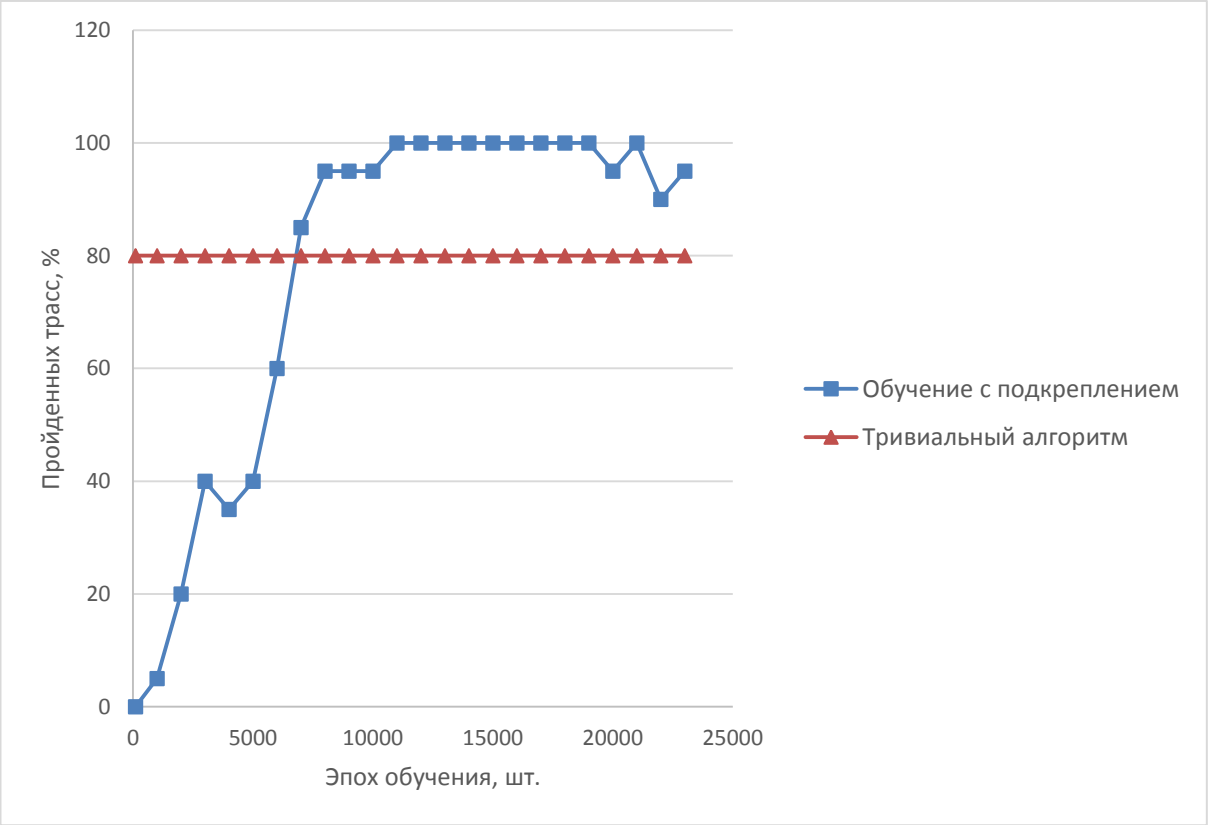


Рисунок 3.7 – Количество пройденных трасс

Анализируя рисунок 3.7, можно сделать вывод, что после 10000 эпох обучения робот проходит все 20 контрольных трасс. Увеличение числа эпох обучения после 20000 приводит к ухудшению модели. Тривиальный алгоритм движения на тесте количества пройденных трасс уступает обучению с подкреплением уже после 6000 эпох обучения.

На рисунке 3.8 представлен график отношения длины пройденного пути к длине пройденной трассы. Данные эксперимента свидетельствует о том, что с увеличением числа эпох обучения увеличивается отношения длины пройденного пути к длине пройденной трассы. В данном тесте тривиальный алгоритм уступает обучению с подкреплением после 3000 эпох обучения.

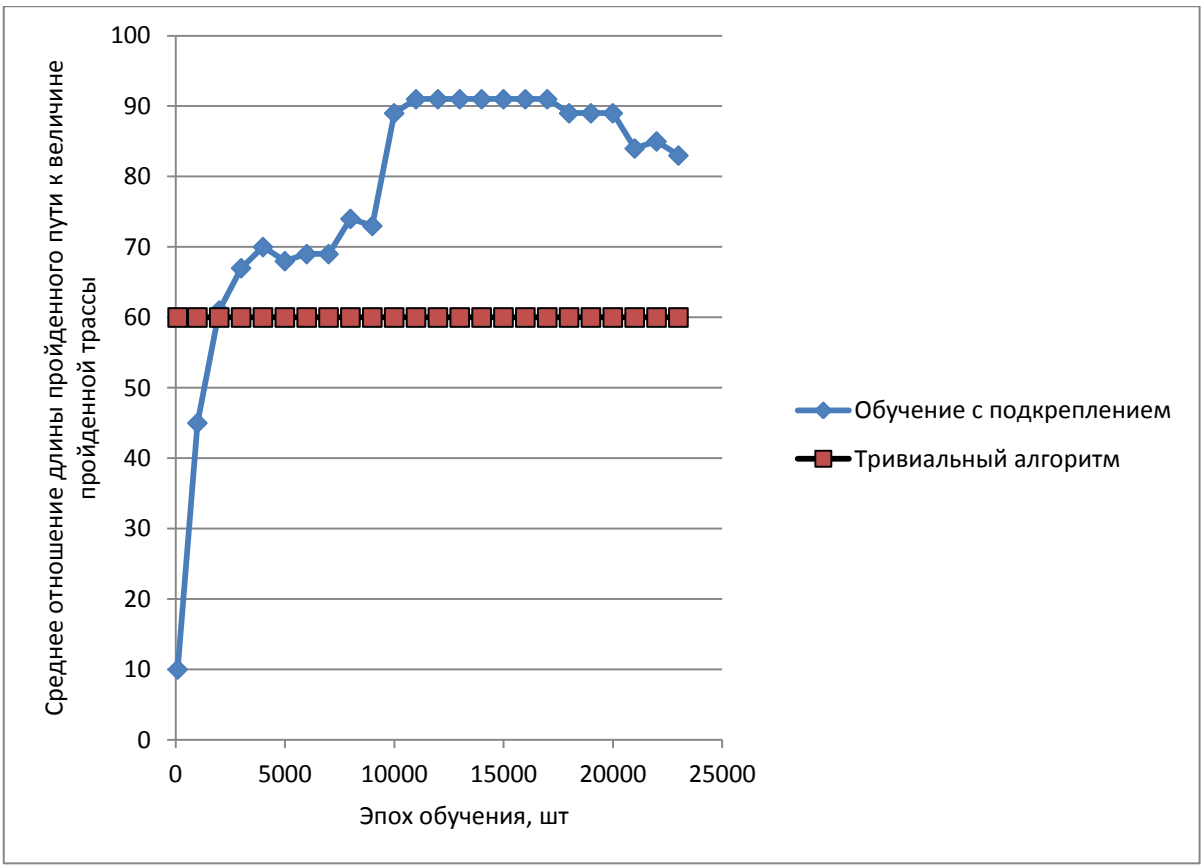


Рисунок 3.8 – Среднее отношение длины пройденного пути к длине пройденной трассы

Среднее отношение длины пройденного пути к длине пройденной трассы учитывает результаты движения агента вдоль линии до момента дисквалификации, которая может случиться из-за чрезмерного отдаления агента от центра линии либо если агент пересечет черту карты, на которой он находится. Тривиальный алгоритм движения показывает стабильные результаты, которые на небольшом количестве эпох обучения методом обучения с подкреплением показывает лучший результат.

На рисунке 3.9 представлен график зависимости максимальной удаленности агента в ходе движения по заранее намеченной линии. Максимальная удаленность от центра линии до агента вычисляется как сумма модулей разности координат, как показано на формуле (17).

, (17)

где d – расстояние от робота до центра линии,

, – текущие координаты агента,

, – координаты не посещенного центра линии.

До 10000 эпох обучения максимальная удаленность равна 25 условным единицам, это происходит так как до этого момента робот не проходит все трассы и при достижении максимального порога отдаленности агента от центра происходит завершение попытки и переход к новой обучающей трассе.

Мы можем наблюдать снижение качества модели, примерно после 20 тысяч эпох обучения, невозможность правильного обобщения может быть связана с тем, что нужной модели просто нет в используемом пространстве моделей.

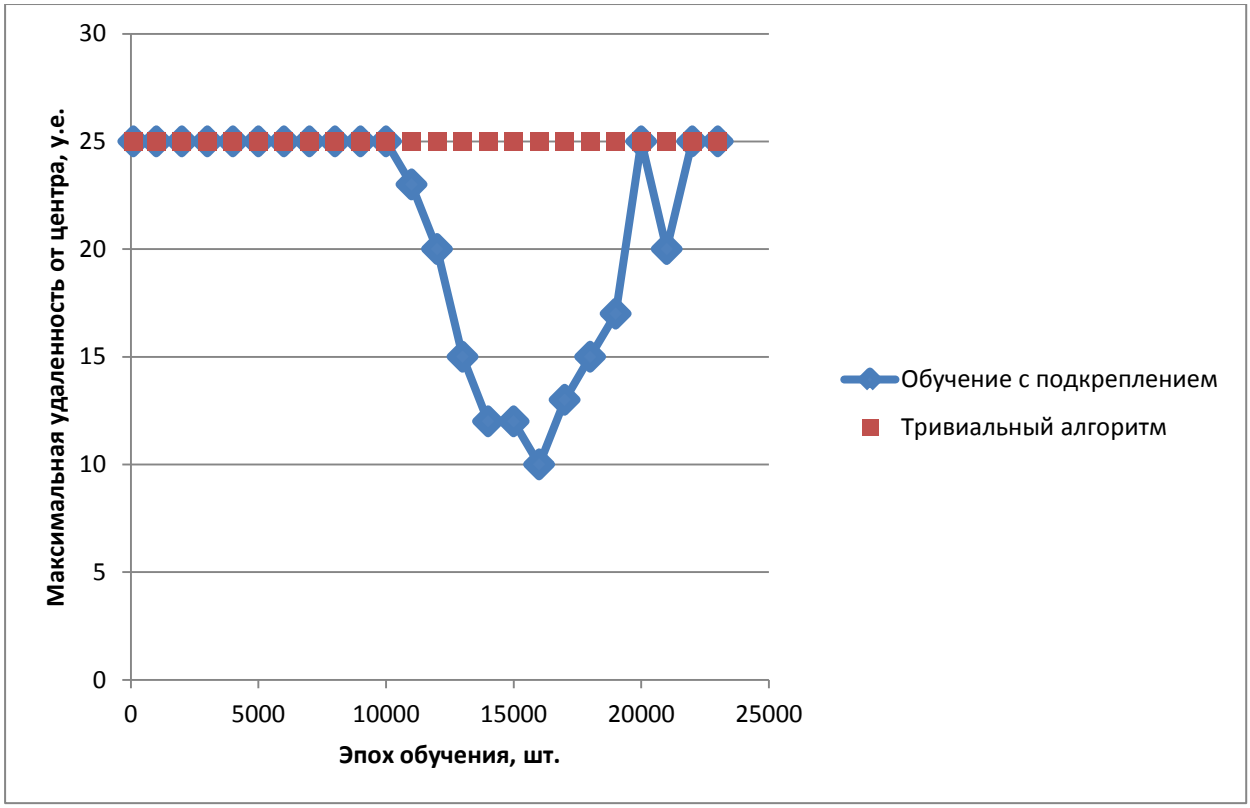


Рисунок 3.9 – Максимальная удаленность от центра

На рисунке 3.10 представлен график зависимости средней дистанции между роботом и центральном линии от количества эпох обучения.

При увеличении числа эпох обучения средняя длина пути уменьшается, однако при переходе через некоторую границу длина пути увеличивается, и качество обучения ухудшается, происходит процесс переобучения и деградации модели.

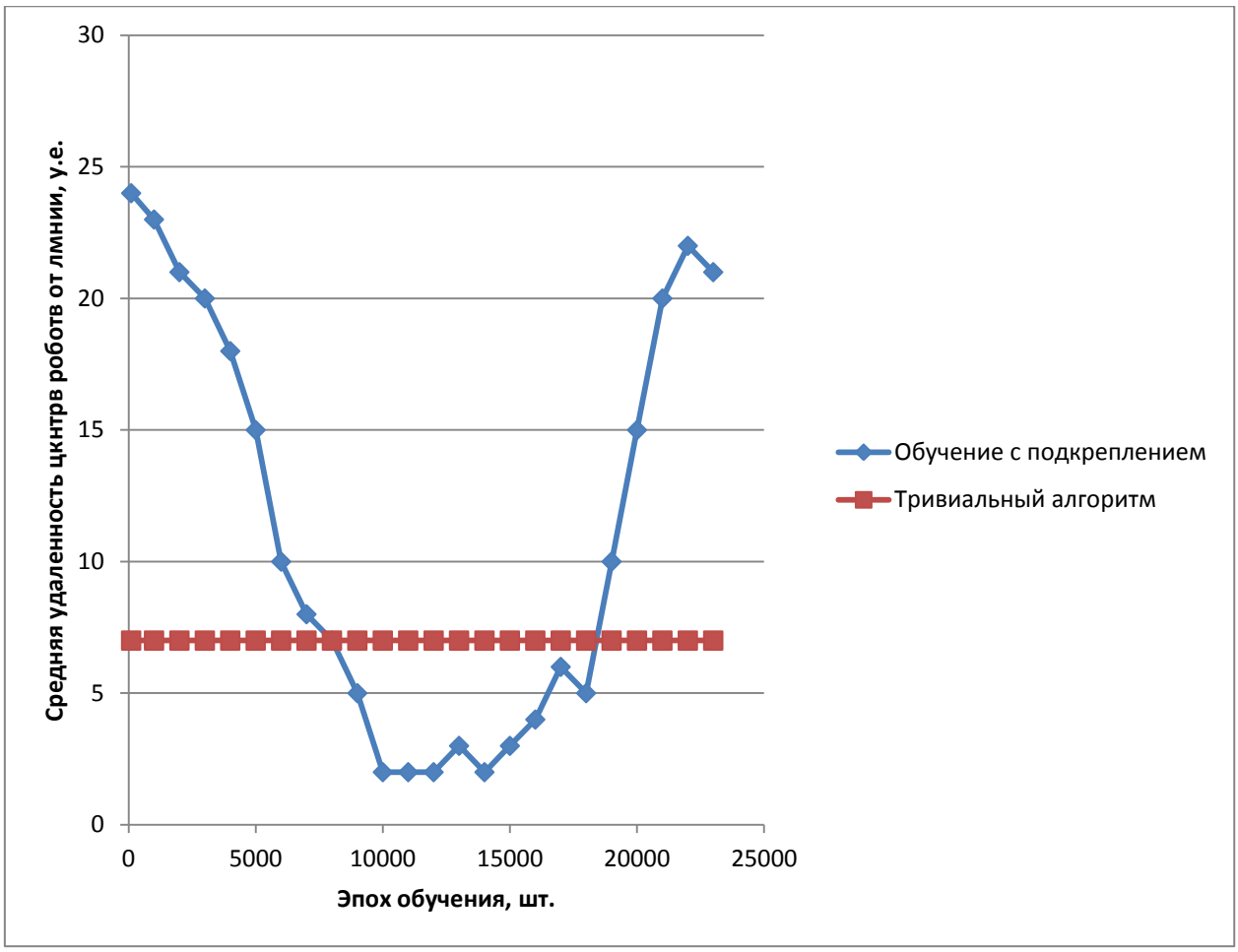


Рисунок 3.10 – Средняя удаленность центра робота от линии

По результатам обучения можно сделать вывод о том, что алгоритм обучения с подкреплением при более чем 6000 эпохах обучения показывает лучшие результаты, чем работа тривиального алгоритма, однако при меньшем количестве обучающих эпох тривиальных алгоритм показывает лучший результат.

Увеличение числа эпох более чем 10000 не рекомендуется, так происходит процесс переобучения и теряется обобщающая способность модели.

3.6 Выводы по разделу

Анализ результатов, полученных с помощью алгоритма обучения с подкреплением, показывает, что наилучшее решение достигается при более чем 10000 эпохах обучения. Однако при значительном увеличении числа

эпох обучения происходит процесс переобучения, коэффициенты сбиваются, и качество модели ухудшается, агент хорошо распознает примеры из обучающей выборки, но плохо из контрольной.

В сравнении с тривиальным алгоритмом наблюдается улучшение таких показателей как количество полностью пройденных трасс, отношение величины пройденного пути к величине трассы, максимальная удаленность робота от линии, средняя удаленность робота от линии.

Была разработана программа на языке программирования python, результатом работы программы являются найденные коэффициенты внутреннего состояния матрицы Q алгоритма обучения с подкреплением.

ЗАКЛЮЧЕНИЕ

Для решения задачи движения робота по линии с помощью алгоритма обучения с подкреплением был проведен обзор методов обучения с подкреплением, а также проанализирована предметная область движения роботов.

Была разработана модель робота и системы обучения, а также виртуальная среда выполнения и тестирования стратегий. Был реализован алгоритм обучения и предложены критерии оценки.

В ходе выполнения работы был создан фреймворк для обучения различных моделей. Выполнена настройка параметров агента – матрица Q в алгоритме обучения Q-learning, для получения наилучших результатов. Были выбраны количество эпох обучения. Разработана программа на языке высокого уровня Python. Результатами работы программы являются коэффициенты матрицы . С помощью нее были получены результаты работы на контрольных картах. Результаты работы говорят о том, что алгоритм обучения работает лучше, чем тривиальный алгоритм.

Самые лучшие результаты получены при 10000 эпох обучения агента. При меньшем количестве агент обучается недостаточно и теряет управление в некоторых ситуациях на крутых поворотах. При увеличении числа эпох агент подвергается переобучению и заучиванию тренировочных трасс, теряет обобщающую способность и показывает по некоторым тестам худшие результаты, чем тривиальный алгоритм обучения. Полученные результаты обучения можно считать приемлемыми, если сравнивать с работой тривиального алгоритма.

Важным заключением данной работы является то, что удалось получить довольно точную модель, используя минимальное количество входной информации и затратив минимальное количество средств для моделирования объекта физической природы. Имеется в виду то, что как описывалось ранее, мы используем виртуальную модель, а не натурную.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Филиппов, С.А. Уроки робототехники / С.А. Филиппов, В.А. Воробьев. – Москва: Лаборатория знаний, 2018. – 252 с.

2 Омельченко, Е. Я. Краткий обзор и перспективы применения микропроцессорной платформы Arduino / Е.Я. Омельченко, В.О. Маклаков // ЭС и К. 2013. №21. – С. 129–133.

3 Адамян, А.А. Управление шаговым двигателем на базе одноплатного компьютера Raspberry Pi 2 в среде codesysv3.5 / А.А. Адамян, В.Х Аль-Тиббби // Молодой исследователь Дона. 2018. №4 (13). – С. 19–29.

4 Intel «Обучение для будущего» [Электронный ресурс]: учебное пособие. – Электрон. дан. – Москва: 2016. – 425 с. – Режим доступа: https://e.lanbook.com/book/100605. – Загл. с экрана.

5 Неустроев, А.В. Начало программирования в Lego Mindstorms EV3 с использованием языка Java / А.В. Неустроев // Academy. 2016. №1 (4). – С. 29–53.

6 Черноножкин, В.А. Система локальной навигации для наземных мобильных роботов / В.А. Черноножкин, С.А. Половка // Научнотехнический вестник информационных технологий, механики и оптики. 2008. №57. URL: https://cyberleninka.ru/article/n/sistema-lokalnoy-navigatsii-dlya-nazemnyh-mobilnyh-robotov (дата обращения: 10.06.2019). URL: https://cyberleninka.ru/article/n/sistema-lokalnoy-navigatsii-dlya-nazemnyh-mobilnyh-robotov

7 Ачкасов, О.Р. Темпы развития робототехники в России и мире / О.Р. Ачкасов, Д.К. Синяков // Актуальные проблемы авиации и космонавтики. 2016. №12. URL: https://cyberleninka.ru/article/n/rates-of-development-of-robotics-in-russia (дата обращения: 10.06.2019).

8 Ставров, А.А. Импульсные лазерные дальномеры для оптико-локационных систем / А.А. Ставров, М.Г. Поздняков // Доклады БГУИР. 2003. №2 (2). URL: https://cyberleninka.ru/article/n/impulsnye-lazernye-dalnomery-dlya-optiko-lokatsionnyh-sistem (дата обращения: 10.06.2019).

9 Власов, С.М. Бесконтактные средства локальной ориентации роботов: учебно-методическое пособие / С.М. Власов – Санкт-Петербург: НИУ ИТМО, 2017. – 120 с.

10 Булгаков, А.Г. Промышленные роботы. Кинематика, динамика, контроль и управление: монография / А.Г. Булгаков, В.А. Воробьев. – Москва: СОЛОН-Пресс, 2008. – 245 с.

11 Регламент соревнований открытого робототехнического турнира // Электронный ресурс. – Режим доступа: http://www.railab.ru/tekushchie-reglamenty/line.html

12 Большаков, А.А. Планирование траектории движения мобильного робота / А.А. Большаков, М.Ф. Степанов // Вестник СГТУ. – 2010. – №3с. URL: https://cyberleninka.ru/article/n/planirovanie-traektorii-dvizheniya-mobilnogo-robota (дата обращения: 10.06.2019).

13 Сотникова, М.В. Алгоритм автоматического удержания колесного робота на визуально заданной линии // Вестник СПбГУ. Серия 10. Прикладная математика. Информатика. Процессы управления. – 2016. – №1. URL: https://cyberleninka.ru/article/n/algoritm-avtomaticheskogo-uderzhaniya-kolesnogo-robota-na-vizualno-zadannoy-linii (дата обращения: 10.06.2019).

14 International Robotic Competition: Робот для траектории на основе LEGO EV3 // Электронный ресурс. – Режим доступа: https://wroboto.ru/rules/RobotExample

15 Робототехника для всех: Движение по линии с двумя датчиками // Электронный ресурс. – Режим доступа: http://studrobots.ru

16 Златопольский, Д.М. Основы программирования на языке Python [Электронный ресурс]: учебник / Д.М. Златопольский. – Электрон. дан. – Москва: ДМК Пресс, 2017. – 284 с. – Режим доступа: https://e.lanbook.com/book/97359. – Загл. с экрана.

17 Коэльо, Л.П. Построение систем машинного обучения на языке Python / Л.П. Коэльо, В. Ричарт; пер. с англ. А.А. Слинкин. – Электрон. дан. – Москва: ДМК Пресс, 2016. – 302 с.

18 Профессиональный информационно-аналитический ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу данных. // Электронный ресурс. – Режим доступа: http://www.machinelearning.ru/

19 Саттон, Р.С. Обучение с подкреплением / Р.С Саттон, Э.Г. Барто – Электрон. дан. – Москва: Бином. Лаборатория знаний, 2014. – 400 с.

20 Синявский, О.Ю. Обучение с подкреплением спайковой нейронной сети в задаче управления агентом в дискретной виртуальной среде, / О.Ю. Синявский, А.И. Корбин // Нелинейная динам., 2011, том 7, номер 4, – С. 859–875

21 Окулов, С.М. Динамическое программирование [Электронный ресурс]: учебное пособие / С.М. Окулов, О.А. Пестов. – Электрон. дан. – Москва: Издательство «Лаборатория знаний», 2015. – 299 с. – Режим доступа: https://e.lanbook.com/book/66114. – Загл. с экрана.

22 Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / П. Флах. – Электрон. дан. – Москва: ДМК Пресс, 2015. – 400 с.

23 Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн – М.: Вильямс, 2005. – 1296 с.

ПРИЛОЖЕНИЕ 1

Листинг программы

```
# -*- coding: utf-8 -*-
import random

from abc import ABC, ABCMeta, abstractmethod
import numpy as np


class AgentA(ABC):
    __metaclass__ = ABCMeta

    @abstractmethod
    def process(self, states, reward):
        """ Установка нового состояния
        и награды за предыдущее действие"""

    @abstractmethod
    def get_chosen_action_number(self):
        """ Получение выбранного действия"""

    @abstractmethod
    def get_params(self):
        """ Получение найденных коэффициентов"""


class Agent(AgentA):
    _last_reward = None
    _last_state = None
    _last_action_number = None
    _total_reward = None
    _number_of_actions = None
    _Q = None

    _entry = None

    def __init__(self, number_of_actions, Q):
        self._last_reward = 0
        self._total_reward = 0
        self._number_of_actions = number_of_actions
        self._last_state = -1
        self._last_action_number = 0
        self._Q = Q

    def process(self, state, reward):
```

```
                if self._Q.get(self._last_state) is None and self._last_state != -1:
                    self._Q[self._last_state] = [0] * self._number_of_actions

                if self._Q.get(state) is None:
                    self._Q[state] = np.zeros(self._number_of_actions)

                if self._last_state != -1:
                    self._Q[self._last_state][self._last_action_number] += reward

                self._last_state = state

                self._last_reward = reward
                self._total_reward += int(reward)
                idx = self._Q[state].argmax()
                self._last_action_number = idx

                # if random.random() < 0.3:
                #                     self._last_action_number    =    random.randint(0,
self._number_of_actions - 1)

            def set_Q(self, Q):
                self._Q = Q

            def get_chosen_action_number(self):
                """ """
                return self._last_action_number

            def get_params(self):
                """ """
                return self._Q

            def get_total_reward(self):
                return self._total_reward

            def reset_reward(self):
                self._total_reward = 0

        # environment.py
        # -*- coding: utf-8 -*-
        import random

        import numpy as np
        from queue import Queue
        from PIL import Image
        from abc import ABC, ABCMeta, abstractmethod
        import os
```

```
import datetime

from MotionMap import MotionMap


class Env(ABC):
    __metaclass__ = ABCMeta

    @abstractmethod
    def get_reward(self):
        """ Награда"""

    @abstractmethod
    def get_states(self):
        """ Состояние световых датчиков"""

    @abstractmethod
    def process_action(self, action_id):
        """ Обработка действия"""

    @abstractmethod
    def get_number_of_actions(self):
        """ Количество допустимых действий"""


class Environment(Env):
    _RED = (255, 0, 0)
    _GREEN = (0, 255, 0)
    _BLUE = (0, 0, 255)
    _WHITE = (255, 255, 255)
    _directory = 'result' + os.sep + str(datetime.datetime.now()).replace(':',
'-')
    _epoch = 0
    motion_map: MotionMap = None
    _last_reward = 0
    _last_center_point = 0

    def __init__(self,
                 start_position: 'Вектор столбец. Начальная позиция робота' =
None,
                 start_theta: 'Угол поворота, куда смотрит агент, в градусах' =
0,
                 step: 'Длина шага' = 1,
                 theta: 'Угол поворота при совершении действия, в градусах' =
15,
                 number_of_last_states: 'Кол-во хранимых состояний датчиков' =
3,
```

```
                dist_btw_sensors: 'Расстояние между сенсорами' = 10
                ):
            Environment._epoch += 1
            self._last_center_point = 0
            if Environment.motion_map is None:
                Environment.motion_map = MotionMap()
            self._dist_btw_sensors = dist_btw_sensors

            self._theta = theta
            self._number_of_last_states = number_of_last_states
            self._last_sensors_state = [0, 0] * number_of_last_states

            self._is_evaluated_sensors_state = False

            self._step = float(step)

            if start_position is None:
                start_positions = Environment.motion_map.get_start_positions()
                start_position          =          start_positions[random.randint(0,
len(start_positions) - 1)]

                # start_position[0] += random.randint(-1, 1)
                # start_position[1] += random.randint(-1, 1)

            start_position = [
                [float(start_position[0])],
                [float(start_position[1])]
            ]

            self._start_point = np.array(start_position)
            end_position = start_position
            end_position[0][0] += step
            end_position = np.array(end_position)
            end_position   =   Environment.rotate(self._start_point,   end_position,
start_theta)
            self._end_point = np.array(end_position)

            self._sensors = np.zeros(2)

            #self._load_map(path_to_map)
            self._map = Environment.motion_map.get_map()
            self._center_line = Environment.motion_map.get_center_line()
            self._predicts = Environment.motion_map.get_predicts()
            self._img_map = Environment.motion_map.get_img_map().copy()
            self._pixels = self._img_map.load()

            self._actions = [self._action1, self._action2, self._action3]
```

```
            self._total = 0

        def get_states(self):
            """ Состояние световых датчиков"""
            if self._is_evaluated_sensors_state:
                return tuple(self._last_sensors_state)

            self._eval_sensor_state()
            # self._is_evaluated_sensors_state = True

            for i in range(2, len(self._last_sensors_state), 1):
                self._last_sensors_state[i - 2] = self._last_sensors_state[i]

            self._last_sensors_state[2 * self._number_of_last_states - 2] =
self._sensors[0]
            self._last_sensors_state[2 * self._number_of_last_states - 1] =
self._sensors[1]

            return tuple(self._last_sensors_state)

        def get_number_of_last_states(self):
            return self._number_of_last_states

        def get_number_of_actions(self):
            return len(self._actions)

        def process_action(self, action_id):
            self._actions[action_id]()
            self._is_evaluated_sensors_state = False
            self._track_agent()

        def log(self):
            print('position: [' + str(self._start_point[0][0]) + ', ' +
str(self._start_point[1][0]) + ']')
            print('sensors: ' + str(self.get_states()))
            print('last reward: ' + str(self._last_reward))
            print()

        def get_reward(self):
            """ Ценность текущего состояния."""
            self._last_reward = self._evaluate_reward()
            return self._last_reward

        def _evaluate_reward_2(self):
            left_sensor_position = self._eval_left_sensor_position()
            right_sensor_position = self._eval_right_sensor_position()
```

```
            x_left = int(left_sensor_position[0][0])
            y_left = int(left_sensor_position[1][0])
            left_val = self._predicts[x_left][y_left]

            x_left = int(right_sensor_position[0][0])
            y_left = int(right_sensor_position[1][0])
            right_val = self._predicts[x_left][y_left]
            if left_val == 1 and right_val == 1:
                return 2

            return -(abs(left_val - 1) + abs(right_val - 1))

        def _evaluate_reward(self):
            """ Оценка награды, расчитывается как расстояние до ближайшей линии,
используя волновой алгоритм(BFS)"""
            queue = Queue()
            visited = set()
            start_point             =             [[int(self._start_point[0][0])],
[int(self._start_point[1][0])]]  # [[12], [43]]
            queue.put(start_point)
            reward = -100
            dist_btw_center_and_robot = 100
            while not queue.empty():
                cur_position = queue.get()
                queue.task_done()

                x_pos = cur_position[0][0]
                y_pos = cur_position[1][0]

                if not self._is_valid_point_position(cur_position):
                    continue

                if (x_pos, y_pos) in visited:
                    continue

                visited.add((x_pos, y_pos))

                if self._center_line[x_pos][y_pos] > 0:

                #if self._center_line[x_pos][y_pos] > self._last_center_point:
                    dist_btw_center_and_robot  =  abs(x_pos  -  start_point[0][0])  +
abs(y_pos - start_point[1][0])
                    if abs(dist_btw_center_and_robot) <= 6:
                        self._last_center_point = self._center_line[x_pos][y_pos]
                    break
```

```
                    directions = [[0, 1], [1, 0], [-1, 0], [0, -1],]# [-1, -1], [-1, 1],
[1, 1], [1, -1]]
                    for direction in directions:
                        if (x_pos + direction[0], y_pos + direction[1]) in visited:
                            continue
                        next_pos = [[x_pos + direction[0]], [y_pos + direction[1]]]
                        queue.put(next_pos)
                if dist_btw_center_and_robot == 0:
                    return 12
                if dist_btw_center_and_robot <= 2:
                    return dist_btw_center_and_robot
                if dist_btw_center_and_robot > 7:
                    return -100
                return -1

            def show(self):
                """ Открытие картинки с треком движения"""
                # self._img_map.show()

                if not os.path.exists(Environment._directory):
                    os.makedirs(Environment._directory)
                self._img_map.save(Environment._directory                                    +
"/{0}.png".format(Environment._epoch))

            def _action1(self):
                """ Движение вперед."""
                d = self._end_point - self._start_point
                self._start_point += d
                self._end_point += d

            def _action_t(self):
                """ Движение вперед."""
                d = self._end_point - self._start_point
                self._start_point += d / 1
                self._end_point += d / 1

            def _action2(self):
                """ Поворот по часовой на фиксированный угол и движение по направлению
вектора."""
                self._end_point = Environment.rotate(self._start_point, self._end_point,
-self._theta)
                self._action_t()

            def _action3(self):
                """  Поворот против часовой на фиксированный угол и движение по
направлению вектора."""
```

```
            self._end_point = Environment.rotate(self._start_point, self._end_point,
self._theta)
            self._action_t()

        @staticmethod
        def rotate(point_a, point_b, theta):
            """ Поворот point_b относительно point_a на угол theta, возврат новые
координаты. """
            d = point_b - point_a  # Перенесли центр координатных осей
            d = np.dot(Environment._get_rot_matrix(theta), d)   # Совершили поворот
против часовой
            d += point_a  # Вернули центр оси
            return d

        def _eval_sensor_state(self):
            """ Вычисление состояния световых датчиков. """
            left_sensor_position = self._eval_left_sensor_position()
            right_sensor_position = self._eval_right_sensor_position()

            if self._is_valid_point_position(left_sensor_position):
                x_left = int(left_sensor_position[0][0])
                y_left = int(left_sensor_position[1][0])
                self._sensors[0] = self._predicts[x_left][y_left]
                #self._sensors[0] = self._map[y_left][x_left]
            else:
                self._sensors[0] = 0

            if self._is_valid_point_position(right_sensor_position):
                x_right = int(right_sensor_position[0][0])
                y_right = int(right_sensor_position[1][0])
                self._sensors[1] = self._predicts[x_right][y_right]
                #self._sensors[1] = self._map[y_right][x_right]
            else:
                self._sensors[1] = 0
            # self._sensors[0] = 1 if self._sensors[0] > 0.8 else 0
            # self._sensors[1] = 1 if self._sensors[1] > 0.8 else 0
            self._is_evaluated_sensors_state = True

        def _is_valid_point_position(self, point: 'Вектор столбец вида [[2\n  1]]'):
            """ Проверка не вышли за границу. """
            if point[0][0] < 0 or point[1][0] < 0:
                return False

            if    point[0][0]    >=    self._map.shape[0]    or    point[1][0]    >=
self._map.shape[1]:
                return False
            return True
```

```
        def _eval_left_sensor_position(self):
            left_point = Environment.rotate(self._start_point, self._end_point, 90)
            k = self._dist_btw_sensors / 2 / self._step
            left_point -= self._start_point
            left_point *= k
            left_point += self._start_point
            return left_point

        def _eval_right_sensor_position(self):
            right_point = Environment.rotate(self._start_point, self._end_point, -
90)
            k = self._dist_btw_sensors / 2 / self._step
            right_point -= self._start_point
            right_point *= k
            right_point += self._start_point
            return right_point

        def _track_agent(self, color=_GREEN):
            """ Пометка позиции робота"""
            if not self._is_valid_point_position(self._start_point):
                return
            self._mark_position(self._start_point, self._GREEN)
            self._mark_position(self._eval_left_sensor_position(), self._RED)
            self._mark_position(self._eval_right_sensor_position(), self._RED)

        def _mark_position(self, position, color=_GREEN):
            if not self._is_valid_point_position(position):
                return
            x = int(position[0][0])
            y = int(position[1][0])
            self._pixels[y, x] = color

        @staticmethod
        def _get_rot_matrix(theta_degree):
            """ Матрица поворота против часовой. """
            theta = np.radians(theta_degree)
            return np.array(((np.cos(theta), -np.sin(theta)),
                             (np.sin(theta), np.cos(theta))))
    # MptionMap.py
    from queue import Queue

    import numpy as np
    from PIL import Image


    class MotionMap:
```

```
        _predicts = None  # Предпросчет
        _map = None  # Карта с линией
        _center_line = None  # Карта с центральной линией
        _start_positions = []
        _dist_btw_sensors = None
        _img_map = None

        def __init__(self, path_to_map: 'Путь до картинки с картой'='maps/test.jpg',
                     dist_btw_sensors=8):
            self._dist_btw_sensors = dist_btw_sensors
            self._load_map(path_to_map)
            self._find_center_line()
            self._eval_predicts()
            self._eval_start_positions()

        def get_start_positions(self):
            return self._start_positions

        def get_map(self):
            return self._map

        def get_center_line(self):
            return self._center_line

        def get_predicts(self):
            return self._predicts

        def get_img_map(self):
            return self._img_map

        def _eval_start_positions(self):
            for i in range(self._center_line.shape[0]):
                for j in range(self._center_line.shape[1]):
                    if self._center_line[i][j] > 0.5 and
self._is_valid_point_position([[i], [j + self._dist_btw_sensors]]):
                        is_valid = True
                        for d in range(self._dist_btw_sensors):
                            if self._center_line[i][j + d] < 0.5:
                                is_valid = False
                                break
                        if is_valid:
                            self._start_positions.append([i, j])

        def _is_valid_point_position(self, point: 'Вектор столбец вида [[2\n  1]]'):
            """ Проверка не вышли за границу. """
            if point[0][0] < 0 or point[1][0] < 0:
                return False
```

```
            if point[0][0] >= self._map.shape[0] or point[1][0] >=
self._map.shape[1]:
                return False
            return True

        def _load_map(self, path):
            self._img_map = Image.open(path)
            # self._pixels = self._img_map.load()
            # im.save('greyscale.png')
            states = np.array(self._img_map.convert('L'))
            mx = states.max()
            states = states / mx
            for x in range(states.shape[0]):
                for y in range(states.shape[1]):
                    if states[x][y] > 0.5:
                        states[x][y] = 1
                    else:
                        states[x][y] = 0

            self._map = states

        def _eval_predicts(self):
            self._predicts = np.zeros(self._map.shape)

            for x in range(self._map.shape[0]):
                for y in range(self._map.shape[1]):
                    sum = 0
                    count = 0
                    for i in range(-int(self._dist_btw_sensors / 2) + 1,
int(self._dist_btw_sensors / 2) - 1):
                        for j in range(-int(self._dist_btw_sensors / 2) + 1,
int(self._dist_btw_sensors / 2) - 1):
                            if i ** 2 + j ** 2 <= self._dist_btw_sensors ** 2:
                                count += 1
                                if self._is_valid_point_position([[x+i], [y+j]]):
                                    sum += self._map[x + i][y + j]
                    self._predicts[x][y] = round(sum / count * 6)

        def _find_center_line(self):
            self._center_line = np.zeros(self._map.shape)
            for x in range(int(self._dist_btw_sensors / 2), self._map.shape[0] -
int(self._dist_btw_sensors / 2)):
                for y in range(int(self._dist_btw_sensors / 2), self._map.shape[1] -
int(self._dist_btw_sensors / 2)):
                    is_valid = True
```

```
                    for i in range(-int(self._dist_btw_sensors / 2) + 1,
int(self._dist_btw_sensors / 2) - 1):
                        for j in range(-int(self._dist_btw_sensors / 2) + 1,
int(self._dist_btw_sensors / 2) - 1):
                            if i ** 2 + j ** 2 <= self._dist_btw_sensors ** 2:
                                if self._map[x + i][y + j] < 0.5:
                                    is_valid = False
                    if is_valid:
                        self._center_line[x][y] = 1

            queue = Queue()
            visited = set()
            start_point = [[[124], [16]], 1]
            queue.put(start_point)
            while not queue.empty():
                cur_position = queue.get()
                queue.task_done()

                x_pos = cur_position[0][0][0]
                y_pos = cur_position[0][1][0]
                step = cur_position[1]

                if not self._is_valid_point_position(cur_position[0]):
                    continue

                if (x_pos, y_pos) in visited:
                    continue

                visited.add((x_pos, y_pos))

                if self._center_line[x_pos][y_pos] == 1:
                    self._center_line[x_pos][y_pos] = step
                else:
                    continue

                directions = [[0, 1], [1, 0], [1, 1], [-1, 0], [-1, -1], [0, -1], [-
1, 1], [1, -1]]
                for direction in directions:
                    if (x_pos + direction[0], y_pos + direction[1]) in visited:
                        continue
                    next_pos = [[[x_pos + direction[0]], [y_pos + direction[1]]],
step + 1]
                    queue.put(next_pos)
```

director.py

```
# -*- coding: utf-8 -*-
# Посредник между средой и агентом.
```

```
from environment import Environment
from agent import Agent
from MotionMap import MotionMap
import numpy as np
import random


def run():
    agent = Agent(number_of_actions=3, Q={})
    # Запустим N эпох
    N = 10000
    show_every_n = 1
    for epoch in range(N):
        if epoch % show_every_n == 0:
            env = Environment(start_position=[124, 16],
                              number_of_last_states=1,
                              theta=5,
                              step=1,
                              start_theta=90,
                              dist_btw_sensors=7)
        else:
            env = Environment(number_of_last_states=1,
                              theta=7,
                              step=1,
                              start_theta=90,
                              dist_btw_sensors=7)
        for i in range(700):
            states, reward = env.get_states(), env.get_reward()
            agent.process(states, reward)

            action_id = agent.get_chosen_action_number()

            env.process_action(action_id)
            if reward < -16:
                print(i, reward)
                break
        agent.reset_reward()
        if epoch % show_every_n == 0:
            env.show()
    new_q = agent.get_params()
    for i in new_q:
        print(i, ':', new_q[i])
```