Санкт–Петербургский государственный университет

Радева Анастасия Александровна

Выпускная квалификационная работа

Система управления мобильным роботом

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.17.5005.1 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Компьютерные технологии и системы»

Научный руководитель: доцент,
кафедра компьютерных технологий
и систем, к. ф.- м. н.
Коровкин Максим Васильевич

Рецензент:
Семакова Анна Анатольевна

Санкт-Петербург
2021 г.

Содержание

Введение

В современном мире процесс автоматизации проникает во многие отрасли жизни: собираются транспортные средства на производствах, предоставляются различные услуги доставки мобильными роботами в кафе и отелях. Мобильные роботы обеспечивают высокую производительность работы различных предприятий как промышленных, так и сферы услуг. Нет необходимости переживать об организации труда и ошибках, связанных с человеческим фактором. Для развития процесса автоматизации необходимо всестороннее исследование возможности управления мобильным роботом, используя современные передовые способы разработки роботов. Одной из первоочередных задач является задача отработки движения робота по заданной траектории с помощью управления значениями на моторах колес. Такое движение часто используется в сельскохозяйственной промышленности роботами, использующими GPS-навигацию, а также роботами-экскурсоводами или уже многим знакомыми роботами-пылесосами.

Работа с физическим прототипом робота может осложняться большой стоимостью его составных частей и исследовательского оборудования. Кроме того, существует риск неудачных и травмирующих экспериментов, которые могут привести к поломке оборудования. Симуляция позволяет решать проблемы безопасности, позволяя предотвратить опасные для целостности и работоспособности робота ситуации, а также симуляция существенно экономит время исследования в случае необходимости многократного повторения экспериментов, используя возможности одновременного запуска различных ситуаций, необходимых, например, для обучения нейронных сетей.

Благодаря продвинутым технологиям средств симуляции можно воссоздать разнообразные поверхности, наделяя их необходимыми физическими свойствами, симулируя сопротивление и силу трения.

Некоторые исследователи считают, что работа в симуляции — это слишком сложно и трудоемко [6]. Однако в действительности современные платформы имеют большой спектр возможностей, что позволяет реализовать все необходимое для решения поставленных задач без особых ограничений и усилий.

Учитывая развитие, которое получили различные движки симуляции за последнее время, актуальной становится задача исследования возможности использования средств симуляции для разработки систем управления мобильными роботами. В настоящее время доступно не так много бесплатных сред симуляции для эффективного проведения экспериментов по управлению транспортными средствами. В данной работе рассматривается платформа Unity3D, являющаяся общедоступной средой разработки, имеет неплохую документацию и широко используется для решения различных задач. Несмотря на основное направление применения данной платформы в качестве среды для разработки компьютерных игр, что влечет за собой разносторонние упрощения и оптимизации процессов вычисления, Unity поддерживает PhysX SDK 4, благодаря чему его можно использовать в качестве физического движка, для чего создано множество дополнительных возможностей описания взаимодействия объектов окружающей среды и транспортных средств с физической точки зрения.

Постановка задачи

Целью данной работы является разработка полной системы управления мобильным роботом для преследования заданного движущегося объекта в среде симуляции. На основе реализованной системы в дальнейшем планируется перейти к взаимодействию с реальным физическим прототипом робота.

Для достижения этой цели были поставлены следующие задачи:

- реализовать симуляционную модель движения двухколесного робота с дифференциальным приводом в среде Unity;
- построить динамическую и кинематическую модели робота;
- синтезировать регулятор для движения по заданной траектории;
- реализовать коммуникацию модели в Unity с внешней программой для передачи изображения с виртуальной камеры для последующего распознавания объектов симуляции и вычисления необходимых скоростей для передачи их роботу;
- реализовать внешнюю управляющую программу, принимающую через компьютерную сеть изображение с виртуальной камеры, распознающую робота и объект преследования, а также способную передавать желаемые скорости для отработки роботом;
- синтезировать регулятор для преследования цели во внешней программе;
- продемонстрировать работоспособность на примере нескольких симуляционных экспериментов.

Обзор литературы

В статье [3] представляется математическая модель унициклического робота с явным учетом его динамики для формулирования стратегий управления движением для отслеживания траектории. Рассмотрены два метода когда транспортное средство должно отслеживать параметризованный по времени эталонный путь (набор положений), и следования пути, в котором цель состоит в том, чтобы направить транспортное средство на свой путь с эталонной скоростью. На её основе была описана динамическая модель унициклического робота в данной работе, кроме того также исследуются отработка движения по траектории, без каких-либо ограничений на скорости.

На данный момент существует множество методов для контроля движения робота, в том числе и для задачи позиционирования, что хорошо описано в данной книге [4]. Здесь так же приведен необходимый код для разработки в системе Matlab и Simulink. В данной работе будут рассмотрены один из методов данной книги и метод управления с обратной связью по состоянию из статьи [9] для сравнения, все эксперименты будут проводиться в среде симуляции Unity в качестве исследования.

Одной из довольно популярных тем в разработке роботов с распознаванием изображения является автономный футбол роботов. Например, в статье [2] используют цветные макеры, закрепленные на роботах, но в данном методе имеются недостатки из-за проблем с калибровкой цвета. Обнаружение объектов по ArUco-маркерам позволяет избежать необходимость калибровки цвета, что делает решение задачи ориентации объектов в пространстве более надежным и стабильным [1].

Таким образом, можно заметить что в сфере робототехники изучено и проведено много экспериментов, но многие из них не образуют цельную систему, состоящую из нескольких компонентов: симуляции, внешнего регулятора, различные виды управления и компьютерного зрения, которые можно изучать и развивать во многих направлениях.

Глава 1. Среда симуляции

Каждая среда симуляции определяет свои правила работы с объектами, правила задания роботов, а также возможности создания окружающей среды и других необходимых объектов. Представляет интерес реализация симуляции робота в нетипичных для данных задач средах разработки, таких как Gazebo или Matlab, так как они развиваются в направлении симуляции и навигации, но редко изучаются специалистами робототехники. Именно поэтому было решено обратить внимание на использование графических и физических движков, таких как Unity и Unreal Engine.

Unity и Unreal Engine в первую очередь бесплатные и открытые для персонального использования программы разработки, дающие возможность работать с 3D графикой высокого уровня. Они предоставляют открытый доступ к исходному коду, что помогает более тщательно разобрать каждый этап симуляции движения. Среда разработки и симуляции может представлять из себя тестовую среду, являющуюся «цифровым двойником» реального окружения, которая позволила бы тестировать и экспериментировать в любом масштабе. В дополнение к этому есть возможность немедленной модификации виртуализированного оборудования, что позволяет сделать его надежным (или ненадежным), насколько это необходимо.

Существуют огромные библиотеки предметов окружения и средства для предельно точной визуализации объектов, что очень важно для различных оптических систем связанных с роботом, получающих информацию с картинки. Unity имеет обширные настройки камеры, что позволяет симулировать разные виды оптических шумов и искажений.

Unity, кроме того, обладает большим потенциалом к расширению и готовности к адаптации в стремительно меняющихся условиях, благодаря мощной системе программирования на C#, богатым API и документации. Благодаря интуитивно понятному интерфейсу и инструментам Unity, разработка рабочего прототипа проходит без необходимости тратить время на низкоуровневое программирование.

1.1 PhysX

Физика и физическое движение поддерживается в Unity с помощью кросспалтформенного физического движка PhysX от Nvidia, предоставляющего большой набор функций для симуляции. PhysX 4.0 обеспечивает высокое качество моделирования на уровне производительности моделирования игр. Он поддерживает динамику твердого тела, динамику мягкого тела (например, моделирование ткани, включая разрыв и сжатую ткань), тряпичные куклы и контроллеры персонажей, динамику транспортных средств, моделирование частиц и объемной жидкости. Ключевой особенностью также является открытость источников кода данного движка.

PhysX используют и такие среды симуляции как Microsoft Robotics Studio,Unreal Engine, пакеты 3D разработки Autodesk, а также недавно созданная среда разработки Nvidia Omniverse также симулирует движение твёрдых тел на основе поддержки PhysX и другие. Поэтому можно сказать, что Unity является не более чем удобным интерфейсом для взаимодействия с PhysX и не накладывает никаких ограничений на симуляцию движения мобильного робота по сравнению с другими схожими инструментами разработки роботов.

Глава 2. Модель двухколесного робота с дифференциальным приводом

Модель — это своего рода некоторая форма отражения действительности. В математической среде обычно под моделью понимается набор формул, описывающий физические свойства, которые или выводятся в процессе наблюдения за моделируемым объектом или системой, или уже выведены для изучения поведения модели в реальной среде.

Далее будем рассматривать предметно-физическое моделирование, что означает сходство физических свойств нашей модели робота с реальным объектом. Кроме того, рассматриваемая модель копирует не только свойства робота, но и отображает его действительный размер в формате 3D-объекта.

2.1 Двухколесный робот с дифференциальным приводом

Колесо является безусловно самым популярным механизмом передвижения в мобильной робототехнике и в искусственных транспортных средствах в целом. Одним из плюсов колесного робота в том, что баланс обычно не является проблемой для исследования в конструкциях колесных роботов, потому что колесные роботы почти всегда спроектированы так, что все колеса находятся в контакте с землей. Таким образом, трех колес достаточно для обеспечения стабильного баланса [8].

Существует четыре основных класса колес, они сильно различаются по своим кинематическим свойствам, и, следовательно, выбор типа колеса оказывает большое влияние на общую кинематику мобильного робота. Унициклические роботы широко используются в робототехнике, поскольку их движение легко программировать и ими можно хорошо управлять. Практически все потребительские роботы на рынке сегодня используют дифференциальное рулевое управление в первую очередь из-за его низкой стоимости и простоты. Унициклический робот — это мобильный робот, движение которого основано на двух отдельно приводимых колесах, расположенных по обе стороны от корпуса робота. Он может менять свое направление, изменяя относительную скорость вращения своих колес, и, следовательно, не требует дополнительного рулевого движения, имеет радиус поворота, сравнимый с

размером робота и высокую маневренность. Учитывая популярность таких роботов в индустрии и относительную простоту кинематической модели, в этой работе используется модель данного типа.

Рассматриваемый робот оснащён двумя колесами, имеющими одну степень свободы и собственный дифференциальный привод, с третьей точкой контакта для опоры. В мобильном роботе с дифференциальным приводом два двигателя, прикрепленных к двум колесам, которые должны приводиться в движение по одному и тому же профилю скорости.

2.2 Кинематическая модель

На протяжении всего описания будем представлять робота как твердое тело на колесах, движущееся в горизонтальной плоскости. Чтобы указать положение робота на плоскости, установим связь между глобальной системой отсчета плоскости и локальной системой отсчета робота, как показано на рисунке 1.

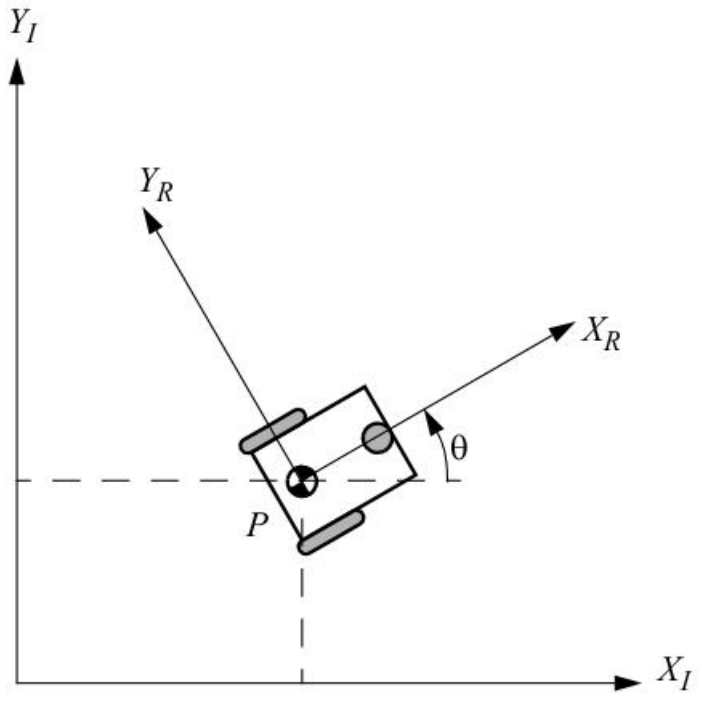

Рис. 1: Глобальные и локальные оси координат робота [8]

Оси X_I и Y_I определяют произвольный инерциальный базис на плоскости как глобальную систему отсчета от некоторой точки отсчета (например,

от центра камеры) O: X_I,Y_I. Чтобы указать положение робота, выбираем точку P посередине между колесами робота в качестве контрольной точки своего положения. Базис X_R,Y_R определяет две оси относительно P и, таким образом, является локальной системой координат робота. Положение P в глобальной системе отсчета задается координатами x и y, а угол между глобальной и локальной системами отсчета определяется как θ.

Положение робота можно описать как вектор с этими тремя элементами (1). Индекс I, используется чтобы указать положение робота в глобальной системе отсчета:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1}$$

Чтобы описать движение робота в его локальной системе отсчета, необходимо будет повернуть глобальную систему координат относительно локальной системы отсчёта.

Данное отображение является функцией, которая задаёт текущую позицию робота. Это отображение выполняется с использованием матрицы поворота (2):

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Эта матрица может использоваться для отображения движения в глобальной системе отсчета X_I,Y_I на движение в локальной системе отсчета X_R,Y_R. Эта операция обозначается через $R(\theta)\dot{\xi}_I$. Поэтому вычисление этой операции зависит от значения θ:

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I, \tag{3}$$

где $\dot{\xi}$ — вектор скоростей для соответствующих компонент.

В простейших случаях отображения, описываемого последним уравнением (3), достаточно для формулы, которая отражает кинематику мобильного

робота.

Если рассматривать соотношение между скоростями подаваемыми роботу и скоростями каждого из колес, можно записать:

$$V_L = r\omega_L, \quad V_R = r\omega_R,$$

где r — радиус колеса, L — расстояние между колёсами.

$$\omega = \frac{V_R - V_L}{L}, \quad v = \frac{V_R + V_L}{2},$$

Тогда кинематическая модель соответственно переписывается:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Кинематическая модель унициклического робота в основном связана с геометрическим описанием траекторий, по которым движется робот, без учета сил. Следовательно, могут произойти существенные ошибки при моделировании или тестировании реального прототипа. Поэтому необходимо рассмотреть динамическую модель, которая позволяет учитывать физические ограничения робота или сил.

2.3 Динамическая модель

Согласно второму закону Ньютона, упрощенная поступательная и вращательная динамика двухколесного робота описывается следующими формулами

$$M\dot{v} = F - B_v v$$

$$J\dot{w} = T - B_w w,$$

где M - масса робота, J - момент инерции, F - силы, приложенные к системе, T - сила трения, B_v - коэффициент трения поступательного движения, а

B_w - коэффициент трения вращения. Будем предполагать, что эти значения постоянны для значений скоростей движущегося робота.

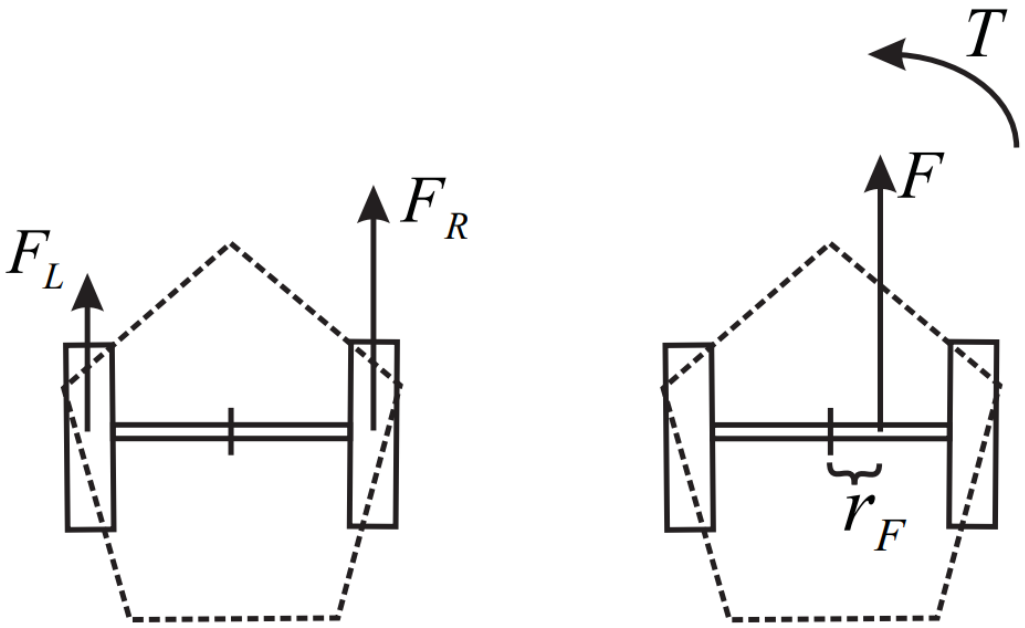

Рис. 2: Приложение силы на колесах (а) и эквивалентное представление (б) [3]

Учитывая силы, действующие на левое и правое колеса, F_L и F_R, получается, что эти силы эквивалентны силе F, приложенной в точке r_F, относительно центра оси колеса, как показано на рисунке 2.

$$F = F_R + F_L$$

$$T = l(F_R - F_L),$$

где l представляет половину длины оси, соединяющую колеса.

Аналогично, учитывая напряжения на моторах левого и правого колеса можно также определить среднее (пропорциональное сумме моментов m_R и m_L на моторах) и разностное (пропорциональное разности моментов на моторах) напряжения, e_{am} и e_{ad}, и, наконец, получить соотношение между напряжениями и силами, $F = K_m e_{am} - K_v$ и $T = K_w e_{ad} - K_w w$, где K_m, K_v, K_d, K_w — постоянные. В итоге мы имеем следующую динамическую модель униццикличесого робота:

$$
\begin{aligned}
Mdv &= -K_v v + a(m_L + m_R) \\
Jdw &= -K_w w + b(m_L - m_R),
\end{aligned}
\tag{4}
$$

Глава 3. Идентификация динамической модели

Проведя необходимые преобразования с моделью (4) и внеся значения масс и инерции в значения констант, можно получить следующую систему дифференциальных уравнений, с помощью которых определяются скорости мобильного робота:

$$\begin{aligned} dv &= k_v v + a(m_L + m_R) \\ dw &= k_w w + b(m_L - m_R), \end{aligned} \tag{5}$$

где m_L, m_R — значения крутящего момента двигателя на левом и правом колесах соответственно, задающие напряжение, v, w — линейная и угловая скорости.

Коэффициенты k_v, k_w, a, b системы (5) неизвестны, для их нахождения необходимо составить несколько систем путем проведения нескольких экспериментов с различными значениями крутящих моментов, получая линейные и угловые скорости из Unity.

Было проведено несколько экспериментов с одинаковыми и разными крутящими моментами. Система с конкретными коэффициентами численно решалась с помощью пакета `scipy`, а нужные коэффициенты были найдены с помощью минимизации функции среднеквадратичной ошибки. Данная функция количественно определяет ошибку между прогнозируемыми и ожидаемыми значениями и представляет ее в виде единственного действительного числа. Цель состоит в том, чтобы найти значения параметров модели, для которых функция возвращает как можно меньшее число.

Подставляя значения крутящих моментов и скорости, полученные в результате экспериментов, решались дифференциальные уравнения и таким образом вычислялось прогнозируемое значение, а в качестве ожидаемого значения использовались значения скоростей из экспериментов полученных в Unity.

Формулу среднеквадратичной ошибки MSE (Mean Square Error) можно записать так:

$$MSE = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^i - y^i),$$

где i — индекс выборки, $\hat{y}$ прогнозируемое значение, y — действительное значение, m — количество выборок в наборе данных.

3.1 Задание модели в Unity

За физику колёс в Unity отвечает компонент `WheelCollider`, поведение которого описывается и реализовано в PhysX, параметры самих колес можно задавать в Unity. Каждое колесо имеет свой Wheel Collider, который аппроксимирует форму колеса определенного радиуса, и имеет встроенное обнаружение коллизии, отвечающей за свойства объекта при столкновении, а также модель трения шин и свойства подвески.

На приведенном выше рисунке круг колеса и диаметр колеса отмечены зеленым, сегмент хода подвески - оранжевым, а сфера точки приложения силы - зеленым. На участке хода подвески есть отметки для положения максимального сжатия, положения максимального наклона и целевого положения. Колесо может перемещаться только между положениями максимального сжатия и максимальной нижней точки. Положение покоя находится именно там, где пружинящая масса уравновешивается силой упругости; то есть положение, в котором находится колесо, когда автомобиль просто стоит на плоской поверхности.

Именно эти движения подвески определяются в `WheelCollider` параметрами *Suspension Spring*: *Spring*, *Damper*, *Target Position* и *Force App Point Distance*. Также в `WheelCollider` задаются масса колеса, его радиус, значение демпинга, применяющегося к колесам, которое симулирует потерю энергии.

Трение шин может быть описано показанной ниже кривой трения колеса. Изначально определяется то, как сильно скользит шина (на основании разности скоростей между резиной шины и дорогой). Затем значение скольжения используется для того, чтобы найти силу, применяемую шиной к точке соприкосновения.

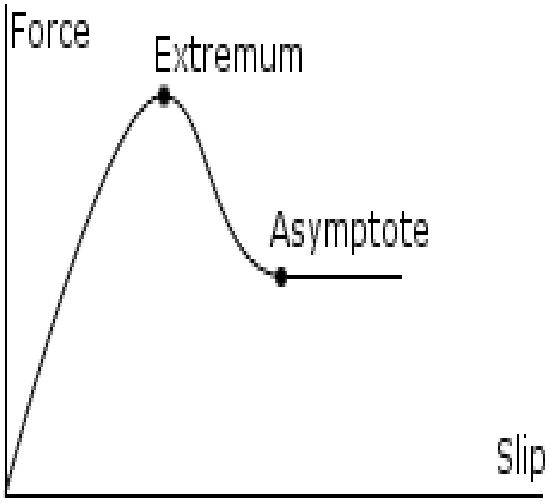

Рис. 3: Кривая трения [10]

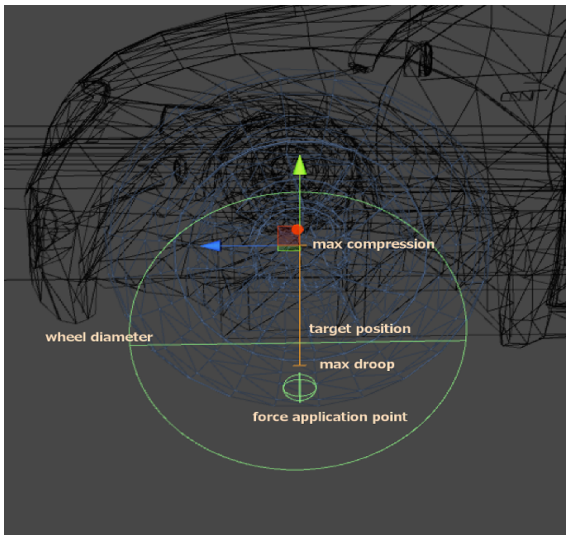

Рис. 4: Структура [10]

Кривая, аппроксимируемая по двум секциям, принимает измеренное скольжение шины и выдаёт силу на выходе. Первая секция идёт от нулевой точки до точки *Extremum*, задаваемой параметрами *ExtremumSlip* и *ExtremumValue*, в которой касательная кривой равна нулю. Вторая секция идёт от *Extremum* до точки *Asymptote*, определяемой параметрами *AsymptoteSlip* и *AsymptoteValue*, где касательная к кривой вновь равна нулю.

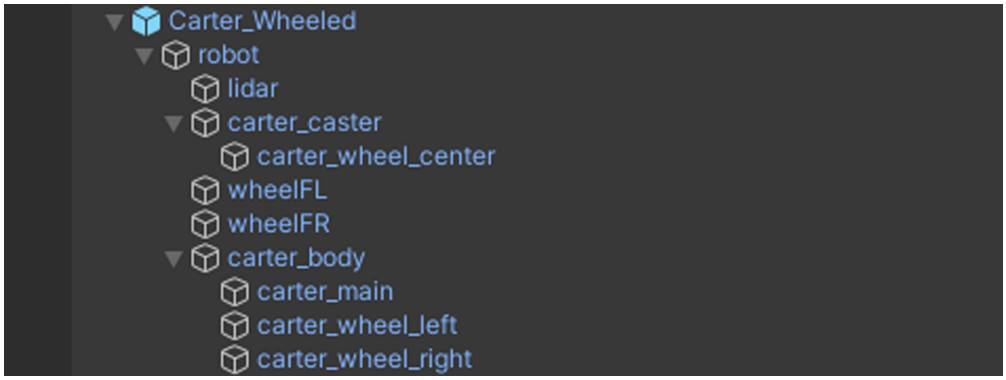

Рис. 5: Структура «Carter» в Unity

Детальное описание модели «Carter_Wheeled» :

- Carter_Wheeled содержит основной скрипт для работы с роботом `UnityDifferentialBaseSimulation`;
- подсекция robot преедставляет из себя `RigidBody`, позволяющий двигаться модели по законам физики, с помощью него же задается масса

в килограммах (50), сопротивление воздуха и сопротивление воздуха при повороте, использование гравитации;

- lidar — сенсор для определения глубины (не используется);
- carter_caster — 3D-модель(меш) основания маленького колёсика;
- carter_wheel_center — 3D-модель колёсика;
- wheelFL и wheelFR — коллайдеры левого и правого колеса соответственно;
- carter_body содержит в себе carter_main — 3D-модель основной части робота, carter_wheel_left и carter_wheel_right — 3D-модели левого и правого колёс соответственно, содержащие в себе скрипт Wheel Visualizer, отвечающие за вращение модели колес соответственно скорости;

Глава 4. Позиционирование робота

Рассмотрим проблему перемещения к цели (x_*,y_*) на плоскости с помощью контролирования линейной скорости робота, чтобы она была пропорциональна его расстоянию от цели [4]:

$$v_* = K_v\sqrt{(x_* - x)^2 + (y_* - y)^2}, \tag{6}$$

где x,y — координаты робота в глобальной системе координат.

Вычисляем угол, на котором находится цель по отношению к роботу:

$$\theta_* = arctg\frac{y_* - y}{x_* - x} \tag{7}$$

Используя пропорциональный контроллер, который определяет угловую скорость для поворота, направляемся к цели:

$$\omega_* = K_h(\theta_* - \theta), \tag{8}$$

где θ — угол поворота локальной системы координат относительно глобальной системы координат.

Если в добавок к повороту из 2.2 для перехода в локальную систему координат ещё производить сдвиг, можно заметить, что координаты робота в локальной системе координат всегда равны нулю, что упрощает вычисление формул (6), (7) и (8), так как остается только определить координаты цели в локальной системе координат робота.

Регулятор (6), (8) можно использовать в случае, когда робот обладает низкоуровневой системой управления для отработки желаемого вектора скоростей. Также в Unity существует возможность подавать управляющее воздействие в виде крутящих моментов на моторах, что обычно используется для работы с физическим прототипом робота. В статье [9] описан вопрос построения низкоуровневого закона управления с обратной связью по состоянию пропорционально-дифференциальной формы, который стабилизирует желаемое равновесие.

Рассмотрим уравнение следующего вида:

$$u = -K_v v - R^T K_p (x - x_*) \tag{9}$$

$$R = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix},$$

где $u = (m_L, m_R)$ – моменты на колесах, θ — угол поворота локальной системы координат относительно глобальной системы координат, вектор v — вектор текущих скоростей (линейной и угловой), x и x_* являются векторами текущих координат робота и цели соответственно в виде (1). Матрицы K_v, K_p — диагональные, положительно-определенные.

Произведем переход в локальную систему координат робота, как уже было сделано выше с помощью формулы (3), и опять таки заметим что текущие координаты робота x будут равны нулю.

Для сравнения данных методов управления были проведены эксперименты и построены графики. Их подробное сравнение рассмотрено в главе 6.

Глава 5. Система управления роботом

Среда симуляции Unity позволяет двумя способами управлять роботом. Один из которых определяется заданием соответствующих линейной и угловой скоростей, вычисленными подходящим методом исходя из необходимой для решения задачи. Второй способ — это напрямую задание необходимых значений крутящего момента на каждое колесо. Данный вариант хорошо согласуется с дальнейшей работой над физическим прототипом, так как управление роботом осуществляется именно за счёт управления значениями крутящего момента.

Для дальнейших экспериментов и получения обратной связи были разработаны две различные системы управления, одна из которых работает полностью на стороне симуляции Unity, другая же имитирует взаимодействие с физическим прототипом в реальном мире посредством работы с изображением, поступающим с камеры и дальнейшим распознаванием.

5.1 Комплекс связи с внешней программой

Для того, чтобы симулировать реальное дистанционное управление физическим прототипом робота, которое часто осуществляется с использованием протоколов передачи данных таких как Wi-Fi [5] или Bluetooth [7], реализована программа, организующая приём видео, определение положения объектов и последующее управление роботом. Происходит расчёт вектора необходимых линейной и угловой скоростей по формулам (6) и (8) соответственно, после чего вектор передается роботу.

Для передачи видео во внешний комплекс, в Unity записывается текстура заданного размера и формата, полученная с камеры, и кодируется в JPEG-изображение, которое отправляется http-сервером в формате MJPEG (Motion JPEG), потоком изображений JPEG, передаваемых по протоколу http. В настоящее время он обычно используется во многих мультимедийных приложениях, особенно в цифровых камерах, IP-камерах и веб-камерах. Основное преимущество использования MJPEG заключается в том, что для него не требуется установка клиентского программного обеспечения на удаленный компьютер. Для просмотра потокового видео с сервера потоковой переда-

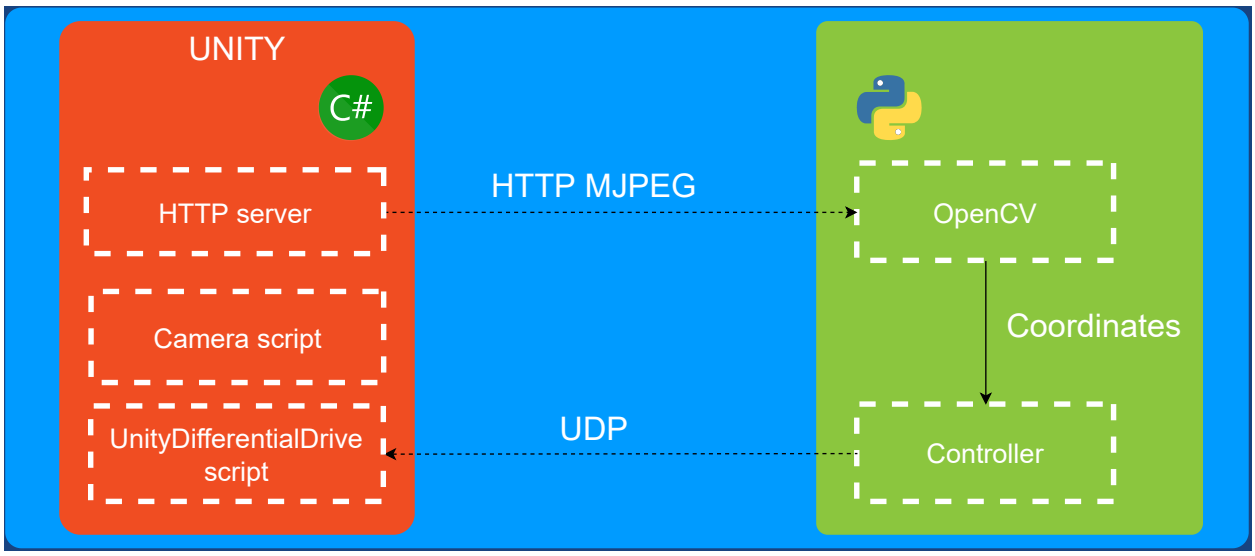

Рис. 6: Диаграмма компонентов решения

чи MJPEG понадобится любое программное обеспечение, поддерживающее потоковую передачу в формате Motion JPEG, например Firefox, Chrome или VLC.

Прием и распознавание изображения производится во внешней программе написанной на языке программирования Python с использованием библиотеки OpenCV — библиотеки компьютерного зрения и алгоритмов обработки изображений.

Существует два популярных протокола передачи данных по компьютерным сетям на транспортном уровне — это Transmission Control Protocol (TCP) и User Datagram Protocol (UDP), главное различие которых в том, что первый обеспечивает гарантию доставки данных, а второй нет.

Так как позиционирование робота — задача реального времени, гарантия доставки сообщений вносит слишком большие накладные расходы и может привести к доставке уже неактуальных данных. Сообщения от робота отправляются с частотой 30 Гц, поэтому потеря небольшого числа из них не создает проблем, тем более в современных каналах связи такие потери практически не возникают. Следовательно, внешняя программа использует UDP протокол для организации обратной коммуникации с моделью робота, отправляя строку, состоящую из линейной и угловой скоростей, разделенных пробелом, например, «3 2». Схематично, разработанное решение можно увидеть на рисунке 6.

5.2 Обратная связь по изображению

Оценка позы имеет большое значение во многих приложениях компьютерного зрения: навигации роботов, дополненной реальности и многих других. Этот процесс основан на нахождении соответствий между точками в реальной среде и проекцией их 2-мерного изображения. Обычно это трудный этап, поэтому для облегчения обычно используют синтетические или реперные маркеры.

В среде симуляции была установлена камера, основанная на физических параметрах реального сенсора 65mm ALEXA, имеющего довольно большое разрешение и угол обзора, что позволяет охватить большее пространство вокруг объектов.

Один из самых популярных подходов - использование двоичных квадратных меток координат. Основное преимущество этих маркеров заключается в том, что один маркер обеспечивает достаточное количество соответствий (его четыре угла) для получения позы камеры. Кроме того, внутреннее двоичное кодирование делает их особенно надежными, позволяя применять методы обнаружения и исправления ошибок.

Модуль aruco основан на библиотеке ArUco, популярной библиотеке для обнаружения квадратных реперных маркеров, разработанной Рафаэлем Муньосом и Серхио Гарридо [1].

Маркер ArUco — это квадратный маркер, состоящий из широкой черной границы и внутренней двоичной матрицы, которая определяет его идентификатор (id). Черная рамка способствует ее быстрому обнаружению на изображении, а двоичная кодификация позволяет ее идентифицировать и применять методы обнаружения и исправления ошибок. Для лучшего обнаружения маркер был помещён на белый фон, во избежание ошибок распознавания.

Для генерации маркера используются словари отвечающие за кодификацию и количество маркеров, которые можно использовать в рамках данного словаря, а именно это список двоичных кодификаций каждого из его маркеров.

В среде симуляции размещены двухколесный робот и куб — обозначающий цель позиционирования. К этим объектам прикреплены маркеры с иден-

тификаторами 0 и 1 соответственно, сгенерированные из словаря Dictionary 4x4(50), размер маркера 4x4 состоит из 16 бит.

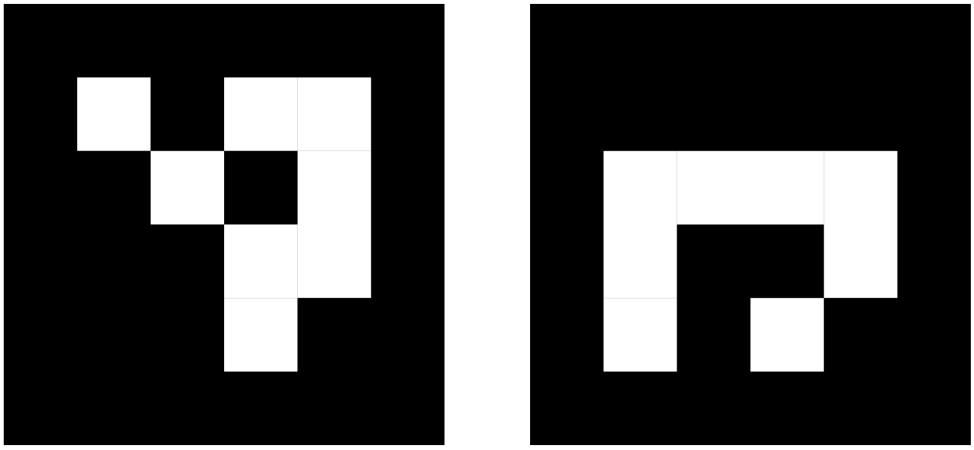

Рис. 7: ArUco маркеры $id = 0$ и $id = 1$ [8]

Во внешний программный комплекс поступает поток изображений, которые можно автоматически расшифровывать и выводить на экран, далее с помощью метода detectMarkers из класса aruco [11] происходит распознавание маркеров. Входными параметрами метода является изображение, словарь, с помощью которого были сгенерированы маркеры на изображении и набор параметров, в данном случае используется стандартный набор параметров предлагаемых классом.

Процесс обнаружения маркера состоит из двух основных этапов: обнаружение кандидатов в маркеры и их проверка. На этом этапе изображение анализируется, чтобы найти квадратные формы, которые могут быть маркерами. Вторым этапом является определение, действительно ли они являются маркерами, путем анализа их внутренней кодификации.

Метод возвращает координаты четырёх углов по часовой стрелке в системы координат изображения, начиная с левого верхнего для каждого обнаруженного маркера, id найденных маркеров, а также координаты четырех углов отклонённых кандидатов. Далее по этим данным на изображении обводятся зеленой линией распознанные маркеры и подписывается идентификатор, для прямого отслеживания моментов обнаружения и проблем из-за которых происходит потеря маркеров.

Для решения задачи позиционирования по изображению необходимо совершить перевод объекта цели в локальную систему координат робота, и получить угол поворота относительно новой системы координат. Переход к новой системе координат осуществлялся последовательно: переносом и поворотом системы координат. Реализация этих действий требует понимания вектора направления, и соответствующего поворота локальной системы координат относительно глобальной. Ниже на изображении наглядно показана связь систем координат камеры, изображения и мировой системы координат.

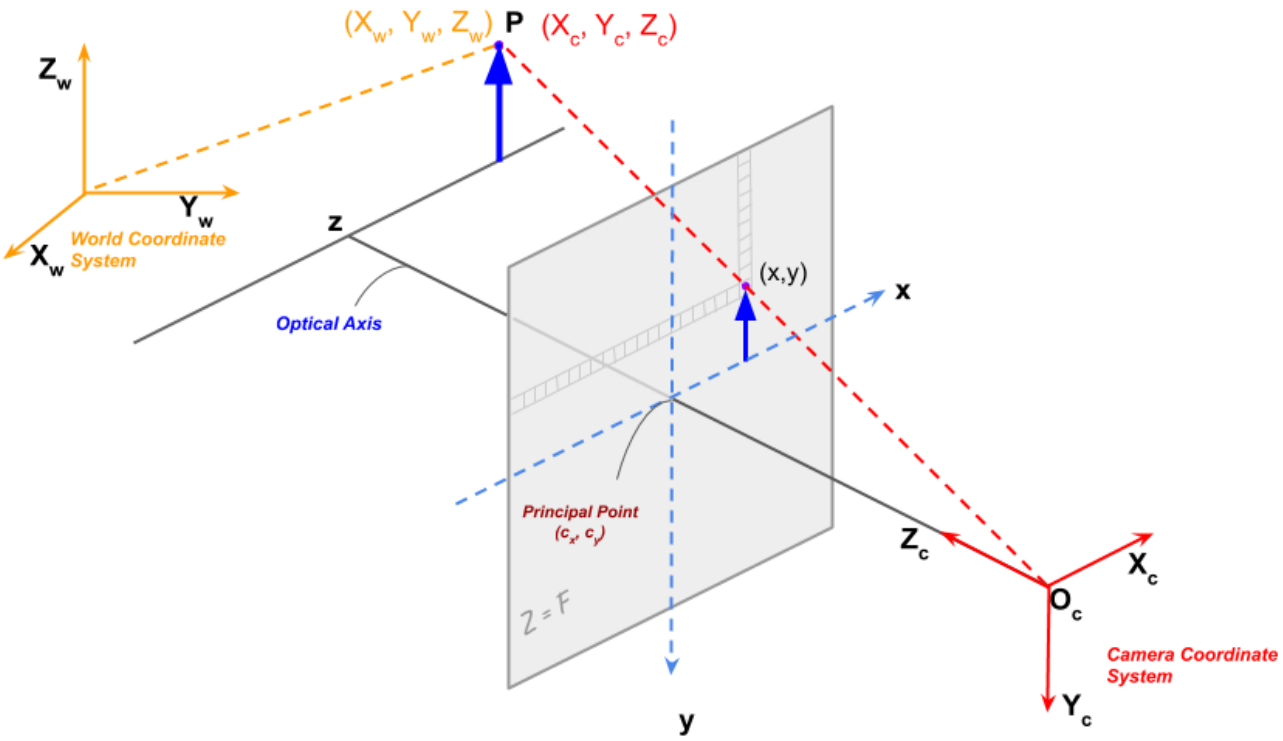

Рис. 8: Связь систем координат [12]

Модуль aruco предоставляет метод estimatePoseSingleMarkers для оценки положений всех обнаруженных маркеров в системе координат камеры. Принимая в качестве входных параметров углы маркеров, реальные размеры маркера, вектор коэффициентов искажения, а также параметры камеры, такие как матрицу калибровки камеры, состоящую из значений фокусных расстояний f_x и f_y, если пиксели в датчике изображения квадратные, то они будут равны, и координат оптического центра камеры (c_x, c_y).

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Данную матрицу и вектор коэффициентов искажения можно получить в ходе калибровки камеры с помощью OpenCv. Это означает, что будет получена вся информация о камере, необходимая для определения точной взаимосвязи между трехмерной точкой в реальном мире и ее соответствующей двухмерной проекцией (пикселем) на изображении, полученном этой откалиброванной камерой.

Калибровку камеры можно проводить несколькими способами, одним из которых является калибровка по шахматной доске, так как клетки доски легко различимы на изображении, а углы клеток удобно использовать для локализации, поскольку они имеют четкие границы.

В среде симуляции было размещено изображение шахматной доски на плоскости, далее были созданы изображения, использующиеся в качестве снимки камеры с разных ракурсов. Таким образом есть доска с известными нам размерами и координатами, теперь можно провести обнаружение углов шахматной доски с помощью встроенной функции OpenCv, которая вернет координаты четырех углов. После чего их можно уточнить и следующим шагом калибровки необходимо передать положения углов в мировых координатах и в координатах изображения в метод calibrateCamera.

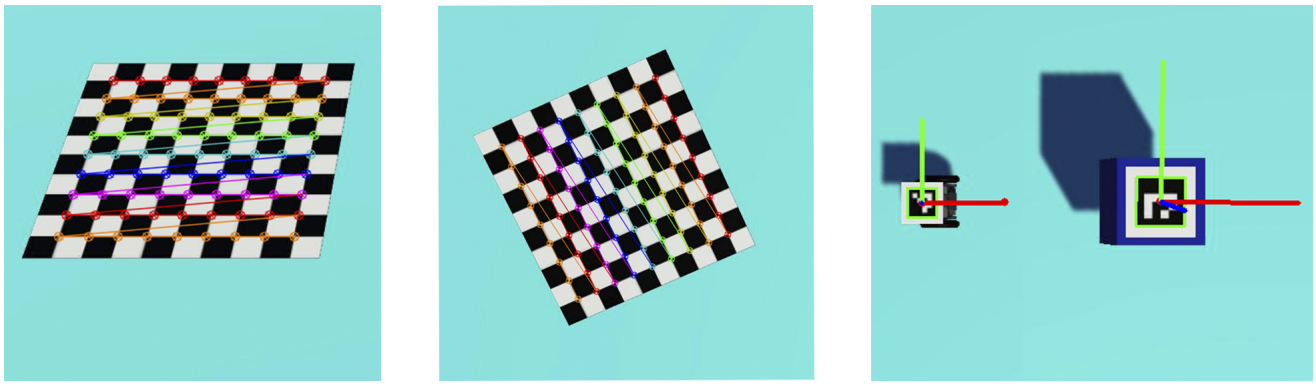

Рис. 9: Откалиброванные изображения шахматной доски 9x9 и ориентации осей маркеров

Вычислив все необходимые данные, можно воспользоваться методом для получения положения маркеров, который вернет нам векторы поворота

локальной оси координат относительно камеры, связанной с маркером, где ось x указывает на правую сторону маркера, ось y указывает в верх маркера, а ось z направлена от плоскости маркера, как показано на изображении 9, а также векторы смещения для каждого маркера относительно камеры.

Завершающим этапом является перенос и поворот систем координат объектов в локальную систему координат робота, после чего можно передать новые координаты цели в регулятор, описываемый формулами (6) и (8).

Исходный код всех реализованных программ можно посмотреть на сайте [13].

5.3 Отработка траектории движения робота

Отработка траектории движении робота осуществляется посредством работы программного комплекса в среде симуляции. Координаты цели позиционирования, преследуемой роботом, при достаточно малом перемещении за достаточно малое время могут описывать необходимую траекторию движения робота.

Таким образом для формирования набора точек в симуляции существует возможность первой итерацией построить траекторию движения посредством перемещения преследуемого объекта «куб», параллельно осуществляя запись его координат и момента времени перемещения отсчитываемого от момента запуска.

Второй итерацией является непосредственное чтение кортежа данных из сформированного в первой итерации или любым другим путем файла, отслеживание момента времени смены позиции и перемещение в необходимую точку, тем самым формируя траекторию, которой будет придерживаться робот во время движения.

Данный метод также применяется в следующей главе для проведения набора экспериментов с единой отслеживаемой траекторией.

Таким образом отработка траектории движения и задачи позиционирования на данный момент может осуществляться тремя основными способами:

- в среде симуляции напрямую, вручную определяя положение цели, перетаскивая «куб»;

- в среде симуляции с помощью чтения набора координат из файла;
- во внешней программе по распознаванию изображения (так же, перетаскивая "куб либо перемещая куб по траектории из файла).

Также для первых двух способов можно использовать два описанных выше управления (6)-(8) и (9). Во внешней программе управление осуществляется по формулам (6), (8).

Глава 6. Эксперименты

Для того, чтобы проверить корректность реализованного позиционирования в среде моделирования был реализован следующий сценарий. При инициализации симуляции создается объект «Куб», который можно перетаскивать во время исполнения с помощью курсора. Координаты этого объекта являются координатами цели в задаче позиционирования. Таким образом, задача робота — строго следовать за перемещением куба. Для проведения более честных экспериментов, траектория цели всегда считывается из одного и того же файла, соответственно цель передвигается всегда на одни и те же промежутки за одно и то же время. Также для некоторой численной оценки были посчитана метрика l_2, показывающая общее среднее отклонение от целевой траектории и вычисляемая по формуле:

$$\sqrt{\frac{1}{2n}\sum_{i=1}^{n}(x_i^* - x_i)^2 + (y_i^* - y_i)^2},$$

где n — количество точек каждой траектории, (x^*,y^*) — координаты цели, (x,y) — координаты робота.

На каждом из набора графиков синим обозначается траектория цели, оранжевым — робота. Зелеными и красными точками указаны начало и конец соответственно каждой траектории движения. Для каждого эксперимента представлены три графика: график траекторий, описываемый набором точек (x,y), графики зависимости координат от времени $x(t)$ и $y(t)$, для более подробного анализа.

Рассмотрим для начала сравнение управления (9) с разными коэффициентами, находящееся полностью на стороне среды симуляции, что облегчает получение текущих скоростей для формулы.

Как видно из рисунков (10) и (11), недостаточно большое значение коэффициентов в матрице, связанной с текущими скоростями, приводит к низко набираемой скорости и невысокой маневренности, из-за чего происходит большое отклонение от траектории, о чем также свидетельствуют значения метрик равные 0.87 и 0.65 соответственно.

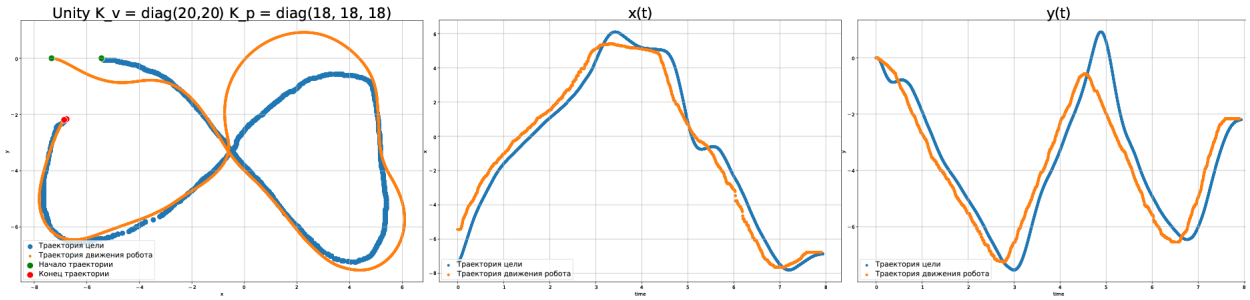

Рис. 10: Графики траекторий, $x(t)$, $y(t)$ для регулятора с обратной связью по состоянию (9). $K_v = diag(20{,}20)$, $K_p = diag(18{,}18{,}18)$

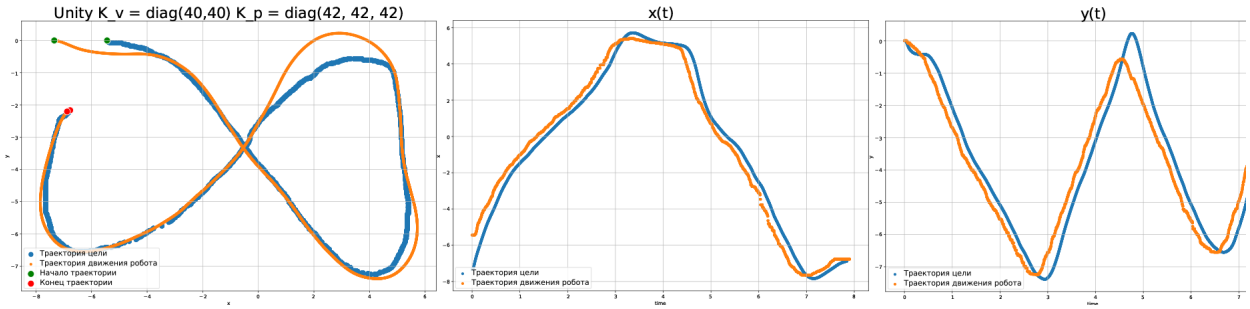

Рис. 11: Графики траекторий, $x(t)$, $y(t)$ для регулятора с обратной связью по состоянию (9). $K_v = diag(40{,}40)$, $K_p = diag(42{,}42{,}42)$

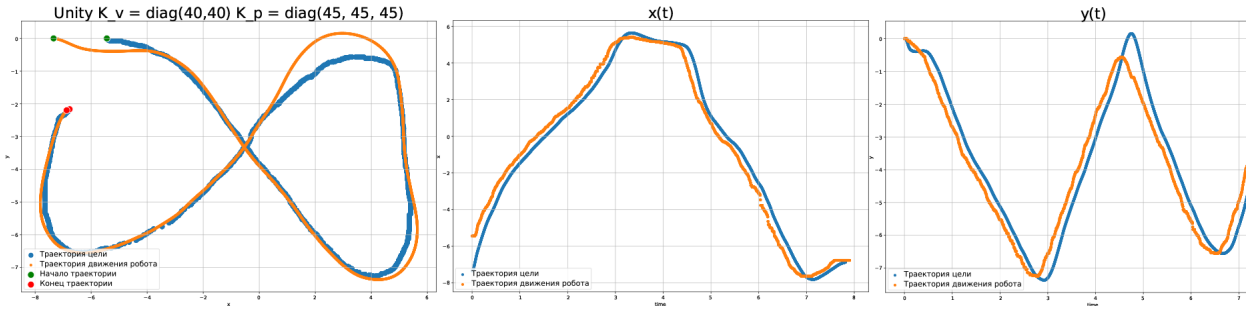

Рис. 12: Графики траекторий, $x(t)$, $y(t)$ для регулятора с обратной связью по состоянию (9). $K_v = diag(40, 40)$, $K_p = diag(45, 45, 45)$

В то же время отклонение уменьшается до 0.6, при уменьшении влияний текущих скоростей, зависящее от матрицы K_v, при увеличении значений матрицы K_p, что оказывает влияние на оценку расстояния до цели.

Рассмотрим далее сравнение управления линейными и угловыми скоростями, пропорциональными расстоянию до цели (6), (8). Данное управление показывает более удачные результаты при различных изменениях коэффициентов. Кроме того оно же используется для контроля робота по распознанному изображению, так как не требует вычисления текущих скоростей по изображению, которые могут искажаться в зависимости как от качества ви-

деооборудования, так и от погрешностей в данных, которые могут возникать из-за потери распознавания.

Проведем анализ данного регулятора в сравнении как с данными управления прямиком из среды симуляции, так и с управлением осуществляемым по изображению.

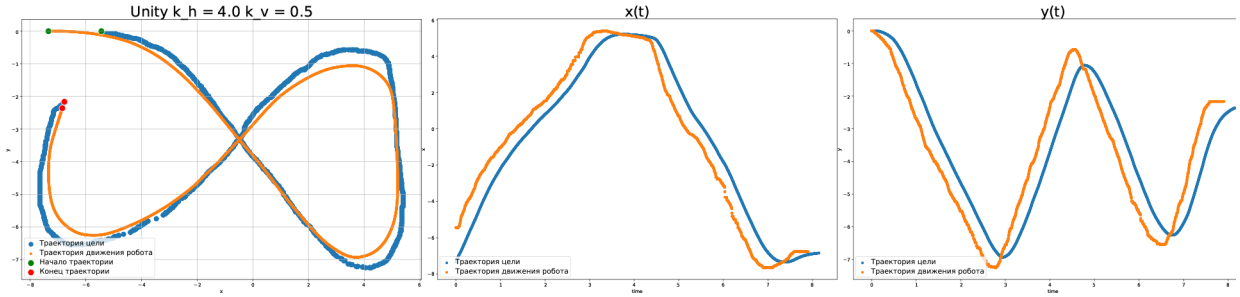

Рис. 13: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8)реализованного в среде симуляции Unity. $k_h = 4$, $k_v = 0.5$

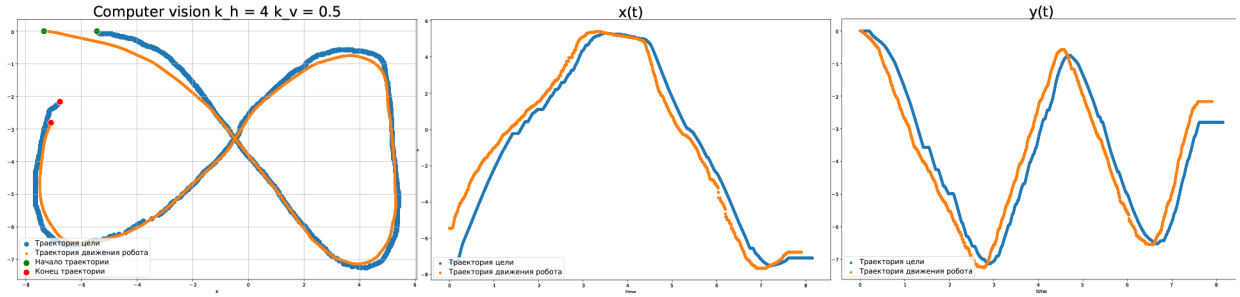

Рис. 14: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8) реализованного во внешней программе. $k_h = 4$, $k_v = 0.5$

Обращая внимание на рисунки (13), (15) ,(17) можно заметить, что небольшой коэффициент при угловой скорости по сравнению с коэффициентом у линейной скорости приводит к низкой маневренности робота, что также подтверждается самым большим значением метрики среди выборки данного управления: 1.07. Немного лучше становится при уменьшении скорости и увеличении параметра при угловой скорости, что можно заметить на изображении (15), робот набирает скорость медленнее и успевает аккуратнее поворачивать, благодаря чему отклонение от траектории становится меньше: 0.8. Самый лучший результат для данного метода управления среди проведенных экспериментов равный 0.58 достигается благодаря как увеличению коэффициента $k_h = 4{,}5$ при линейной скорости, так и увеличению коэффициента $k_v = 1$ при угловой скорости, таким образом робот хорошо набирает

скорость, не отставая далеко от робота и соответственно благодаря этому также хорошо поворачивает.

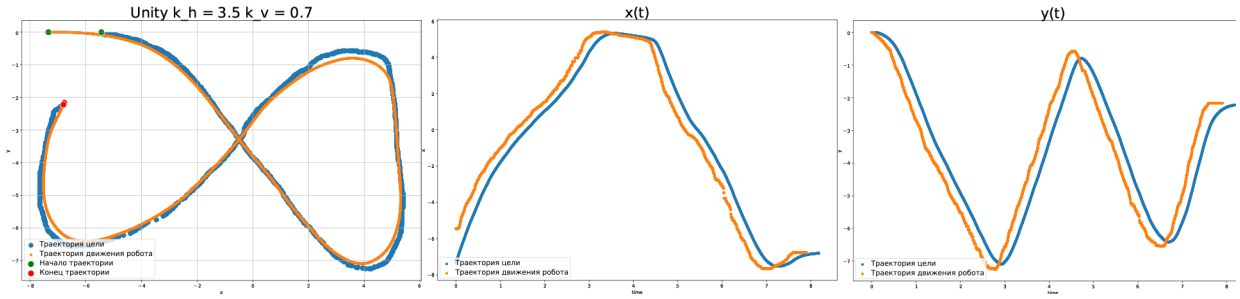

Рис. 15: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8) реализованного в среде симуляции Unity. $k_h = 3.5$, $k_v = 0.7$

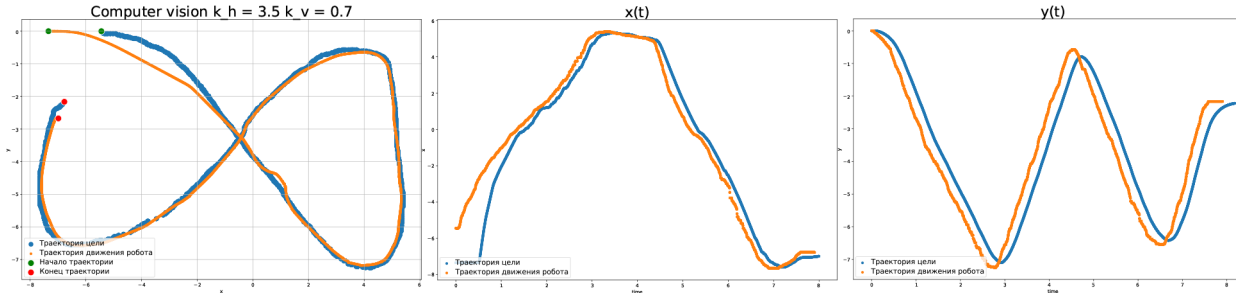

Рис. 16: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8) реализованного во внешней программе. $k_h = 3.5$, $k_v = 0.7$

Рассматривая графики связанные с внешним регулятором стоит обратить внимание на некоторые отклонения, обусловленные различными причинами. На изображениях (16), (14) построенных по данным управления во внешней программе можно заметить большое начальное отклонение от траектории, связанно с задержкой запуска внешней программы и отправки управления, в то время как цель уже успевает переместиться. На изображении (18) данное отставание нивелируется быстрым набором скорости. Также на изображениях (16), (18) можно наблюдать небольшие всплески, возникшие из-за того, что при достижении роботом цели, маркеры иногда могут пересекаться и программа утратит возможность распознать маркер робота, в таком случае на робота отправляется последняя вычисленная скорость. Когда цель меняет местоположение, снова происходит обнаружение всех маркеров и робот возвращается на необходимую траекторию. Кроме того, можно заметить, что траектория робота, регулируемого внешним управлением, является менее гладкой, что, очевидно, можно объяснить постоянными поиском

и распознаванием маркеров по поступающему изображению, а также небольшой задержки связи, ввиду чего могут происходить некоторые погрешности, незначительно влияющие на движение робота.

Таким образом, ожидаемо, отработка траектории по изображению происходит с некоторыми потерями в точности, но это решается более тонкой настройкой для конкретной ситуации, а также не мешает решению задачи позиционирования.

Проводя сравнение управления из среды симуляции и из внешней программы, значения метрики показывают практически аналогичную ситуацию. Коэффициенты, позволяющие роботу наиболее точно отслеживать траекторию равны $k_h = 4{,}5$ и $k_v = 1$, в данном варианте значение метрики будет равно 0.61. На изображении (16) представлен самый худший вариант по значению метрики, так как из-за сравнительно небольшого коэффициента набора скорости, робот медленно нивелирует разрыв в расстоянии из-за описанной выше проблемы.

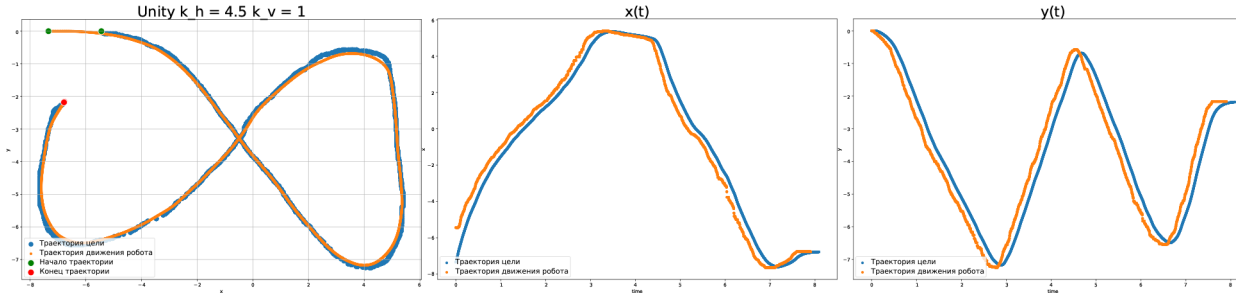

Рис. 17: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8) реализованного в среде симуляции Unity. $k_h = 4.5$, $k_v = 1$

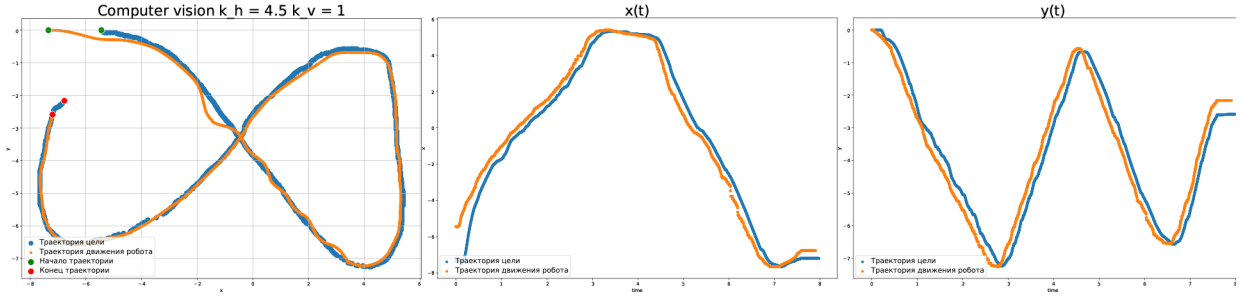

Рис. 18: Графики траекторий, $x(t)$, $y(t)$ для пропорционального регулятора (6), (8) реализованного во внешней программе. $k_h = 4.5$, $k_v = 1$

Таким образом, лучшие коэффициенты регулятора среди проведенных экспериментов для регулятора с обратной связью по состоянию явля-

ются две диагональные матрицы со значениями $K_v = diag(40, 40)$, $K_p = diag(45, 45, 45)$. Для пропорционального регулятора как во внешней программе, так и в среде симуляции, коэффициентами, обеспечивающими более точную отработку траектории по значениям метрики, являются $k_h = 4{,}5$, $k_v = 1$.

Выводы

В работе рассмотрена задача реализации системы управления двухколесным мобильным роботом в среде симуляции Unity3D. Представлена математическая модель робота, а также регуляторы для отработки движения по траектории в двух режимах: с использованием точных координат робота в обратной связи и с использованием положения робота, оцененного по результатам распознавания изображений с виртуальной камеры. Также проведены эксперименты с отработкой траектории в различных режимах. Анализ полученных графиков позволяет сделать вывод о том, что использование точных координат естественным образом дает более лучшую точность при движении по заданной траектории, однако в целом использование внешней камеры приводит лишь к незначительному ухудшению качества управления.

За счет своей структуры полученная система управления может быть легко перенастроена для управления реальным мобильным роботом с внешней стационарной камерой, что является темой дальнейших исследований. При этом использование ArUco-маркеров и стационарной камеры позволяет в общем случае рассмотреть задачу с несколькими роботами и статическими или динамическими препятствиями.

Заключение

В ходе работы были получены следующие результаты:

- реализована симуляционная модель движения мобильного двухколесного робота с дифференциальным приводом в среде Unity;
- организована коммуникация модели в Unity с внешней программой для управления с использованием информации с изображений;
- синтезированы регуляторы для отработки движения по траектории в двух режимах управления, выбран наиболее оптимальный регулятор;
- работоспособность системы продемонстрирована на примере нескольких симуляционных экспериментов.

Отметим, что в работе применены достаточно простые регуляторы. В дальнейшем планируется использовать полную информацию о траектории (скорости и ускорения) в обратной связи, а так же попробовать более сложные законы управления, в том числе, с учетом запаздывания управляющего сигнала. Помимо этого, особого внимания требует вопрос оценки скоростей по информации с видеокамеры ввиду неточностей распознавания. Также перспективным направлением является вопрос управления с визуальной обратной связью (visual servoing), который заключается в минимизации текущего и желаемого набора точек на изображении.

Список литературы

[1] Automatic generation and detection of highly reliable fiducial markers under occlusion / S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, M.J. Marín-Jiménez // Pattern Recognition. — 2014. — Т. 47, № 6. — С. 2280–2292. — Режим доступа: https://www.sciencedirect.com/science/article/pii/S0031320314000235.

[2] Budlov Egor O., Sevostyanov Ruslan A. Colour markers localization system / Ed. by Klyuev Vitaly, Pyshkin Evgeny, Natalia Bogach. — United States : Association for Computing Machinery, 2018. — Nov. — P. 143–145.

[3] Carona Ricardo, Aguiar A. Pedro, Gaspar José. CONTROL OF UNICYCLE TYPE ROBOTS Tracking, Path Following and Point Stabilization. — 2008. — 11.

[4] Corke Peter. Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised. — Springer, 2017. — Т. 118.

[5] Herbrechtsmeier Stefan, Witkowski Ulf, Rückert Ulrich. Bebot: A modular mobile miniature robot platform supporting hardware reconfiguration and multi-standard communication // FIRA RoboWorld Congress / Springer. — 2009. — С. 346–356.

[6] Ha Sehoon, Xu Peng, Tan Zhenyu и др. Learning to Walk in the Real World with Minimal Human Effort. — 2020. — 2002.08550.

[7] Papcun Peter, Zolotova Iveta, Tafsi Karim. Control and teleoperation of robot khepera via android mobile device through bluetooth and wifi // IFAC-PapersOnLine. — 2016. — Т. 49, № 25. — С. 188–193.

[8] Siegwart Roland, Nourbakhsh Illah R., Scaramuzza Davide. Introduction to Autonomous Mobile Robots. — 2nd изд. — The MIT Press, 2011. — ISBN: 0262015358.

[9] Veremey Evgeny I. Dynamical Correction of Positioning Control Laws // IFAC Proceedings Volumes. — 2013. — Т. 46, № 33. — С. 31–36. —

9th IFAC Conference on Control Applications in Marine Systems. Режим доступа: https://www.sciencedirect.com/science/article/pii/S147466701646129X.

[10] Документация Unity по Wheel Collider. — Режим доступа: https://docs.unity3d.com/ru/current/Manual/class-WheelCollider.html (дата обращения: 2021-05-24).

[11] Обнаружение маркеров с помощью OpenCv. — Режим доступа: https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html (дата обращения: 2021-05-24).

[12] Описание связи систем координат изображения и камеры с геометрической точки зрения. — Режим доступа: https://learnopencv.com/geometry-of-image-formation/ (дата обращения: 2021-05-24).

[13] Репозиторий с исходным кодом. — Режим доступа: https://github.com/AnJoie/MultipurposeRobots (дата обращения: 2021-05-24).