Санкт–Петербургский государственный университет

КАЛИТИН Александр Олегович

Выпускная квалификационная работа

Применение методов обучения с подкреплением для разработки стратегии игры в футбол роботов

Уровень образования: **бакалавриат**

Направление: **01.03.02 «Прикладная математика и информатика»,**

Основная образовательная программа: **СВ.5004.2017 «Прикладная математика и информатика»**

Профиль «Управление и обработка информации в кибернетических и робототехнических системах»

Научный руководитель: доцент,
кафедра теоретической кибернетики,
к.ф. - м.н. Липкович Михаил

Рецензент: научный сотрудник,
ФГБУ науки Институт проблем
машиноведения Российской академии наук
к.т.н., Томчин Дмитрий Александрович

Санкт-Петербург
2021 г.

SAINT-PETERSBURG STATE UNIVERSITY

KALITIN Aleksandr Olegovich

Graduation Thesis

Strategy design for robofootball using reinforcement learning

Bachelor's Degree

01.03.02 «Applied Mathematics and Computer Science»,

«Information management and processing in cybernetic and robotic systems»

Scientific supervisor: Associate Professor ,
department of Theoretical Cybernetics,
Candidate of Mathematics and Physics
Lipkovich Michael

Reviewer: Researcher of
Institute of Problems of Mechanical Engineering
Russian Academy of Sciences
Candidate of Technical Sciences
Tomchin Dmitry Alexandrovich

Saint-Petersburg
2021

Содержание

Введение

Робофутбол

С 1996 г. для содействия научным исследованиям в областях робототехники, искусственного интеллекта, технического зрения, навигации, группового взаимодействия роботов и мехатронных устройств начал проводиться чемпионат мира по робофутболу «RoboCup»[10]. За это время в них принимало участие множество университетов со всего мира. В рамках турнира существуют и иные виды соревнований, не связанных с робофутболом — танцы роботов, бытовые роботы, роботы-спасатели, промышленные роботы и другие. Главной целью соревнований ставится создание к середине 21-го века команды полностью автономных человекоподобных роботов-футболистов, подобных изображенным на Рис. 1 , которые способны выиграть футбольный матч, соблюдая правила FIFA у победителя Чемпионата мира по футболу среди людей.

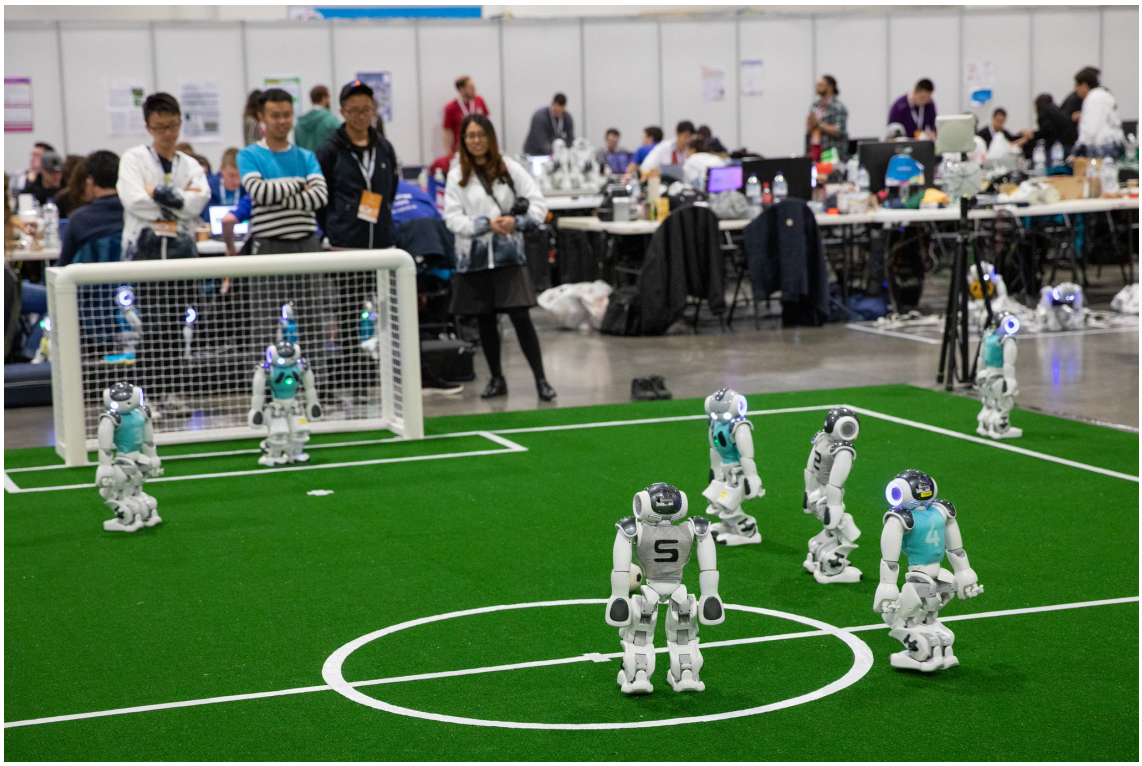

Рис. 1: Матч между автономными человекоподобными роботами.

Рассмотрим соревнования по робофутболу подробнее. Всего существует несколько лиг: RoboCupSoccer Humanoid League – состязание авто-

номных человекоподобных роботов футболистов, RoboCupSoccerStandard Platform League (ранее Four Legged League) – лига где команды состоят из одинаковых роботов, которые работают полностью автономно без какого-либо контроля со стороны человека или компьютера (централизованного управления), RoboCupSoccer Middle Size League – состязание роботов футболистов средних размеров, RoboCupSoccer Simulation League – одна из старейших лиг, в которой соревнуются компьютерные программы, моделирующие процесс игры в футбол на виртуальном компьютерном поле независимо движущимися программными игроками, и последняя лига это Small Size League (SSL) [11], которую отличает наличие единого вычислительного центра, управляющего всей командой, роботы здесь перемещаются с помощью четырехколесной омниплатформы, имеют два ударных устройства (кикера) один используется для удара вдоль поверхности поля, а другой для навесов. Крышки роботов маркируются цветными кругами, благодаря которым система компьютерного зрения распознает игроков и отличает друг от друга, а также определяет их положение на поле. Далее речь будет идти о роботах из SSL, представление об устройстве которых можно получить, изучив Рис. 2 ниже, на котором изображен матч чемпионата мира RoboCup в лиге SSL.

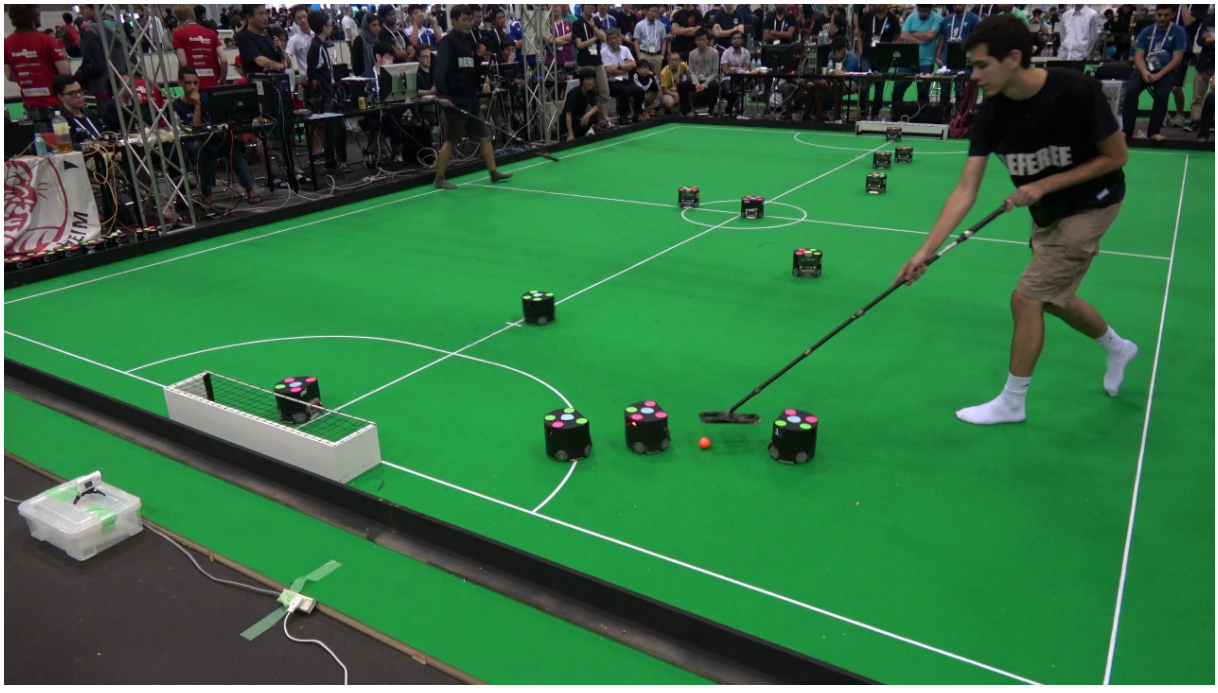

Рис. 2: Матч между роботами в лиге SSL.

Российские исследования в области робофутбола

В 2000-х годах Российские ученые интересовались данной тематикой, но полноценного участия в соревнованиях отечественные роботы еще никогда не принимали. Следует отметить, что в СПбГУ также велись разработки в этом направлении [13, 14]. С недавних пор на базе сотрудничества университета и Президентского физико-математического лицея № 239 существует команда «URoboRus» [12, 18] , впервые квалифицировавшаяся в финальную часть турнира в 2019 году, и которая планирует принимать участие в финальной части соревнований в 2021 году.

Составляющие части организации работы команды по робофутболу

Формирование команды роботов-футболистов делится на несколько основных составляющих: инженерную, которая предполагает разработку самих роботов и их обслуживание, написание программного обеспечения и его обновление, а также создание алгоритмов с их последующей реализацией. Каждый из этих аспектов важен для слаженной работы команды, однако опыт показывает, что большинство команд одинаково успешно справляются с первыми двумя задачами, а победителя от остальных отличает качество низкоуровневых и высокоуровневых алгоритмов управления. Точно так же в в своей статье рассуждали победители последнего чемпионата 2019 года, ZJUNlict(Zhejiang University, P.R.China) [15]. Именно удачную стратегию они считают основной причиной своей победы. Далее в этой работе будут рассмотрены алгоритмы решения некоторых задач робофутбола с помощью методов машинного обучения.

Машинное обучение

Машинное обучение – это класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач [16]. Под классическим машинным обучением понимается обучение с учителем (по размеченным данным) и без учителя(не размеченные данные). Однако классиче-

ские методы машинного обучения плохо применимы для задач робофутбола, так как мы не имеем изначальных данных, особенно размеченных,если собирать их достаточно проблематично, то разметка этих данных является еще более проблематичной. Помимо этого робот действует в изменяемой среде и ему нужно постоянно адаптироваться под условия окружающей среды, что затрудняет использование классических алгоритмов машинного обучения. Хорошим способом решения данной задачи является обучение с подкреплением [6]. Обучение с подкреплением (англ. reinforcement learning) – один из способов машинного обучения, в ходе которого испытуемая система (агент) обучается, взаимодействуя с некоторой окружающей средой.

Обучение с подкреплениями и Q-learning

Формально простая модель обучения с подкреплением состоит из множества состояний среды S, множества действий A и множества вещественнозначных наград R. Также в обучении с подкреплением существует агент, который взаимодействует с окружающей средой , предпринимая действия . Окружающая среда дает награду за эти действия, а агент продолжает их предпринимать. Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию (политику) $\pi : S \to A$, которая максимизирует величину $\bar{R} = r_0 + r_1 + \cdots + r_n$, где r_i представляет из себя награду за соответствующее действие, а процесс имеет терминальное состояние, в противном случае сумма превращается в ряд.

Одним из видов обучения с подкреплением являются Q-learning и Deep Q-learning, рассмотрим их более подробно. Q-learning – один из методов обучения с подкреплением. Его суть заключается в том, что на основе получаемого от среды вознаграждения агент формирует функцию полезности Q, что впоследствии дает ему возможность уже не случайно выбирать стратегию поведения, а учитывать опыт предыдущего взаимодействия со средой. Одно из преимуществ Q-learning – то, что оно в состоянии сравнить ожидаемую полезность доступных действий, не формируя напрямую мо-

дель окружающейсреды. Формально Q-функция вычисляется следующим образом(из Уравнения Беллмана [7]):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \qquad (1)$$

где s_t – состояние агента в момент времени t, a_t – выбранное действие для состояния s_t, α – шаг обучения, r_{t+1} – награда за действие a_t в состоянии s_t, λ – коэффициент скидки, а максимум берется по всем возможным действиям в состоянии s_{t+1}.

Нейронные сети и Deep Q-learning

Для дальнейшего рассмотрения методов обучения с подкреплением необходимо ввести понятие нейронной сети. Искусственная нейронная сеть представляет собой упрощенную математическую модель биологической нейронной сети в виде искусственных нейронов, взаимодействующих между собой по определенным правилам. Сетями прямого распространения называются такие искусственные нейронные сети, у которых сигнал распространяется строго от входного слоя к выходному [7]. Нейронная сеть называется многослойной, если содержит более одного слоя нейронов. Таком образом, искусственная многослойная нейронная сеть прямого распространения представляет из себя последовательное умножение входного слоя, а затем промежуточного результата на матрицы весовых коэффициэнтов нейронной сети, являющихся параметрами обучения, и применение к результату активационной функции. Обучением нейронной сети называется поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется в необходимый нам выходной, минимизирующий заранее заданную функцию потерь, зависящую от параметров нейронной сети. Обучение многослойной нейронной сети осуществляется с помощью применения алгоритма градиентного спуска для весов последнего слоя нейронной сети и алгоритма обратного распростронения ошибки [7] для остальных обучаемых параметров (весов) нейрнной сети.

Другой версией обучения с подкреплением является Deep Q-learning. Его суть заключается в том, чтобы вместо построения Q-таблицы и посте-

пенного ее заполнения пытаться аппроксимировать с помощью нейронной сети значение Q-функции, выполняющей ту же роль. На вход сети подаются значения состояния, а на выход получаем вектор значений Q-функции для каждого действия. Формула расчета будет иметь похожий на (1) вид, однако шаг обучения будет уже заложен в алгоритм обучения нейронной сети и формула примет следующий вид :

$$Q(s_t, a_t) \leftarrow r_{t+1} + \lambda \max_a Q(s_{t+1}, a), \tag{2}$$

где s_t – состояние агента в момент времени t, a_t – выбранное действие для состояния s_t, r_{t+1} – награда за действие a_t в состоянии s_t, λ – коэффициент скидки, а максимум берется по всем возможным действиям в состоянии s_{t+1}.

Симулятор

Для моделирования среды робофутбола в процессе обучения был использован симулятор «grSim» [2], разработанный организаторами Чемпионат мира по футболу среди роботов «RoboCup». Ознакомиться с интерфейсом симулятора можно на Рис. 3.

Рис. 3: Интерфейс симулятора grSim.

Постановка задачи

Для начала работы необходимо разработать клиент для взаимодействия с симулятором на языке программирования *Python*. Он должен реализовать все базовые возможности симулятора : постановку мяча в заданную точку, постановку мяча в заданную точку, возвращение значений состояния робота и мяча, а также выполнение основных действий, а именно удара по мячу, использование спиннера и присваивание значений угловой и линейной скорости каждому роботу.

Для начала разработки в области обучения была поставлена цель решения простых задач, возникающих в робофутболе с последующей перспективой перехода к более сложным. Также необходимо было проверить применимость рассматриваемых методов для решения задач робофутбола.

Постановка решаемой задачи: На штрафной линии ворот соперника случайным образом ставится робот с мячом и включенным спиннером со случайным направлением кикера (приспособление для нанесения удара по мячу) и необходимо последовательно совершать поворот направо, поворот налево или нанесение удара так, чтобы в итоге забить гол в ворота, препятствия на траектории удара отсутствуют.

Ставится цель обучить робота решать данную задачу в ста процентах случаев.

Обзор литературы

Для выполнения рабыты была изучена литература, посвяшенная фреймворку *Pytorch* [1] языка программирования *Python*, симуляторам среды робофутбола для выбора наиболее удачного варианта, методам машинного обучения, нейронным сетям и обучению с подкреплениями в частности. Рассмотрим пригодившиеся для решения поставленной задачи источники.

Для понимания возможностей и правил использования фреймворка *Pytorch* была изучена инструкция разработчиков, выложенная на их сайте [5], а также соответствующие разделы книги «Знакомство с PyTorch» [1]. В частности в данных источниках можно найти более подробную информацию про установку *PyTorch*, тензоры, оспользуемые в данном фреймворке,

теорию и устройство нейронных сетей, разъяснение таких терминов, как функция активации, градиентный спуск, функция потерь, обратное распространение ошибки и оптимизатор, в частности Adam и его варианты.

Про используемый симулятор можно прочитать подробную информацию, выложенную вместе с кодом в открытом доступе на GitHub репозитории разработчиков[2]. Также были рассмотрены альтернативы симулятору «grSim»[3, 4], но было решено их не использовать, так как они показались менее удобными и являются менее распространенными и используемыми в области.

Дла начала была изученна литература посвященная методам обучения с подкреплением, в частности Q-learning[6] и Deep Q-learning [7], для данной цели использовались соответствующие главы учебников «Обучение с подкреплением» [6] и «Глубокое обучение, погружение в мир нейронных сетей» [7].

Также были изученны статьи посвященные разработкам стратегии игры в футбол роботов без применения методов машинного обучения, нпример, в статье «Команда роботов-футболистов для соревнований RoboCup в лиге SSL: система и алгоритмы» [19] авторы говорят о возможных методах решения задач управления для роботов-футболистов, таких как движение в точку, поворот на точку, движение по окружности, движение с объездом препятствий, прием паса, поведение вратаря, удар в заданную точку, а также модель игры двух полевых игроков против вратаря. После чего были разобраны попытки применения методов обучения с подкреплением именно для решения различных задач робофутбола [8, 9], которые были предприняты для реальной игры и симуляторной лиги соответственно и подтвердили наличие интереса среди других исследователей к данной тематике и перспективность ведения разработок в этой области.

Глава 1. Программная составляющая

1.1 Клиент для симулятора «grSim»

Клиент работает с симулятором «grSim». Для работы сначала необходимо скачать симулятор с сайта разработчиков и только потом установить разработанный клиент для работы с ним.

Клиент имеет пять изначально доступных действий для работы с ним. Это определение параметров мяча, определение параметров указанного робота, постановка мяча в точку с заданными скоростями вдоль каждой из осей, постановка робота в точку с заданной ориентацией и выполнение действия (присвоение роботу скорости вдоль какой-то оси или вдоль направления ориентации, включение и выключение спиннера, выполнение удара или навеса с указанной силой).

Более подробную информацию о том, как установить клиент и описание работы методов можно просмотреть в Приложении А.

1.2 Язык программирования и фреймворк

Для написания программы был выбран наиболее распространенный в среде машинного обучения язык программирования Python. Данный выбор был обусловлен простотой и понятностью данного языка, а также наличием большого количества готовых библиотек и фреймворков, которые могут быть полезны, как для данной задачи, так и для последующих разработок. Для разработки нейронной сети был выбран фреймворк PyTorch [5], разработанный компанией Facebook.

Глава 2. Алгоритмы и методы необходимые для обучения нейронной сети

Для обучения нейронной сети было необходимо реализовать некоторые алгоритмы не предусмотренные клиентом и которые используются в процессе работы программы. Рассмотрим их более подробно. Ознакомиться с кодом можно в соответствующих приложениях.

2.1 Алгоритм постановки робота на штрафную линию

За основу алгоритма постановки робота на штрафную линию был взят метод клиента симулятора для постановки робота в заданную точку поля с заданным углом направления кикера относительно оси абсцисс, а также постановка мяча в заданную точку с заданными скоростями вдоль осей. В методе постановки робота координата абсциссы постоянна, так как штрафная линия ортогональна соответствующей оси, а для координаты ординат берется случайное число с равномерным распределением из возможных значений ординат точек штрафной линии, используемых для обучения. Также необходимо поставить мяч перед кикером робота для последующего нанесения удара. Для этого используется метод постановки мяча в точку, где все скорости нулевые, а координаты вычисляется по следующим формулам:

$$\begin{cases} x = robot_x + k\cos\left(robot_{orientation}\right) \\ y = robot_y + k\sin\left(robot_{orientation}\right) \end{cases}, \tag{3}$$

где $robot_x$, $robot_y$ – абсцисса и ордината центра робота, а $robot_{orientation}$ – угол между направлением кикера и осью абсцисс, k – коэффициент зависяций от размера робота. Так как робот имеет не полностью круглую форму, а немного скошен в области кикера, то k берется меньше значения радиуса и подбирается опытным путём, в работе для формулы (3) было взято значение $k = 0.111$. Также важным моментом является необходимость включения спиннера после постановки робота с мячом, для того, чтобы робот сохранял контроль над мячом и мог нанести удар после любого количества поворотов. Для этого использовался метод симулятора для выполнения действия, где флагу спиннера присваивалось значение $True$, остальные параметры при этом не менялись. Результат работы алгоритма изображен на Рис. 4.

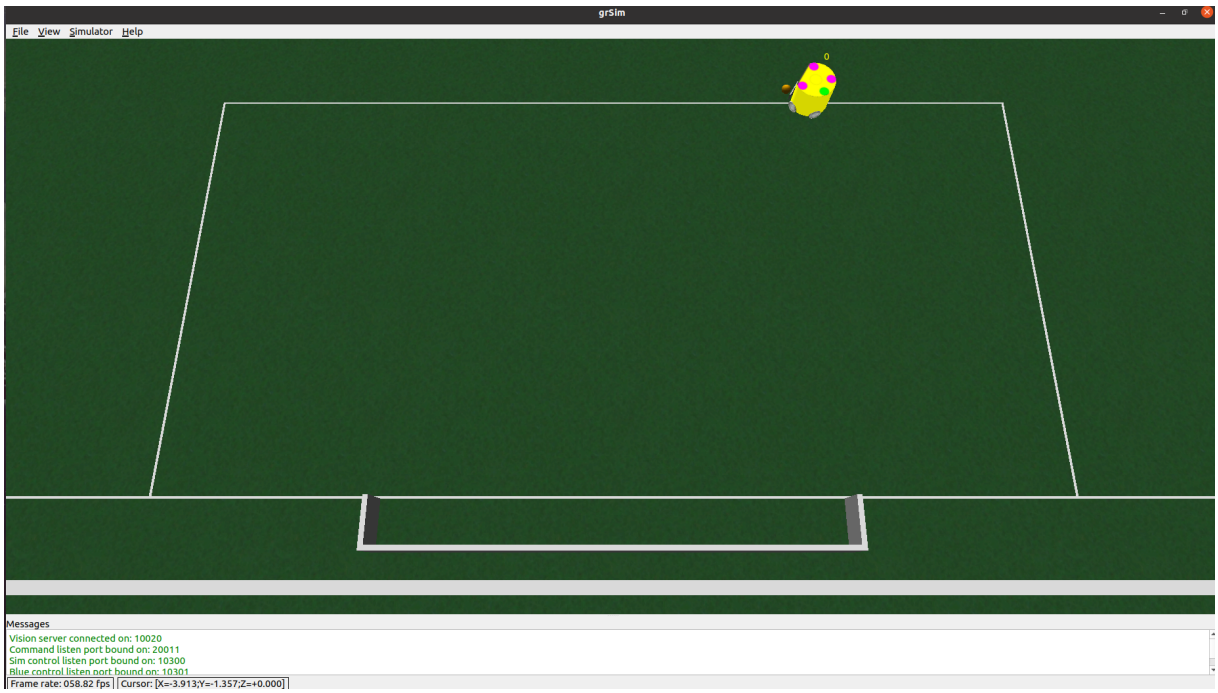

Рис. 4: Результат работы алгоритма постановки робота с мячом.

2.2 Алгоритм нанесения удара

Для нанесения удара использовался предусмотренный клиентом метод выполнения действия, где значению силы удара присваивалось значение равное 5, после чего обнулялось, а остальные параметры оставались неизменными.

2.3 Алгоритм поворота налево или направо на постоянный угол

Для алгоритма поворота направо и налево был использован метод постановки мяча в точку с заданным углом, где значения координат робота не менялось, а к значению угла робота относительно оси абсцисс прибавлялось или вычиталось значение постоянного угла в зависимости от направления поворота робота, в том случае, если значение угла было равно $-\pi$, то ему присваивалось значение π, если полученное значение превосходило π, то ему присваивалось значение $-\pi + \varepsilon$, где ε – постоянный угол поворота ($\varepsilon > 0$).

2.4 Алгоритм определения забитого гола

Для ускорения процесса обучения был использован геометрический метод определения забития гола. Для этого строилась точка пересечения прямой, образованной границей поля вдоль линии ворот и прямой, образованной точкой центра робота и его ориентацией. Если кикер направлен в сторону ворот, т.е. значение ориентации робота по модулю больше $\frac{\pi}{2}$, а также ордината точки пересечения названных выше прямых лежит в диапазоне ординат точек ворот, то будем считать, что нанесенный из данного положения удар приведет к голу. Более наглядно алгоритм можно изучить на Рис. 5.

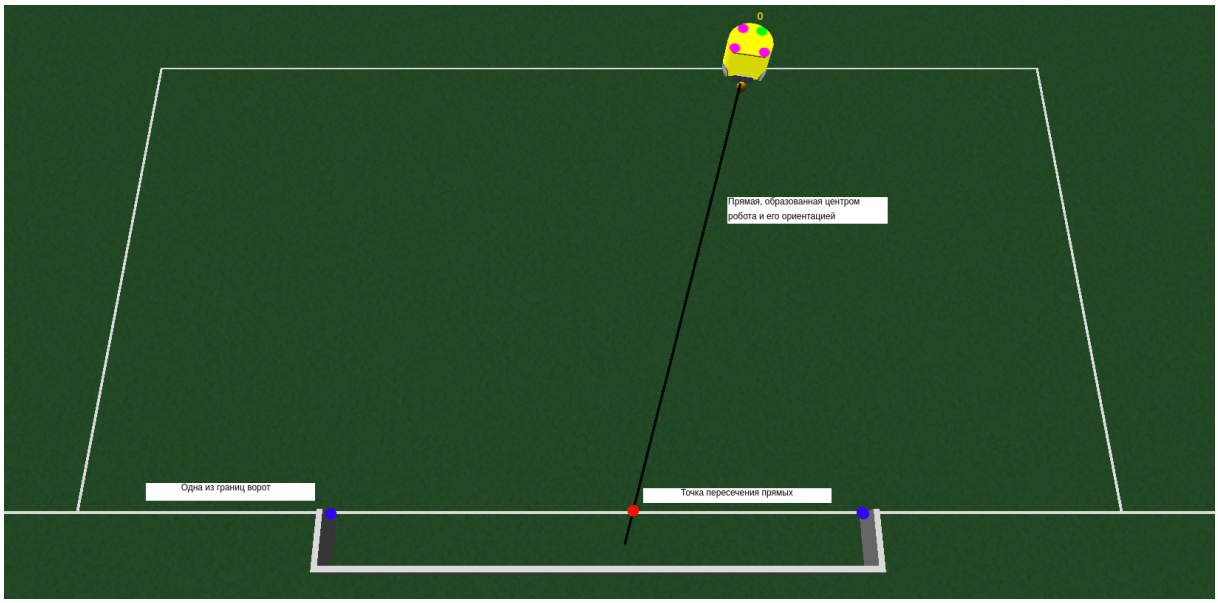

Рис. 5: Результат работы алгоритма постановки робота с мячом.

Глава 3. Обучение нейронной сети

Для обучения нейронной сети было решено разбить пространство состояний на конечное множество точек. Вдоль оси ординат было взято разбиение штравной линии на точки отстоящие друг от друга на одинаковом расстоянии, всего 13 точек, а множество значений углов представляет из себя разбиение полуинтервала $(-\pi; \pi]$ с шагом в $\frac{\pi}{18}$. После полученное решение для дискретной модели состояний применяется для непрерывной, где каждому непрерывному сектору сопоставляется решение соответствующей точки дискретной модели состояний.

3.1 Структура нейронной сети

Для решения задачи была Использована многослойная нейронная сеть прямого распространения. Входной слой нейронной сети представляет из себя два нейрона, где на вход подаются значения ординаты и ориентации робота. Далее следуют три внутренних слоя с 40, 60 и 80 нейронами соответственно. На каждом слое используется одна и та же активационная функция $ReLU$. Выходной слой состоит из трех нейронов, которые представляют из себя значение Q-функции для поворота налево, поворота направо и удара соответственно.

В качестве оптимизатора был выбран встроенный в $Pytorch$ метод $Adam$ с шагом обучения равным 0.0025 и надстройкой $AMSGrad$, которая вместо экспоненциального среднего использует максимум прошлых квадратов градиентов для обновления параметров, что улучшает сходимость алгоритма. За функцию потерь было взято среднеквадратичное отклонение старого значения Q-функции от нового, посчитанного по формуле (2) из введения.

Также производится нормирование входных данных делением на соответствующие максимальные значения в пространстве состояний, таким образом мы получаем значения из отрезка [-1;1] для каждого нейрона входного слоя.

Способ генерации награды берется максимально абстрактный, $r = 1$ при забивании гола и $r = 0$ в любом другом случае.

Также важным моментом, улучшающим сходимость процесса обучения является уточнения состояния, считываемое специальным методом клиента. Так как симулятор имитирует помехи, которые возникают в реальной работе с роботами, то возвращает неточные значения координат и угла робота, что влияет на процесс обучения. Решить данную проблему можно присваивая каждой составляющей вектора состояния значения из набора допустимых, которое наиболее близко к считанному методом клиента.

3.2 Использование двух нейронных сетей

В процессе обучения нейронной сети было выявлено, что использование той же Q-функции, которая изменяется каждую эпоху обучения, для предсказания максимума значения Q-функции при расчёте ее обновления приводит к плохой сходимости алгоритма обучения. Для решения данной проблемы было решено использовать две нейронные сети, одна из которых обновляется каждую эпоху, а другая используется для предсказания максимумов Q-функции при обновлении первой, согласно уравнению Беллмана, ей присваиваются веса первой нейронной сети с какой-то частотой относительно итерации эпох, которая является одним из гиперпараметров обучения.

3.3 Метод памяти воспроизведения

При обучении возникли проблемы со сходимостью алгоритма, для их решения помог метод использования памяти воспроизведения [17]. Суть заключается в том, что мы храним переходы, которые наблюдает агент, что позволяет нам повторно использовать эти данные позже. При случайной выборке из него переходы, образующие пакет, декоррелируются. В результате это значительно стабилизирует и улучшает процедуру обучения DQN. Для метода нам понадобятся два класса:
1)Переход - именованный кортеж, представляющий один переход в нашей среде. По сути, он сопоставляет пары (состояние, действие) с их результатом (след. состояние, вознаграждение).

2)Память воспроизведения - циклический буфер ограниченного размера, в котором хранятся недавно наблюдавшиеся переходы. Он также реализует метод *.sample*() для выбора случайной партии переходов для обучения.

3.4 Полученные результаты

Использование описанной нейронной сети, метода памяти воспроизведения, а также разделения Q-функции на две нейронные сети позволило сушественно улучшить результаты работы нейронной сети и добиться решения задачи для дискретного случая во всех случаях. При переходе к непрерывному пространству состояний также удалось добиться забивания голов.

Глава 4. Анализ результатов

Полученные результаты показали применимость методов обучения с подкреплением, в частности Deep Q-learning, для решение поставленной задачи, возможность и удобство использования языка программирования *Python* и фреймворка *Pytorch* для написания программной части работы, а также симулятора «grSim» для моделирования среды робофутбола.

Заключение

В результате выполнения данной выпускной квалификационной работы была выполнена поставленная цель – решение задачи забивания гола со штрафной линии. Попутно был разработан клиент для работы с симулятором и разработаны некоторые методы и алгоритмы, которые могут быть полезны при дальнейших разработках.

Осталось только модернизировать полученное решения для наиболее оптимального решения поставленной задачи, что планируется сделать в дальнейшем, также планируется решить поставленную задачу при наличии неподвижных и движущихся препятствий. При успешных результатах последующей разработки данные алгоритмы планируется использовать в реальных условиях в команде роботов-футболистов СПбГУ «URoboRus».

Список литературы

[1] Макмахан Брайан, Рао Делип «Знакомство с PyTorch: глубокое обучение при обработке естественного языка»—2020—256 с.

[2] GitHub репозиторий разработчиков симулятора «grSim»: [Электронный ресурс]. URL: https://github.com/RoboCup-SSL/grSim. (Дата обращения : 24.05.2021).

[3] Noda, I., Matsubara, H., Hiraki, K., Frank, I. (1998). Soccer server: A tool for research onmultiagent systems.Applied Artificial Intelligence,12, 233–250

[4] Chen, Foroughi, Heintz, Kapetanakis, Kostiadis, Kummeneje,Noda, Obst, Riley, Steffens, Wang, and Yin (2003) : [Электронный ресурс]. URL: https://github.com/rcsoccersim. (Дата обращения: 24.05.2021).

[5] Инструкция от разработчиков *PyTorch*: [Электронный ресурс]. URL: https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html. (Дата обращения: 24.05.2021).

[6] Саттон Р.С., Барто Э.Г. «Обучение с подкреплением»—2-е изд. —2014 —402 с.

[7] Николенко С., Кадурин А., Архангельская Е. «Глубокое обучение» — 2018—480с.

[8] Franco Ollino,Miguel A. Solis, Héctor Allende «Batch Reinforcement Learning on a RoboCup Small Size League keepaway strategy learning problem»—2018—Proceedings of the 4th Congresson Robotics and Neuroscience

[9] Stone P, Sutton RS Kuhlmann G. «Reinforcement learning for RoboCup soccer keepaway»— 2005/ URL: https://www2.cs.duke.edu/courses/spring07/cps296.3/keepaway.pdf(Дата обращения: 24.05.2021).

[10] Сайт RoboCup: [Электронный ресурс] URL: https://www.robocup.org/ (Дата обращения:24.05.2021).

[11] Сайт лиги SSL RoboCup: [Электронный ресурс] URL: https://ssl.robocup.org/ (Дата обращения:24.05.2021).

[12] Petr Konovalov, Dmitry Korolev, Dmitry Kapustin, Galina Reneva, AnastasiiaVoloshina, Alexander Kalitin, and Alexander Fradkov «URoboRus 2020 Team Description Paper»—2020

[13] Kotova O. A., Pavlovskij V. E. Teoretikoigrovie kinematicheskie modeli virtual'nogo futbola [Game-theoretic kinematic models of virtual football]. Moscow, Institut prikladnoj matematiki imeni M. V. Keldysha RAN Publ., 2010 20 p. (Preprint. Institut prikladnoj matematiki imeni M. V. Keldysha RAN, no. 77). Available at: http://library.keldysh.ru/ preprint.asp?id=2010- 77 (accessed 29 March 2019) (In Russian).

[14] Lipkovich M. M., Luchin R. M., Soharev A. Yu. Educational and research stand for multi- agent control systems (using the example of robot football). Proc. of the Intern. Scientific Conf. of Students and Young Scientists "Theoretical and Applied Aspects of Cybernetics", Kiev, Bukrek, 2011, pp. 161–164 (In Russian).

[15] Z. Chen et al. Champion Team Paper: Dynamic Passing-Shooting Algorithm Based on CUDA of The RoboCup SSL 2019 Champion. Zhejiang University, Zheda Road No.38, Hangzhou, Zhejiang Province, P.R.China.

[16] Trevor Hastie, Robert Tibshirani, Jerome Friedman. «The Elements of Statistical Learning: Data Mining, Inference, and Prediction». New York: Springer— 2001— Print.

[17] Метод воспроизведения памяти: [Электронный ресурс] URL: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html (Дата обращения: 24.05.2021).

[18] П.А. Коновалов, А.И. Волошина, А.О. Калитин, Д.М. Королев, А.Л. Фрадков. Команда роботов-футболистов UROBORUS-2020 для сорев-

нований ROBOCUP SSL.Тезисы докла- дов Всероссийской конференциии «Информационные технологии в управлении» (ИТУ- 2020). АО «Концерн «ЦНИИ «Электроприбор», Санкт-Петербург, 6-8 октября 2020 г.

[19] Коновалов Петр Алексеевич, Корнилова Анастасия Валерьевна, Королев Дмитрий Михайлович, Ренева Галина Вадимовна, Фрадков Александр Львович, Широколобов Илья Юрьевич, Ярош Дмитрий Сергеевич «Команда роботов-футболистов для соревнований RoboCup в лиге SSL: система и алгоритмы»—2019

Приложение А. Репозиторий GitLab, клиент для симулятора и описание работы с ним

GitLab с кодом всех программ:
https://gitlab.com/diploma_2021/robo_football.git

Для работы с клиентом необходимо скачать папку simulator. Установить клиент можно с помощью команды:

Листинг 1: Установка клиента

```
cd simulator/
pip install .
```

Для работы с клиентом необходимо использовать:

Листинг 2: Инициализация и закрытие клиента

```
from grsim.client import GrSimClient

client = GrSimClient()
client.init()

# Work with client

client.close()
```

Для определения параметров робота и мяча нужно использовать следующие строки соответственно :

Листинг 3: Определение параметров робота и мяча

```
from grsim.model import Team

detection = client.get_detection()
robot = detection.get_robot(team=Team.YELLOW, robot_id=0)
robot = detection.get_ball()
```

Для использования постановки робота в точку с заданный углом использовать(присвоить x,y и direction необходимые значения):

Листинг 4: Постановка робота в заданную точку

```
from grsim.model import RobotReplacement
from grsim.model import Team

command = RobotReplacement(team=Team.YELLOW, robot_id=0, x=0, y=0, direction=0)
client.send_robot_replacement(command)
```

Для использования постановки мяча в точку с заданной скоростью использовать(присвоить x,y и скоростям необходимые значения):

Листинг 5: Постановка мяча в заданную точку

```
from grsim.model import BallReplacement
from grsim.model import Team

command = BallReplacement(x=0, y=0, vx=0, vy=0)
client.send_ball_replacement(command)
```

Для выполнения действия нужно использовать эти строчки с необходимыми значениями:

Листинг 6: Выполнение действия

```
from grsim.model import ActionCommand
from grsim.model import Team

command = ActionCommand(team=Team.YELLOW, robot_id=0, kickspeedx=0, kickspeedz=0,
veltangent=0, velnormal=0, velangular=0, spinner=False, wheelsspeed=False,
wheel1=0, wheel2=0, wheel3=0, wheel4=0)
client.send_action_command(command)
```

Также для использования алгоритмов поворота на угол, удара по мячу и постановку робота на штрафную линию со случайными значениями необходимо скачать с той же ссылки GitLab папку strtegy и использовать следующий код:

Листинг 7: Выполнение методов поворота, удара и постановки робота на штрафную линию

```
from grsim.client import GrSimClient
from grsim.model import Team
from strategy.penalty import Action, PenaltyEnvironment

client = GrSimClient()
client.init()

environment = PenaltyEnvironment(client)
environment.init_scenario(Team.YELLOW, 0)
environment.perform_action(Team.YELLOW, 0, Action.KICK)

client.close()
```

Приложение Б. Код класса Penalty.Actions

Листинг 8: Класс Penalty.Actions

```
import attr

from grsim.model import Team
from grsim.client import GrSimClient

from ..base.movements import Controller


@attr.s(auto_attribs=True)
class PenaltyActions:
    client: GrSimClient
    controller: Controller = attr.ib(init=False)

    def __attrs_post_init__(self) -> None:
        self.controller = Controller(self.client)

    def robot_kick_5_0(self, team: Team, robot_id: int, speed_x=5) -> None:
        self.controller.robot_kick(team, robot_id, speed_x, 0)
```

Приложение В. Класс PenaltyEnvironment

Листинг 9: Класс PenaltyEnvironment

```
from enum import Enum
```

```
import attr
import random
import math
import numpy as np
sr1 = random.SystemRandom()
sr2 = random.SystemRandom()

from grsim.client import GrSimClient
from grsim.model import RobotReplacement, BallReplacement, Team

from . import PenaltyActions

PENALTY_LINE_X: float = -4.2
COEFF: float = 0.111
values1 = [-1.8, -1.5, -1.2, -0.9, -0.6, -0.3, 0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8]
values2 = [-170, -160, -150, -140, -130, -120, -110, -100, -90, -80, -70, -60,
-50, -40, -30, -20, -10, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120,
130, 140, 150, 160, 170, 180]


class Action(Enum):
    TURN_RIGHT = 0,
    TURN_LEFT = 1,
    KICK = 2


@attr.s(auto_attribs=True)
class PenaltyEnvironment:
    client: GrSimClient
    actions: PenaltyActions = attr.ib(init=False)

    def __attrs_post_init__(self):
        self.actions = PenaltyActions(self.client)

    def turn_robot(self, team: Team, robot_id: int, y: float,
    orientation: float, action: Action):
        if(action == Action.TURN_LEFT):
            if((orientation + math.pi/18) > math.pi):
                finish = -math.pi + ((orientation+math.pi/18) % math.pi)
            else:
                finish = orientation + math.pi/18
            self._put_robot_with_ball(team, robot_id, PENALTY_LINE_X,
            y/1000, math.degrees(finish))
        elif(action == Action.TURN_RIGHT):
            if((orientation-math.pi/18) < -math.pi):
                finish = math.pi - ((-orientation + math.pi/18) % math.pi)
```

```
            else:
                finish = orientation - math.pi/18
            self._put_robot_with_ball(team, robot_id, PENALTY_LINE_X,
            y/1000, math.degrees(finish))

    def init_scenario(self, team: Team, robot_id: int):
        sr1.seed(random.randint(1,100))
        sr2.seed(random.randint(1,100))
        v1 = sr1.choice(values1)
        v2 = sr2.choice(values2)
        self._put_robot_with_ball(team, robot_id, PENALTY_LINE_X,
                                  v1,
                                  v2)#

    def perform_action(self, team: Team, robot_id: int, action: Action) -> None:
        if action.value == Action.TURN_KICK.value:
            raise Exception(f"Unsupported action: {action}")

    def _put_robot_with_ball(self, team: Team, robot_id: int, robot_x: float,
    robot_y: float, orientation: float):
        command = RobotReplacement(team=team, robot_id=robot_id, x=robot_x,
                                   y=robot_y, direction=orientation)
        self.client.send_robot_replacement(command)

        command = BallReplacement(x=robot_x + (COEFF *
                                  math.cos(math.radians(orientation))),
                                  y=robot_y + (COEFF *
                                  math.sin(math.radians(orientation))), vx=0, vy=0)
        self.client.send_ball_replacement(command)
```

Приложение Г. Фрагмент программы, посвященный обучению нейронной сети.

Полный код программы можно найти на упомянутом в Приложении А. репозитории GitLab.

Листинг 10: Блок с обучением нейронной сети

```
num_episodes = 7000
memory = ReplayMemory(10000)
for i_episode in range(num_episodes):
    print(i_episode)
    # Initialize the environment and state
    environment.init_scenario(Team.YELLOW, 0)
    time.sleep(0.05)
```

```
        detection = client.get_detection()
        robot = detection.get_robot(team=Team.YELLOW, robot_id=0)
        state = [robot.y, robot.orientation]
        state = filt_st(state)
        state = normalize_feature(state)
        init_state = state.clone()
        states.append((state))
        for t in count():
            # Select and perform an action
            detection = client.get_detection()
            robot = detection.get_robot(team=Team.YELLOW, robot_id=0)
            action, ind = select_action(state)
            ind = torch.tensor(ind)
            if(action == Action.KICK):
                environment.perform_action(Team.YELLOW, 0, Action.KICK)
                time.sleep(0.05)
            else:
                environment.turn_robot(Team.YELLOW, 0, robot.y, robot.orientation, action)
                time.sleep(0.05)

            reward = r_generation(action, robot, state, init_state)
            reward = torch.tensor(reward)
            if(action != Action.KICK):
                robot_next = detection.get_robot(team=Team.YELLOW, robot_id=0)
                next_state = filt_st([robot_next.y, robot_next.orientation])
                next_state = normalize_feature(next_state)
            else:
                next_state = None

            # Store the transition in memory
            memory.push(state, ind, next_state, reward)

            # Move to the next state
            state = next_state

            # Perform one step of the optimization (on the policy network)
            optimize_model()
            if(action == Action.KICK):
                break
        # Update the target network, copying all weights and biases in DQN
        if i_episode % TARGET_UPDATE == 0:
            target_net.load_state_dict(policy_net.state_dict())

print('Complete')
```