Санкт-Петербургский государственный университет

МОИСЕЕВ Олег Сергеевич

Выпускная квалификационная работа

Программно-аппаратный комплекс копирующего управления робототехническими системами

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальные информатика и информационные технологии»
ООП СВ.5003.2016 «Программирование и информационные технологии»
Профиль «Автоматизация научных исследований»

Научный руководитель:
доцент, кафедра компьютерных технологий
и систем, к.ф.-м.н. Погожев Сергей Владимирович

Рецензент:
доцент, кафедра механики управляемого движения
и систем, к.ф.-м.н. Шиманчук Дмитрий Викторович

Санкт-Петербург
2020 г.

Содержание

Введение

Для любого робототехнического комплекса, будь то шестизвенный робот-манипулятор, работающий в недетерминированной среде, или автономная тележка, одним из лучших вариантов для управления является система копирующего управления с обратной связью — билатеральное управление. Одним из подходов для обучения робота действиям внутри детерминированной среды является задание координат точек, по которым ему необходимо пройти, однако более удобным является показ необходимых действий. Примером реализации данного метода являются модели, представленные компанией Rozum Robotics [1].

Подход с "показом необходимых действий" возможно использовать при наличии робота в доступном месте – то есть человеку необходимо находиться в непосредственном контакте с оборудованием. Однако, такое далеко не всегда возможно – например, в случаях использования роботов в недоступных для людей местах (пещеры, завалы, подводная среда и прочее), опасных средах или в космосе. Таким образом, важной задачей является дистанционное копирующее управление роботом, опционально – с наличием обратной связи. Сегодня для такого используются системы, состоящие из управляемого робота и управляющего устройства, представляющего из себя копию робота, которую можно двигать, тем самым задавая положение реального экземпляра. Однако, у таких систем есть ряд недостатков:

- высокая стоимость;
- отсутствие гибкости для различных моделей роботов;
- значительное время создания для конкретной модели робота;
- значительное время разворачивания системы.

Таким образом, важной задачей является разработка и создание прототипа системы копирующего управления, которая лишена данных недостатков. В данной работе проанализировано несколько методов, с помощью которых возможно спроектировать данную систему:

- метод, основанный на использовании инерциальной навигационной системы;
- метод, основанный на использовании системы технического зрения;
- метод, совмещающий два предыдущих.

Постановка задачи

Главной задачей является получение прототипа системы управления робототехнической системой, обладающей следующими характеристиками:

- относительно низкая стоимость экземпляра;
- высокая точность позиционирования;
- высокая скорость развертывания в любом месте;
- управление удаленной робототехнической системой по сети;
- гибкость – быстрая адаптация для любой модели манипулятора, вне зависимости от её характеристик;
- простота и дешевизна масштабирования.

Введение в предметную область

1. IMU (Inertial Measurement Unit) – инерциальное измерительное устройство.
2. OpenCV ArUco – библиотека для обнаружения квадратных опорных маркеров, разработанная Рафаэлем Муньосом и Серхио Гарридо.
3. Маркер ArUco – сгенерированный квадратный маркер, состоящий из широкой черной рамки и внутренней двоичной матрицы, которая определяет его идентификатор.

4. Дисторсия – оптический эффект искривления линий на кадре по мере удаления от оптического центра.

5. Roll, pitch, yaw – углы Эйлера, авиационные углы: крен, тангаж и рысканье.

6. Фильтр – это алгоритм обработки данных, сглаживающий и убирающий шумы и лишнюю информацию (например, выбросы в данных).

Обзор литературы

Датчики, которые описаны далее, используются в различных системах инерциальной навигации уже длительное время. Существует ряд алгоритмов, которые используют данные, получаемые от IMU датчиков для вычисления собственных углов Эйлера, например [2], [3], [4], [5]. Данные методы различаются сложностью реализации, требованиями к вычислительным мощностям, а также количеством областей, в которых возможно их применение. Например, фильтр Калмана возможно применять для работы с любыми системами, его возможно адаптировать для различного количества измерений. Фильтр Маджвика используется для фильтрации данных, получаемых с акселерометра и гироскопа (возможно добавить обработку магнитометра), и получения углов вращения с высокой точностью.

Отдельная задача – это отслеживание линейных перемещений в пространстве. Она имеет высокую сложность по причине неидеальной конструкции датчиков и сильную зависимость полученных результатов от скорости и точности вычислений. Исследованию данной задачи посвящены следующие статьи: [6], [8], [9].

Сейчас популярным методом решения задач, связанных со взаимодействием с окружающим миром, является использования систем технического зрения. Для его использования необходима калибровка используемых камер, что описано, например, в [10]. Для определения координат объектов в пространстве зачастую используются хорошо заметные метки, описанные в [11], [12].

Глава 1. IMU датчики

1.1 Постановка задачи

Необходимо изучить составные элементы IMU датчиков, их сильные и слабые стороны, известные подходы к использованию получаемых данных для вычисления позиции устройства. После чего создать прототип системы управления, использующий данные подходы и оценить результаты данной работы и определить дальнейшие действия.

1.2 Описание оборудования и программных средств

В исследовании был использован бескорпусный инерциальный датчик Pololu IMU v5 (рис. 1).

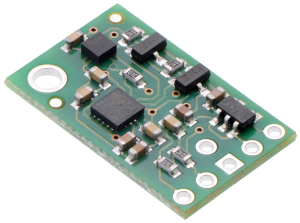


Рис. 1: Датчик Pololu IMU v5.

Данный сенсор состоит из 3-осевого гироскопа, 3-осевого акселерометра LSM6DS33 и 3-осевого магнитометра LIS3MDL, размещенных на плате размером 0,8"×0,5". Конструкция данных датчиков описана в [7]. Интерфейс I^2C обеспечивает доступ к девяти независимым измерениям: угловым скоростям, ускорениям и магнитному полю, которые можно использовать для расчета абсолютной ориентации датчика, а также к настройкам датчика.

В ходе изучения литературы и проведения ряда опытов с данными, полученными с каждого из датчиков по-отдельности, были определены слабые стороны каждой составляющей IMU модуля:

- угловые скорости в покое из-за физического несовершенства конструкции гироскопа ненулевые, а лишь колеблются около 0;

- получаемое ускорение датчика в покое колеблется около значения в 1g, а также включает в себя ускорение свободного падения, по причине чего нельзя использовать сразу полученные данные для вычисления положения датчика в прямоугольных координатах;

- на показания, получаемые с магнитометра влияет любой объект, находящийся рядом и искажающий магнитное поле: например, большой железный предмет в непосредственной близости с устройством или географическая аномалия.

Для того, чтобы получить угловое положение датчика в пространстве относительно его начального положения – его ориентацию – необходимо провести интегрирование значений угловых скоростей, полученных с гироскопа.

При интегрировании необработанных данных с датчика сталкиваемся с проблемой того, что они быстро накапливают погрешность из-за наличия двух видов сдвига в получаемых данных:

- статический – наличие постоянного сдвига в данных (причина – влияние факторов окружающей среды, например, температура);

- динамический – колебания данных около реального значения (причина – несовершенство конструкции реальных устройств).

И если от первого возможно практически полностью избавиться, вычитая перед началом работы усредненные данные, которые получены с покоящегося устройства, то со вторым сложнее – необходимо использование фильтров с нулевым сдвигом, которые будут работать в реальном времени.

В итоге было принято следующее решение: использовать модификацию фильтра Калмана – фильтр Маджвика, что позволило получить определение углового положения устройства в пространстве с высокой точностью (< 0,6 градусов среднеквадратичное отклонение положения в неподвижном состоянии; < 0,8 градусов среднеквадратичное отклонение положения в подвижном состоянии).

Аппаратная часть построена на основе платформы Arduino Uno с контроллером Atmega 328P и IMU модуля, связанного с ним через протокол

I^2C. На контроллере происходит вся работа по обработке данных с датчика и вычислению кватернионов вращения (либо углов Эйлера, но так как у них есть такое явление, как шарнирный замок – предпочтительнее использовать кватернионы для дальнейшей работы). Дальнейшая визуализация и обработка данных происходит на компьютере, к которому устройство подключено (рис. 2).

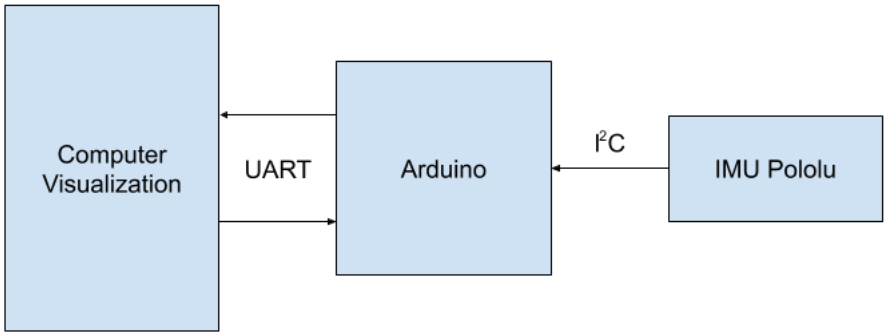


Рис. 2: Схема подключения датчика и контроллера.

1.3 Фильтр Маджвика

Сейчас наиболее популярным методом фильтрации данных, получаемых с IMU датчиков является фильтр Маджвика из-за своей скорости и точности, который будет рассмотрен подробнее.

Получить углы Эйлера возможно через кватернион вращения. Кватернион – четырехмерное комплексное число, которое используется для ориентации остроконечного тела в трехмерном пространстве. Рассмотрим систему отчета B по отношению к системе отсчета A. Обозначим за θ – угол поворота оси $A_{\hat{r}}$ в системе отсчета A (рис. 3). Кватернион, описывающий эту ориентацию, представлен формулой:

$$ {}^{A}_{B}\hat{q} = [q_1,\ q_2,\ q_3,\ q_4] = \left[\cos\frac{\theta}{2},\ -r_x\sin\frac{\theta}{2},\ -r_y\sin\frac{\theta}{2},\ -r_z\sin\frac{\theta}{2}\right], $$

где r_x, r_y, r_z – компоненты вектора $A_{\hat{r}}$ в системе координат A.

Углы Эйлера в свою очередь могут быть получены через компоненты

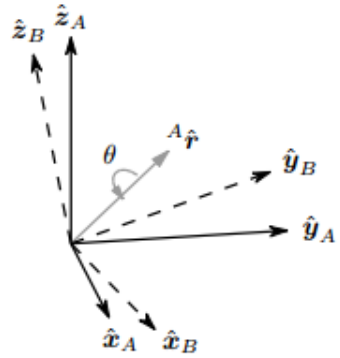


Рис. 3: Ориентация осей A и B.

кватерниона, используя формулы:

$$\psi = atan2(2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_2^2 - 1)$$
$$\theta = -\arcsin(2q_2q_4 + 2q_1q_3)$$
$$\phi = atan2(2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1)$$

Теперь, зная показания акселерометра (a_x, a_y, a_z), гироскопа $(\omega_x, \omega_y, \omega_z)$ и магнитометра (m_x, m_y, m_z), по формулам (1), (2) вычисляются два промежуточных кватерниона.

- Кватернион от гироскопа:

$$\overset{S}{E}\dot{q}_{\omega,t} = \frac{1}{2}\overset{S}{E}\hat{q}_{est,t-1} \otimes {}^S\omega_t, \tag{1}$$

  где $\overset{S}{E}\hat{q}_{est,t-1}$ – предыдущий результат оценки;
  ${}^S\omega_t = [0\ \omega_x\ \omega_y\ \omega_z]$ – значение угловой скорости гироскопа.

- Кватернион от акселерометра и магнитометра:

$$\overset{S}{E}\dot{\hat{q}}_{\epsilon,t} = \frac{\nabla f}{||\nabla f||}, \tag{2}$$

где

$$\nabla f = J_g^T * f_g + J_b^T * f_b,$$

$$f_g = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - a_x \\ 2(q_1q_2 + q_3q_4) - a_y \\ 2(\frac{1}{2} - q_2^2 - q_3^2) - a_z \end{bmatrix},$$

$$J_g = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix},$$

$$f_b = \begin{bmatrix} 2b_x(0.5 - q_3^2 - q_4^2) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z(0.5 - q_2^2 - q_3^2) - m_z \end{bmatrix},$$

$$J_b = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix},$$

$$b_x = \tfrac{1}{2}\sqrt{h_x^2 + h_y^2},$$

$$b_z = \tfrac{1}{2}h_z,$$

$$[0\ h_x\ h_y\ h_z] = {}^S_E\hat{q}_{est,t-1} \otimes {}^S\hat{m}_t \otimes {}^S_E\hat{q}^*_{est,t-1},$$

$${}^S\hat{m}_t = [0\ m_x\ m_y\ m_z],$$

Итоговый кватернион вращения вычисляется по формуле:

$${}^S_E q_{est,t} = {}^S_E\hat{q}_{est,t-1} + {}^S_E\dot{q}_{est,t-1}\,\Delta\,t,$$

где ${}^S_E\dot{q}_{est,t} = {}^S_E\dot{q}_{\omega,t} - \beta {}^S_E\hat{\dot{q}}_{\epsilon,t}$

Δt – время между замерами;

β – коэффициент погрешности измерений гироскопа.

Подробный вывод этих формул изложен в статье [2]. Таким образом, описано получение кватерниона вращения на основе данных, снятых с IMU датчика, далее из которого возможно вычисление углов Эйлера.

1.4 Описание исследований

Следующей задачей является определение положения объекта в пространстве в декартовых координатах. Для этого изначально происходила обработка данных с акселерометра, но у них есть один большой недостаток – необходимо производить вычитание вектора свободного ускорения для того, чтобы затем работать с чистыми ускорениями датчика. Но здесь возникла трудность, состоящая в том, что датчик выдаёт ускорения не в мировой системе координат, относительно Земли, а в подвижной, относительно себя. Поэтому вектор ускорения свободного падения необходимо вычитать, учитывая положение датчика в пространстве.

При данной операции точность определения положения системы в пространстве зависит от того, насколько точно и быстро будет вычтен этот вектор. Это обосновано тем, что для дальнейшего определения положения в пространстве используется двойное интегрирование данных ускорения, и даже малейшие отклонения при вычитании влияют на получение огромных отклонений итоговых значений координат.

Для того, чтобы упростить задачу, было принято решение уменьшить её размерность до 2х – навигация на плоскости. Это может быть применимо, например, для лабораторных исследований на автономных тележках, движущихся по ровному полу.

Датчик был установлен на шестое звено шестизвенного робота – манипулятора и произведено перемещение по тестовой траектории – квадрату 40х40 см. Данные были собраны, проанализированы и получены следующие результаты (рис. 4). В верхней части рисунка представлены графики ускорений по осям X (слева) и Y (справа). Синий цвет – “чистые” данные ускорения, красный – данные, прошедшие через фильтр бегущего среднего. В нижней части рисунка представлены графики полученных скорости (слева) и ускорения (справа). Красный цвет – данные по оси X, синий цвет – данные по оси Y.

Для получения данных результатов были собраны данные с акселерометра, расположенного во время опыта параллельно земле. Из них было вычтена медиана значений, снятых с покоящегося датчика, а также применен фильтр бегущего среднего с окном 109 элементов – при данном значении

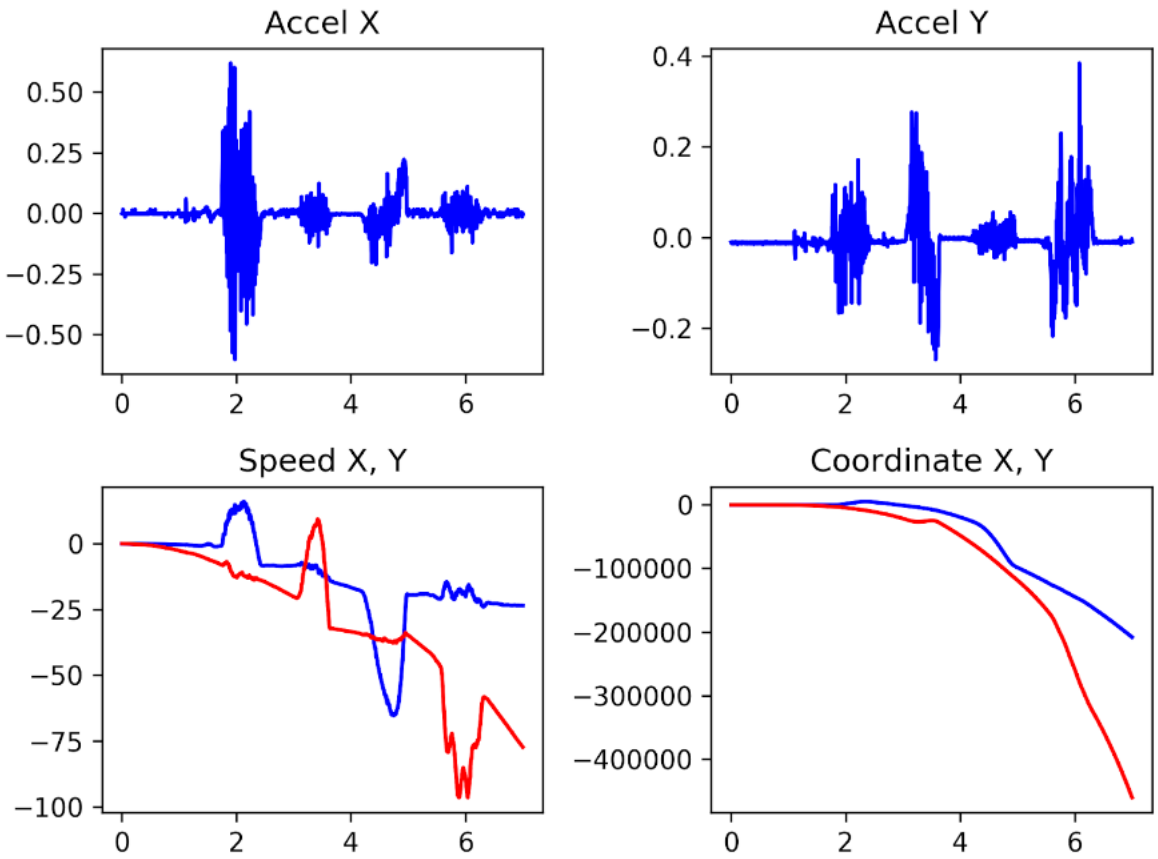


Рис. 4: Визуализация ускорения, скорости и перемещения.

ширины окна были получены наилучшие значения.

Собран прототип задающего устройства (рис. 5) на основе IMU датчика, платы Arduino и 8ми светодиодов – они используются для получения координаты Z.

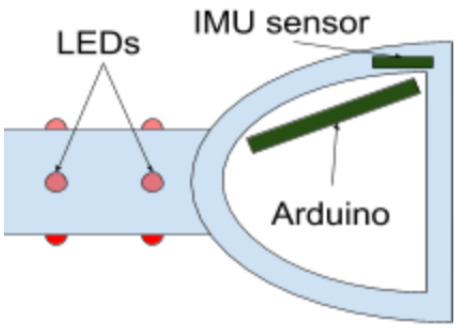


Рис. 5: Чертеж прототипа задающего устройства.

На основании информации об угле крена ручки пары светодиодов, уста-

новленных на внешней части рукоятки, переключаются, так, что всегда включается пара, направленная вниз в пределах границ 90-градусного сегмента (рис. 6).

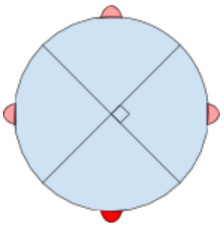


Рис. 6: Расположение светодиодов.

Используя специализированное программное обеспечение и камеру, установленную параллельно земле и направленную вверх, на компьютере рассчитывается декартовое положение задающей рукояти в пространстве на основе информации, полученной от датчиков и координат светодиодов в кадре. Используя кватернион вращения, полученный от IMU датчика, расположенного на рукоятке, рассчитаем вектор v, который указывает положение ручки в пространстве по формулам:

$$\overrightarrow{v_{start}} = (1, 0, 0),$$

$$\overrightarrow{tmp} = q_1 \cdot v_{start} \cdot q_1^{-1},$$

$$\overrightarrow{v} = \frac{\overrightarrow{tmp}}{\overrightarrow{|tmp|}}.$$

Тогда угол α между данным вектором v и его проекцией $proj$ на плоскость камеры (в частном случае на плоскость XY) рассчитывается по формуле:

$$\overrightarrow{\alpha} = \frac{\overrightarrow{v} \cdot \overrightarrow{proj}}{\left|\overrightarrow{v} \cdot \overrightarrow{proj}\right|}.$$

Внизу рабочей зоны находится камера, обнаруживающая рукоятку. Рабочая область оператора ограничена аппаратными характеристиками камеры – уг-

лом обзора и светочувствительностью. На кадре, полученном от камеры с задающим устройством в зоне обзора, есть две яркие точки с известными координатами относительно изображения (рис. 7).

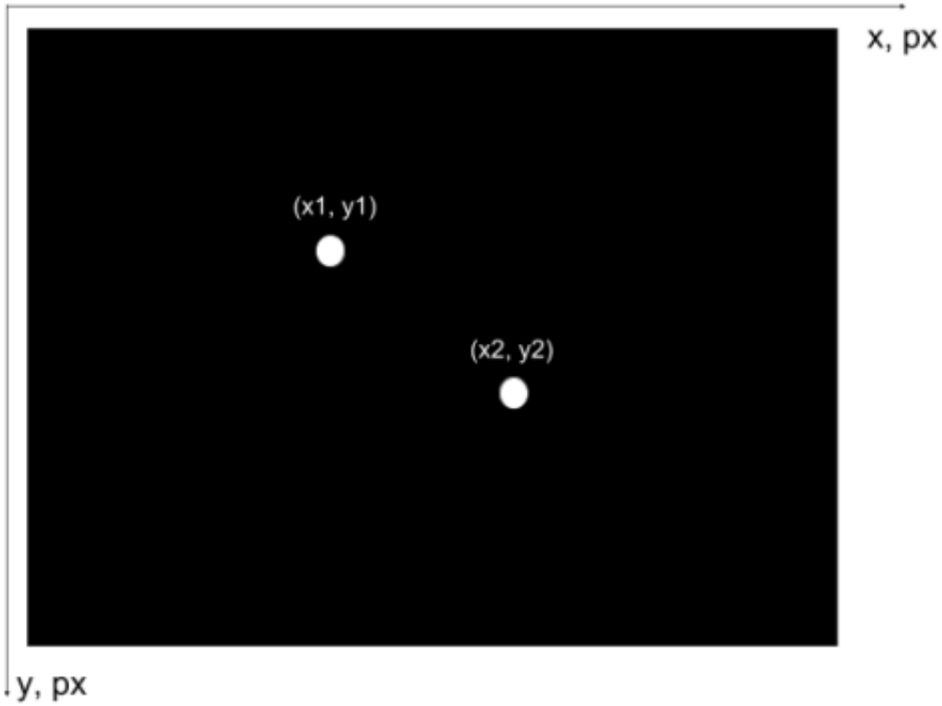


Рис. 7: Кадр после обработки.

Исходя из этого, расстояние между точками рассчитывается по формуле:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Данное расстояние является длиной вектора $proj$. Затем, используя угол между системой координат камеры и системой координат задающего устройства, вычисляется длина проекции $vector$ вектора расстояния между светодиодами, исключая поворот ручки в пространстве. Далее выполняется калибровка системы, состоящая в построении зависимости длины вектора между светодиодами на рамке от высоты рукоятки над камерой и получении графика с этими данными (рис. 8). Программный код данной калибровки приведен в GitHub репозитории https://github.com/OlegMoiseev/HeightCalibration.

При работе алгоритма высота ручки над камерой рассчитывается в соответствии с заданным графиком и длиной вектора d_{diods}.

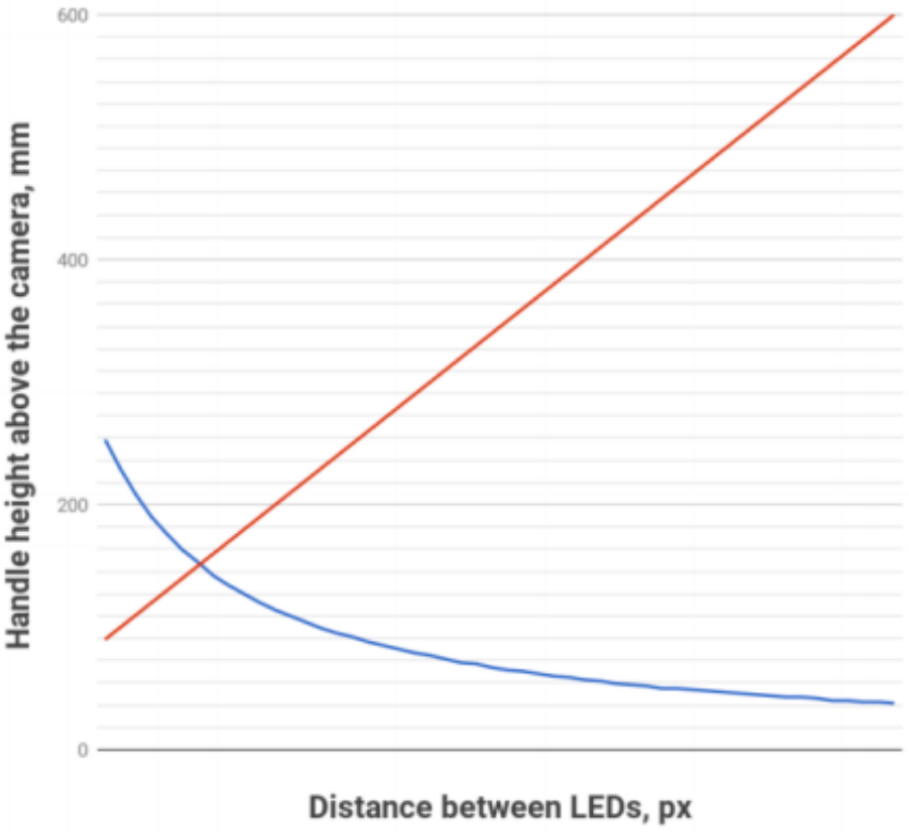


Рис. 8: Зависимость расстояния между светодиодами на кадре от высоты.

После вычисления шести координат рукоятки в пространстве поправляются декартовы координаты – смещение распознанной точки от окружности корпуса рукоятки к ее геометрическому центру. Для этого рассчитывается вектор переноса от светодиодов к центру: вектор, перпендикулярный вектору положения поворота ручки, повернутый на 0/90/180/270 градусов, в зависимости от того, в каком секторе горят светодиоды, а его длина равна по длине фактическому расстоянию между корпусом ручки и ее центром. Затем координаты этого вектора вычитаются из вычисленных координат ручки. Эта процедура необходима, чтобы при переключении светодиодов при вращении ручки не происходило скачка в координате и описанный путь передавался без разрывов.

1.5 Результаты опытов

На основе описанного подхода создан прототип задающего устройства, содержащий 3-х осевые гироскоп и акселерометр ST L3GD20H и 3-х осе-

вой магнитометр LSM303D, на котором был проверен описанный алгоритм (рис. 9).

Рис. 9: Созданное устройство.

Полученное устройство отвечает практически всем требованиям, поставленным в начале работы, но при этом имеет ряд недостатков:

- для масштабирования необходима установка дополнительных камер, что критично влияет на стоимость системы;
- система имеет удовлетворяющую точность только в довольно узком диапазоне высот (это обусловлено тем, что при малой высоте над камерой происходит её засветка, а при большой – камера может не распознать светодиоды);
- расположение камеры с направлением "вверх" не совсем удачное по причине того, что может происходить её засветка искусственными или естественными источниками освещения.

Реализация описанного ПО для ПК на языке C++ приведена в GitHub репозитории https://github.com/OlegMoiseev/Lopata, реализация ПО для платформы Arduino: https://github.com/OlegMoiseev/LopataSketch.

Глава 2. ArUco openCV

2.1 Постановка задачи

Необходимо учесть недостатки, описанные в предыдущей главе, и создать систему без них. Для этого было принято решение использовать библиотеку ArUco openCV для использования меток, с помощью которых будет происходить определение положения задающего устройства в пространстве.

2.2 Описание оборудования и программных средств

В новой системе с ПК будет связана только камера, исключается внешнее взаимодействие с IMU датчиком. Задающим устройством становится камера, с помощью которой происходит распознавание сетки меток (состоящую из любого количества меток больше 0) и определение положения задающего устройства относительно данного ориентира. Таким образом, поле действия данного устройства ограничивается только размером сетки меток, которую можно неограниченно увеличивать и устанавливать метки только туда, где необходимо обеспечить область действия системы.

2.3 Калибровка камеры

Для точного определения местоположения объекта на кадре необходимо провести калибровку камеры. Рассмотрим модель камеры, используемую при разработке систем технического зрения – камера обскура. В этой модели сцена формируется путем проецирования трехмерных точек на плоскость изображения с использованием преобразования перспективы:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

В данной формуле

- s – коэффициент масштабирования;

- X, Y, Z – координаты искомой точки в пространстве в мировой системе координат;
- u, v – координаты искомой точки на кадре в пикселях;
- c_x, c_y – координаты точки в центре кадра;
- f_x, f_y – фокусные расстояния камеры;
- $r_{11}, .., r_{33}, t_1, t_2, t_3$ – внешние параметры камеры, которые отражают движение камеры вокруг статичной сцены, либо движение объектов перед статичной камерой.

Описываемая модель представлена на рис. 10.

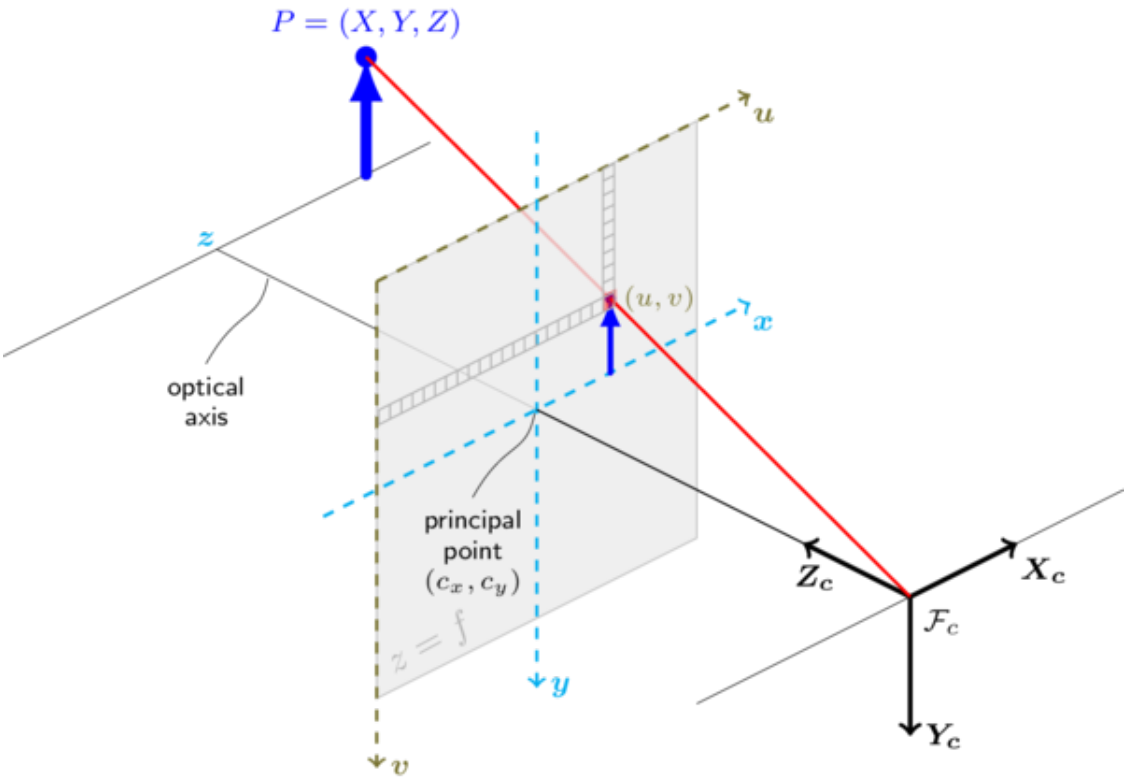


Рис. 10: Модель камеры-обскуры.

Реальные линзы, используемые в камерах, имеют радиальные искажения и небольшие тангенциальные искажения. Вышеприведенная модель (3)

расширяется формулами:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$

$$y' = y/z$$

$$r^2 = x'^2 + y'^2$$

$$x'' = x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2)$$

$$y'' = y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y'$$

$$u = f_x * x'' + c_x$$

$$v = f_y * y'' + c_y$$

Здесь переменные $k_1, .., k_6$ – радиальные коэффициенты дисторсии; p_1, p_2 – тангенциальные коэффициенты.

На рис. 11 представлены (слева направо) примеры отсутствия дисторсии, позитивная и негативная дисторсии.

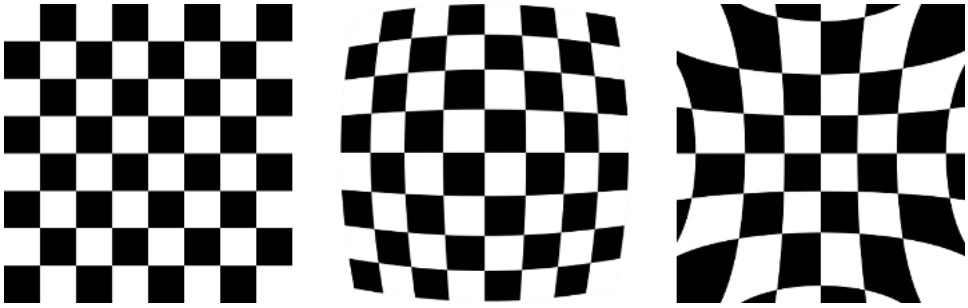


Рис. 11: Примеры дисторсии.

Калибровка в данном случае осуществлялась с помощью изображения шахматной доски (рис. 12) методами из библиотеки openCV.

Для процесса калибровки делается несколько кадров, на которых присутствует шахматная доска известных размеров. Происходит поиск углов клеток, после чего с ними связывается система координат. Результат поиска и отображения углов приведен на рис. 13.

Таким образом собирается информация обо всех полученных изображениях, после чего она передаётся в функцию cv2.aruco.calibrateCamera(),

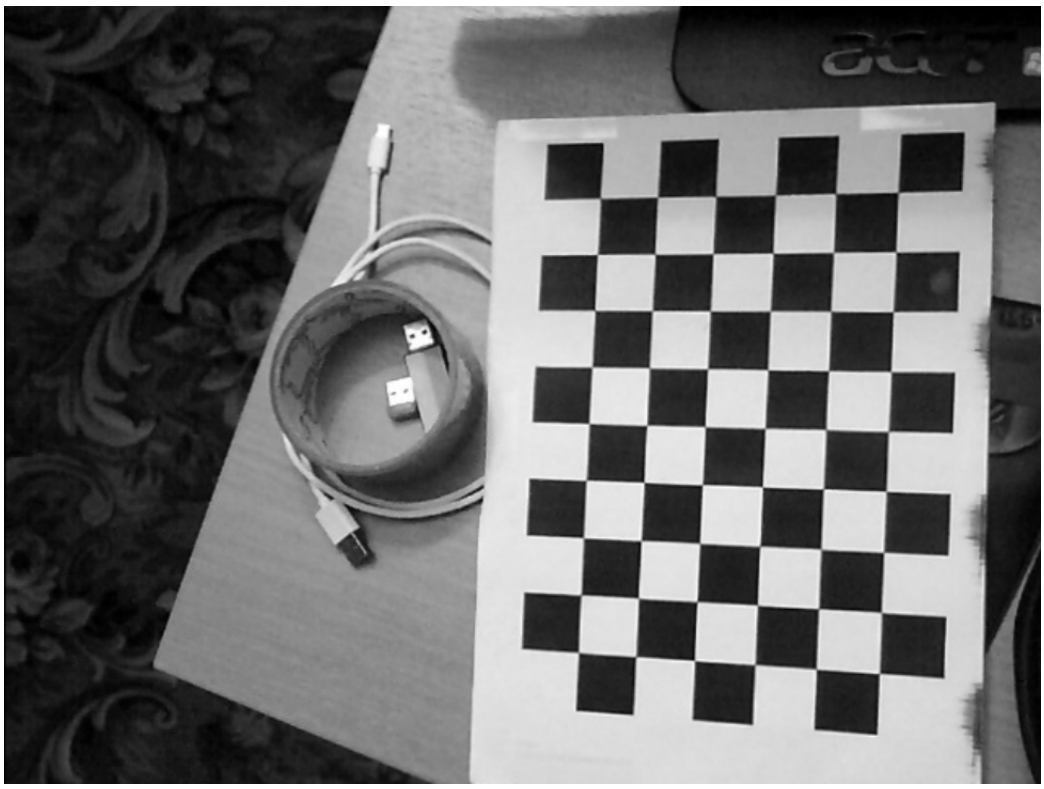


Рис. 12: Шахматная доска, на которой проводилась калибровка.

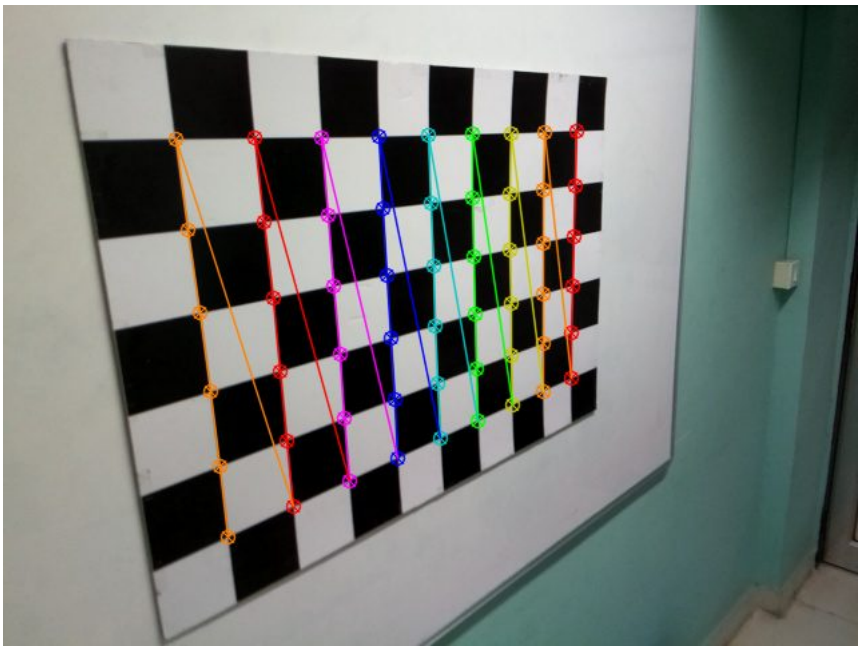


Рис. 13: Отображение найденных углов.

выходом которой являются внутренние параметры камеры.

В результате корректировки кадра на основе данных, полученных в

процессе калибровки, происходят следующие изменения получаемого изображения, что видно на рис. 14. Слева – кадр с камеры без обработки, справа – кадр после коррекции. Для ясности на левой стороне шахматной доски наложен красный отрезок, по которому можно видеть изменение формы доски на кадре.

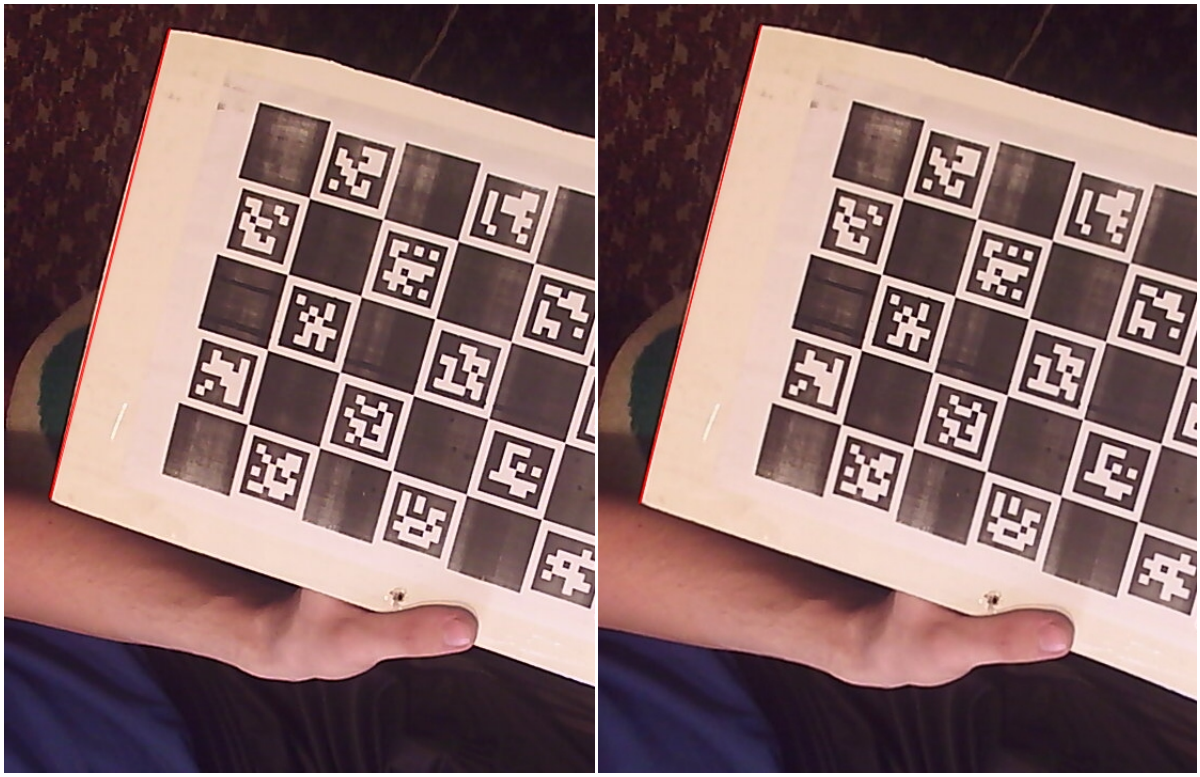


Рис. 14: Пример процесса undistortion.

2.4 ArUco маркеры

Маркер (рис. 15) состоит из внешней черной границы и внутренней части, в которой закодирован бинарный шаблон, являющийся уникальным идентификатором каждого маркера. В зависимости от словаря, на основе которого строятся маркеры, они могут содержать различное количество бит.

Процесс детектирования маркеров состоит из двух шагов:

1. Поиск "кандидатов" для маркеров. На изображении ищутся объекты квадратной формы, которые могут быть маркерами. Данный процесс начинается с адаптивного порогового фильтра, после чего происходит

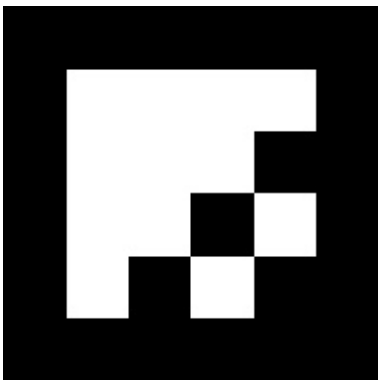


Рис. 15: ArUco маркер.

поиск контуров. Контура, которые не являются выпуклыми или приближенно квадратными, отбрасываются. Также применяется дополнительная фильтрация, например, удаление слишком маленьких или больших контуров.

2. После того, как "кандидаты" были найдены, необходимо проанализировать, являются ли они маркерами, с помощью их внутренней кодификации. Для этого первым этапом применяется перспективное преобразование для получение маркера в его канонической форме. После чего происходит пороговая обработка с использованием метода Оцу для разделения белых и черных битов. Изображение делится на ячейки в зависимости от размера маркера и границ, после чего происходит подсчёт черных и белых пикселей в ячейке для определения её цвета. В итоге полученные значения бит анализируются для определения принадлежности маркера определенному словарю.

Следующий необходимый шаг – это определение позиции камеры относительно маркера. Для определения положения камеры необходимы внутренние параметры камеры, определяемые в процессе калибровки.

Положение камеры относительно маркера – это преобразование из системы координат маркера в систему координат камеры, которое задаётся векторами поворота $\overrightarrow{rvec}$ и переноса $\overrightarrow{tvec}$. Поиск данных векторов осуществляется функцией cv2.SolvePnP(). Данная функция определяет положение объекта по заданному набору точек данного объекта, соответствующим проекциям

на изображении, а также матрице камеры и коэффициентам искажения. Точка в мировой системе координат проектируется на плоскость изображения, используя перспективную проекцию и внутренние параметры камеры:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

2.5 Описание исследований

За основу в данном подходе было взято распознавание ArUco меток и определение положения камеры относительно них.

Камера была установлена на шестом звене манипулятора и направлена вниз, а под ней расположена метка. После чего была произведена запись траектории при движении камерой по трём квадратам на разной высоте с фиксированным шагом (рис. 16).

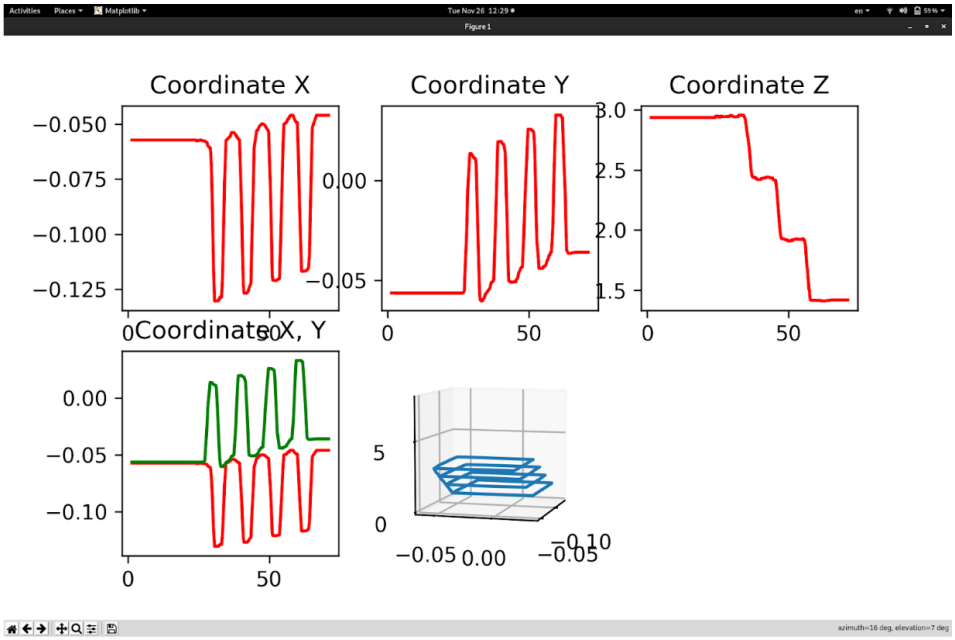


Рис. 16: Визуализация полученных с камеры данных.

Здесь приведены графики изменения координат по осям X, Y, Z по отдельности во времени (три графика сверху), график по осям X, Y вме-

сте (график снизу слева) и график по трём осям (посередине снизу). Была выявлена высокая чувствительность точности системы к повороту камеры относительно перпендикуляра к земле – на рис. 16 виден сдвиг квадратов в плоскости XY при изменении координаты z, который появился по причине наклона камеры при креплении.

Для учёта наклона камеры при принятии того, что маркеры считаются параллельными земле, использовался вектор вращения $\overrightarrow{rvec}$, получаемый после детектирования маркера методами openCV. В формуле (4) $rotation\_matrix$ получена из $\overrightarrow{rvec}$ посредством функции cv2.Rodrigues().

$$coordinates = rotation\_matrix \times \overrightarrow{tvec} \tag{4}$$

Таким образом, получены координаты камеры в системе координат метки. Однако, из-за высокой чувствительности систем компьютерного зрения к качеству освещения, при детектировании меток в реальном времени происходили "скачки" найденных контуров на кадре, что влияло на вычисляемые углы $roll, pitch$. Данные углы колебались в рамках $5°$, но данных колебаний было достаточно для критичного отклонения итоговых координат.

Для демонстрации был проведен следующий эксперимент: камера установлена на поворотный модуль и сняты кадры при развороте на $180°$. Визуализация представлена на рис. 17.

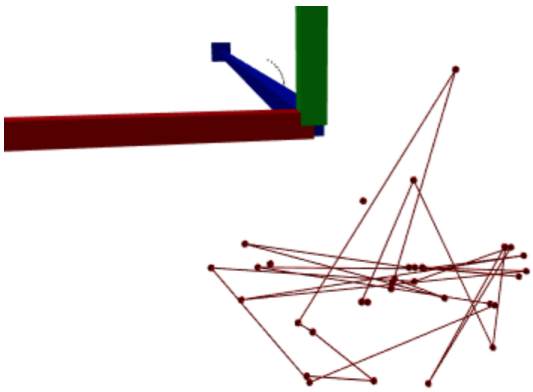


Рис. 17: Учёт трёх углов положения камеры.

На рис. 18 представлены графики изменений углов roll – зеленый, pitch – красный, yaw – синий.

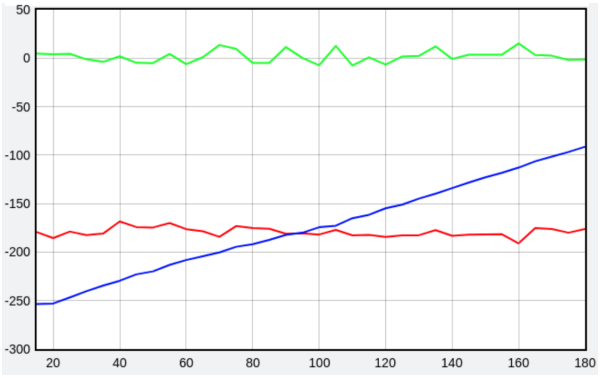


Рис. 18: Roll, pitch, yaw.

Для сравнения была произведена визуализация позиций камеры без учета её угла поворота, что видно на рис. 19.

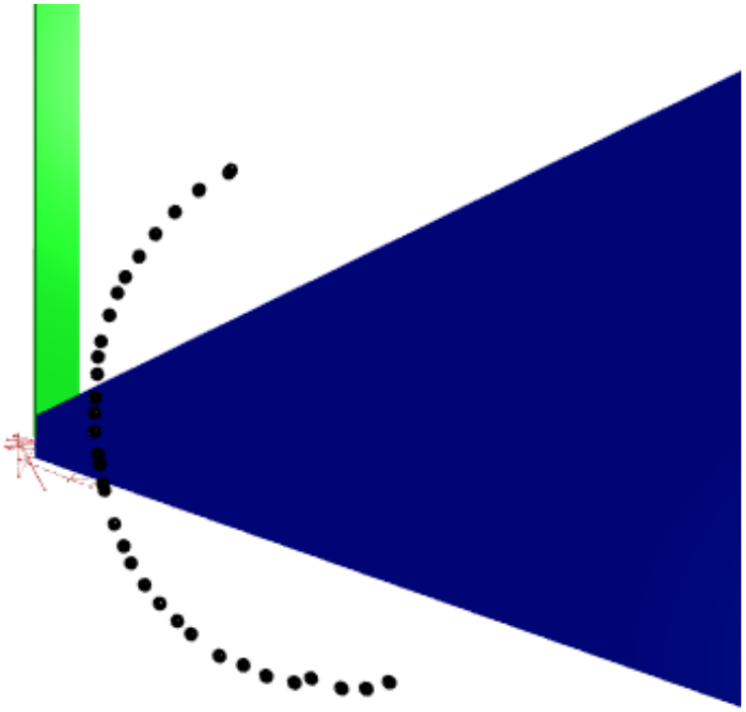


Рис. 19: Без учёта трёх углов положения камеры.

2.6 Результаты опытов

В результате тестов данной системы были выявлены некоторые недостатки. К ним относится следующее:

- низкая точность калибровки камеры – даже в лучших вариантах имелось некорректное определение координат x, y, z в пространстве (например, вычисленная координата z камеры порядка 7 м при фактической 1.28 м)

- определение углов $roll, pitch$ камеры с помощью метки и корректировка декартовых координат с их помощью не представляется возможным по причине неточности их определения.

Таким образом, необходимо доработать систему так, чтобы воспользоваться преимуществами точного определения координат $\overrightarrow{tvec}$ и угла поворота yaw каждой метки с помощью компьютерного зрения, а также точностью определения углов поворота камеры с помощью IMU датчика.

Глава 3. Комплексный подход

3.1 Постановка задачи

При изучении IMU датчика и распознавания ArUco маркеров, были выявлены сильные и слабые стороны обоих подходов. С помощью датчика с применением фильтра Маджвика для получаемых измерений возможно точное определение углов позиционирования объекта, а с помощью камеры – прямоугольных координат меток. Таким образом, для достижения поставленных целей необходимо создать систему, в которой с помощью камеры и меток будут определяться декартовые координаты задающего устройства, и с помощью точного определения углов камера будет позиционироваться перпендикулярно земле. Таким образом, будут использоваться сильные стороны обоих подходов.

3.2 Описание оборудования и программных средств

На основе IMU датчика, фильтра Маджвика и аддитивных технологий был создан стабилизирующий двух-осевой подвес для камеры. CAD-модели представлены на рис. 20.

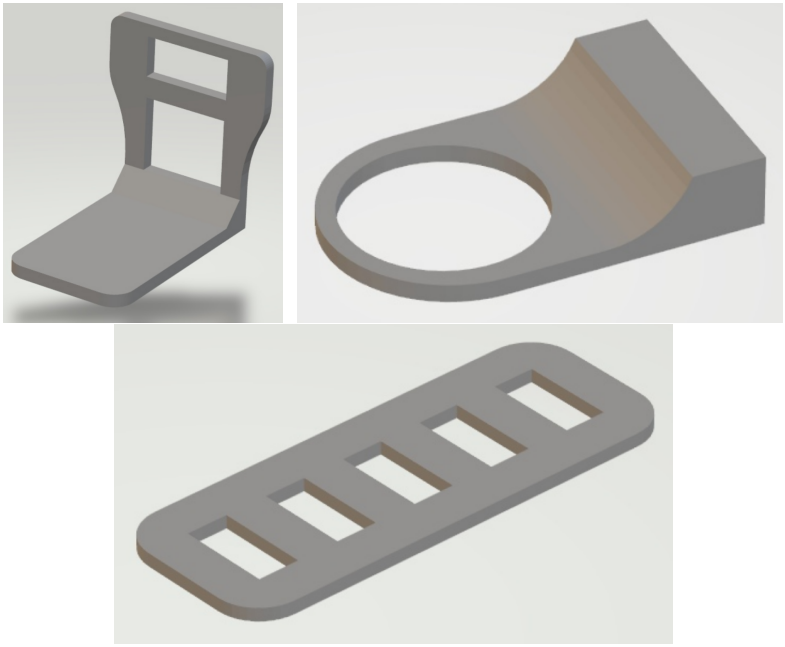


Рис. 20: CAD модели.

Схема подключения оборудования представлена на рис. 21.

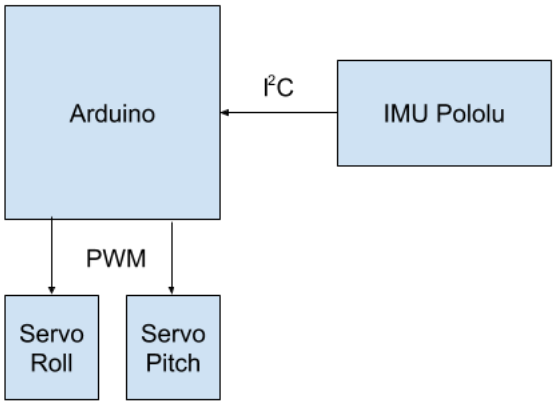


Рис. 21: Подключение оборудования.

Реальный экземпляр представлен на рис. 22.

Было разработано ПО, в котором определятся положение камеры относительно каждой метки, которая попадает в кадр. Затем происходит усреднение полученных значений и на основе этого вычисляется итоговое положение в пространстве относительно заданной точки $(0, 0, 0)$.

Метки на плоскости могут располагаться в любом порядке, необходимо знать только их координаты. Данные координаты подаются на вход программы

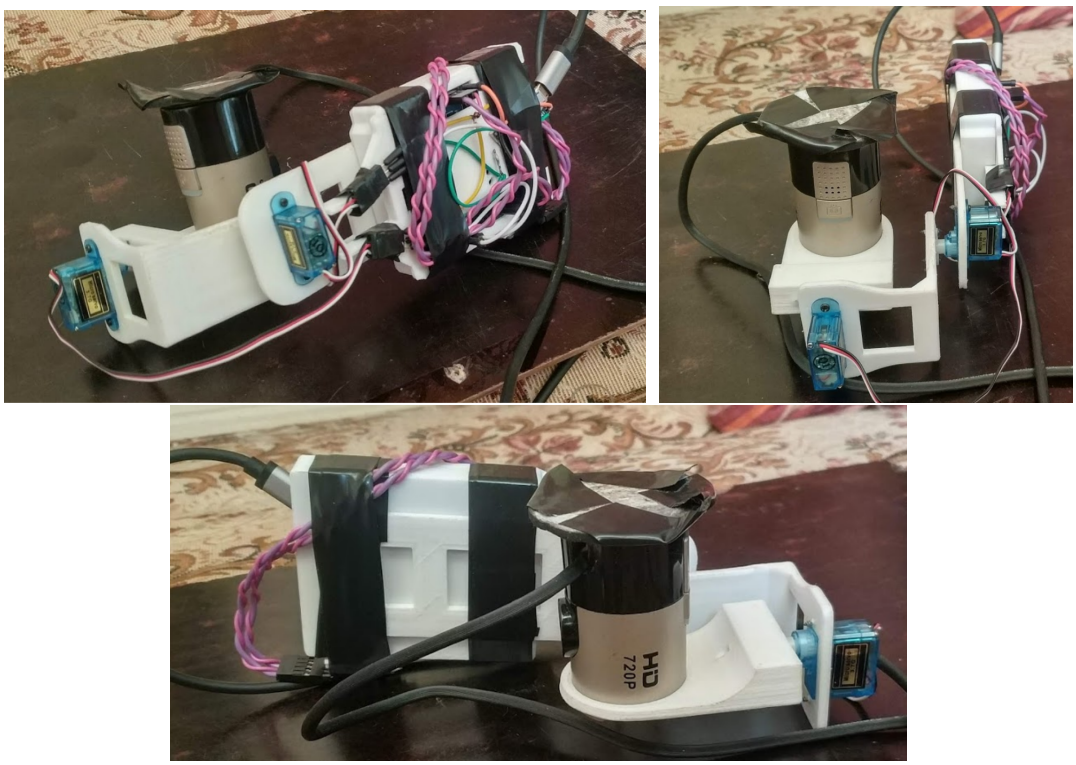


Рис. 22: Стабилизатор для камеры.

в виде словаря, где id метки сопоставляется её координата в пространстве. Пример расположения меток представлен на рис. 23.

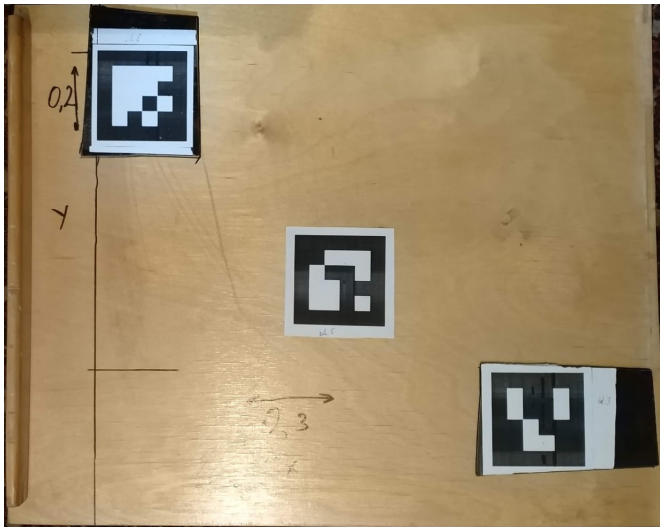


Рис. 23: Реальное расположение маркеров.

Правая нижняя ($id3$) имеет координаты $(0, 0)$, центральная ($id5$) имеет координаты $(0.15, 0.10)$, левая верхняя ($id8$) имеет координаты $(0.3, 0.25)$.

3.3 Описание исследований

В начале работы необходимо провести калибровку камеры, так как предыдущий метод калибровки не показал приемлемые результаты по точности. В текущий момент используем метод калибровки на основе ChArUco доски, представленной на рис. 24. Подробнее о данном методе калибровки написано в [10].

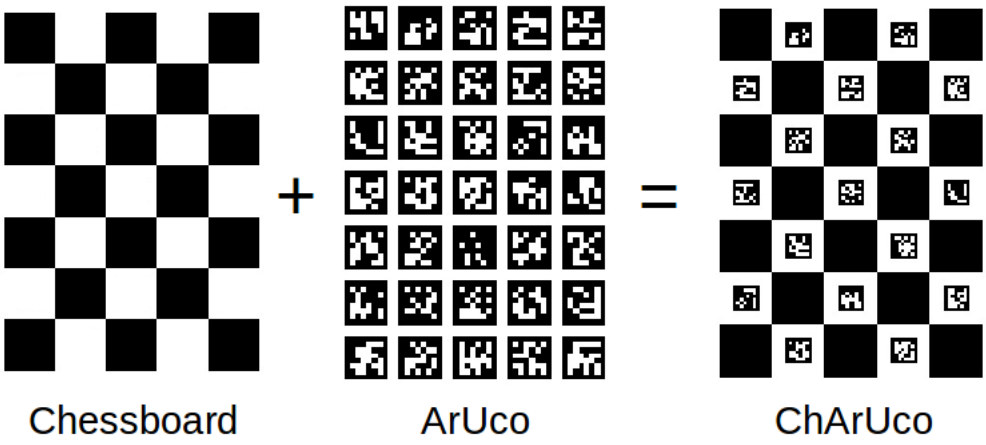


Рис. 24: Описание ChArUco Board из документации openCV.

Распознанные метки на калибровочном объекте данного вида представлены на рис. 25.

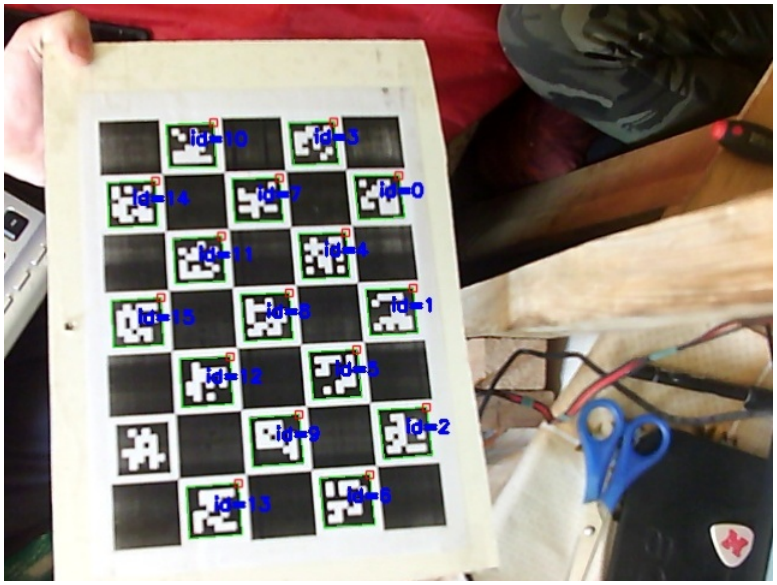


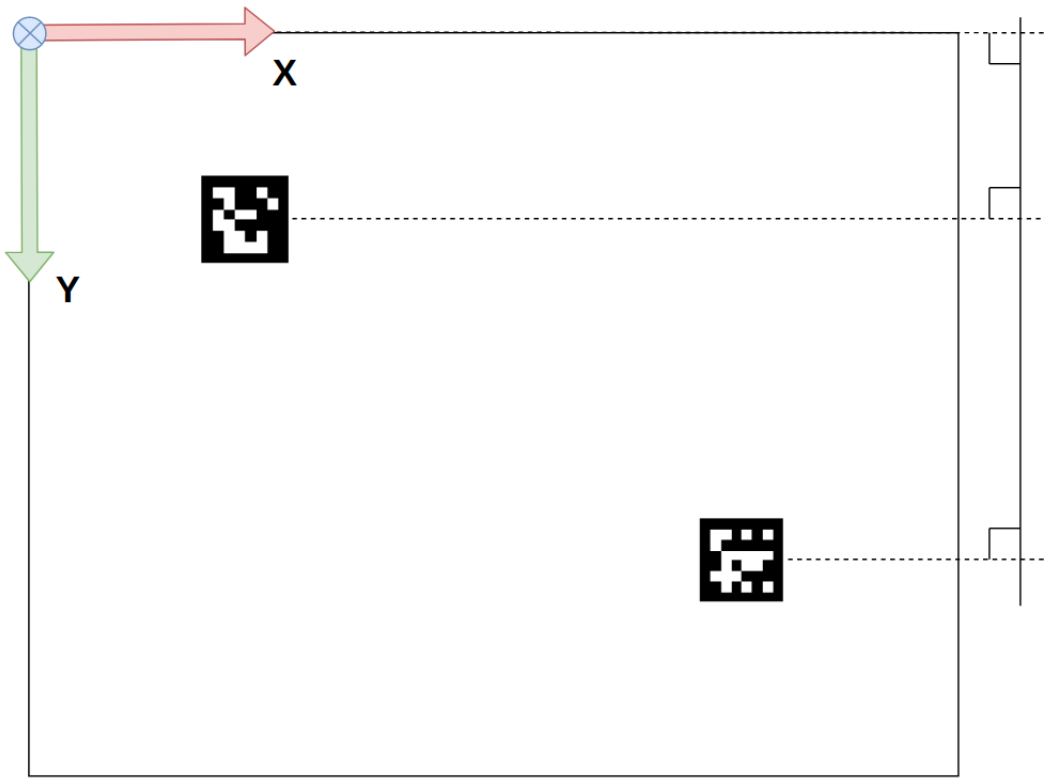
Рис. 25: Детектирование ChArUco board.

Камера, расположенная перпендикулярно земле, выдаёт кадры, на которых с помощью методов библиотеки openCV проводится поиск ArUco меток.

О найденной метке передаётся следующая информация:

- id – уникальный номер;
- $\overrightarrow{tvec}$ – вектор переноса системы координат камеры в систему координат метки;
- $\overrightarrow{rvec}$ – вектор поворота системы координат камеры в систему координат метки.

Для корректного определения вектора переноса с учетом возможности поворота камеры вокруг оси Z необходимо учесть угол рыскания камеры, что проиллюстрировано на рис. 26.



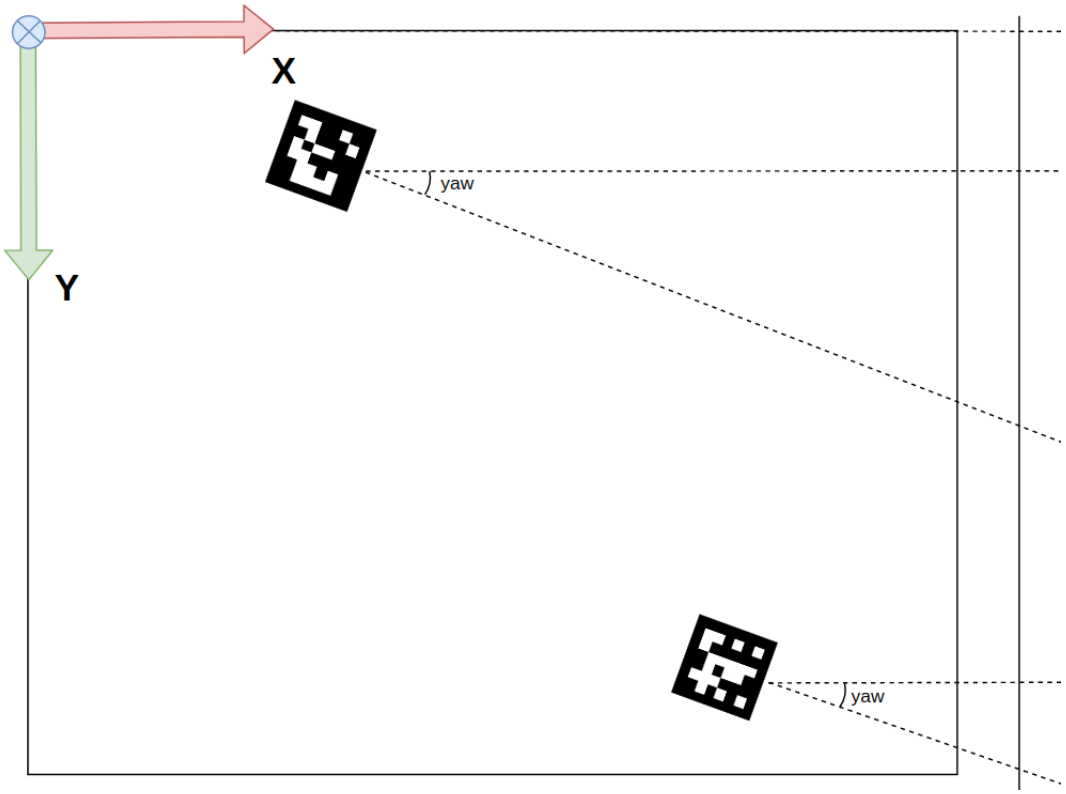


Рис. 26: Угол рысканья камеры.

Полученный $\overrightarrow{rvec}$ преобразуется в три угла Эйлера с помощью методов из библиотеки scipy.spatial.transform. Из полученных углов берётся угол рысканья – yaw, так как при текущем положении метки относительно камеры он, в отличие от двух других углов, определяется точно, не скачет и является корректным поворотом камеры. По формуле 5 строится матрица поворота вокруг оси Z, после чего положение камера относительно маркера вычисляется по формуле 6, где $markerPose$ – это заранее известные координаты маркера в мировой системе координат.

$$rotMat = \begin{pmatrix} cos(yaw) & -sin(yaw) & 0 \\ sin(yaw) & cos(yaw) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$camPose = tvec \times rotMat + markerPose \tag{6}$$

Данные операции проводятся для всех маркеров, которые попадают на кадр. В итоге на основе полученных координат камеры в системе координат

каждого из маркеров вычисляются координаты камеры в мировой системе координат. Так как точность определения координат относительно каждого маркера довольно высокая и разница между получаемыми результатами не превышает 5-8 мм, то итоговые координаты получаются усреднением полученных координат со всего кадра.

3.4 Результаты опытов

Для определения точности данной системы был проведён следующий опыт. Камера на стабилизирующем подвесе была закреплена на фиксированной высоте, после чего система была включена и произведено измерение во время перемещения камеры по отрезку на фиксированной высоте.

Маркеры были расположены на доске (рис. 27) в шахматном порядке для демонстрации отсутствия необходимости расположения маркеров в строго определенных положениях.

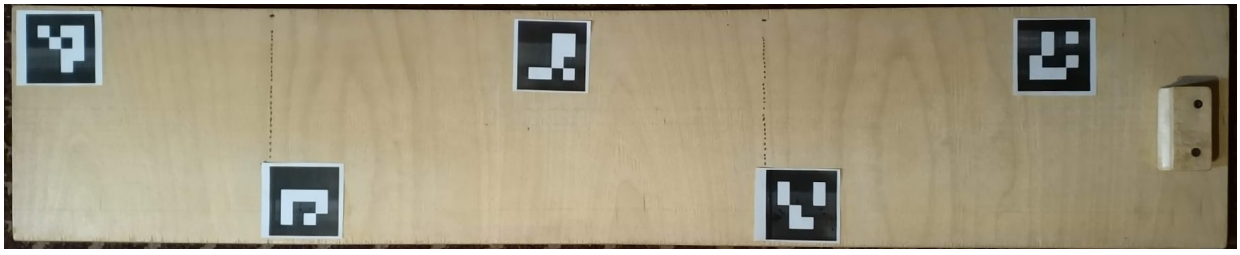


Рис. 27: Расположение маркеров для опыта.

Координаты каждого маркера известны и хранятся в словаре:

$$markers\_poses = \begin{aligned} &\{0 : [0, 0.15, 0], \\ &1 : [0.25, 0, 0], \\ &2 : [0.5, 0.15, 0], \\ &3 : [0.75, 0.0, 0], \\ &4 : [1.0, 0.15, 0]\} \end{aligned}$$

В результате были получены координаты в каждый момент времени данного эксперимента и проанализированы. Визуализация каждой точки была проведена с помощью библиотеки vpython (рис.28).

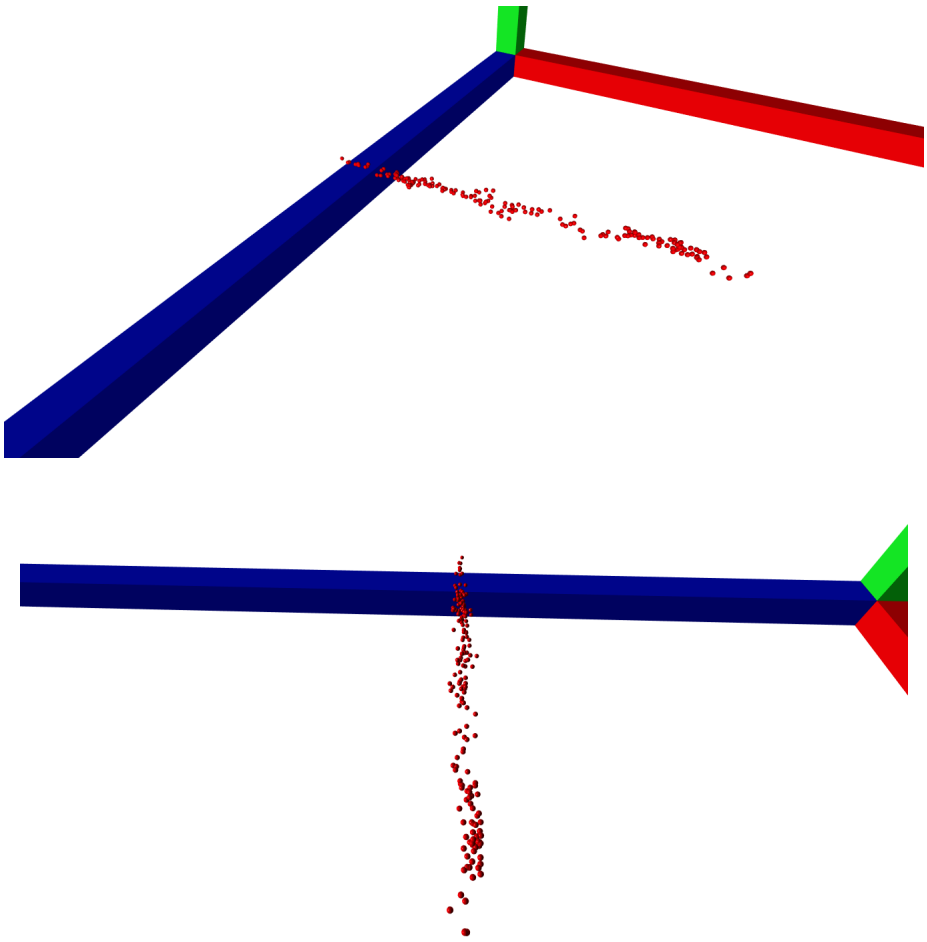


Рис. 28: Визуализация координат задающего устройства.

Исходя из данных точек, вычисленная длина перемещения составила 1.388 м., при этом фактическая – 1.4 м. Таким образом, погрешность измерения координат составляет 0.012 м. Данный показатель возможно улучшить, используя более качественные сервоприводы в конструкции стабилизатора камеры, а также более яркое и равномерное освещение.

В случае необходимости получения не только декартовых координат задающего устройства, а также его углов вращения, есть возможность подключения к стабилизатору камеры через один из доступных протоколов связи и считывать необходимые углы, которые затем передавать на исполнительное устройство.

Выводы

На основе композиции нескольких подходов к определению координат

объекта в пространстве была разработана и создана система позиционирования, которая решает поставленную задачу и отвечает поставленным требованиям. ПО, описанное в данной главе и ПО для калибровки камеры приведены в GitHub репозитории https://github.com/OlegMoiseev/lopata-research. Для решения поставленной задачи по управлению робототехническим комплексом используется TCP сервер, написанный на языке Karel для контроллера Fanuc R-30iA и управляющий роботом Fanuc M-20iA, приведен в приложении А и TCP клиент, код которого приведен в приложении В.

Заключение

В результате данной работы были изучены принципы работы акселерометра, гироскопа, магнитометра, подходы к фильтрации данных, в частности, методы вычисления углов вращения. Был создан прототип системы позиционирования, в основе которого находится IMU датчик. Затем определены его слабые стороны и изучены подходы к решению данной задачи посредством системы технического зрения и алгоритмов openCV, изучен вопрос калибровки камеры, построен прототип. Итогом работы стало создание третьего прототипа системы позиционирования, в основе которого лежат сильные стороны предыдущих двух подходов, и который отвечает всем поставленным требованиям к разрабатываемой системе в начале работы.

Список литературы

[1] Rozum Robotics [Электронный ресурс] – URL: https://rozum.com/

[2] An efficient orientation filter for inertial and inertial/magnetic sensor arrays: Rep. / University of Bristol; Executor: Sebastian O.H. Madgwick.

[3] S. O. H. Madgwick, A. J. L. Harrison and R Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," 2011 IEEE International Conference on Rehabilitation Robotics, Zurich, 2011, pp. 1–7, doi: 10.1109/ICORR.2011.5975346

[4] R. Mahony, Tarek Hamel, Jean-Michel Pflimlin. Nonlinear Complementary Filters on the Special Orthogonal Group. IEEE Transactions on Automatic Control, Institute of Electrical and Electronics Engineers, 2008, 53 (5), pp.1203–1217. doi: 10.1109/TAC.2008.923738

[5] Kalman, R.E. (1960). «A new approach to linear filtering and prediction problems». Journal of Basic Engineering 82 (1): pp.35–45

[6] Moiseev, O.S., Sarsadskikh, A.S., Povalyaev, N.D., Gorbunov, V.I., Kulakov, F.M., Vasilev, V.V. (2018). Force-sensed interface for control and training space robot. Представлено на THE EIGHTH POLYAKHOV'S READING: Proceedings of the International Scientific Conference on Mechanics. doi: 10.1063/1.5034735

[7] МЭМС акселерометры, гироскопы и геомагнитные датчики – революционно новый функционал потребительских устройств // Радиолоцман – июнь 2012. [Электронный ресурс] – URL: https://www.rlocman.ru/review/article.html?di=134058

[8] Моисеев О. С., Горбунов В. И. Система копирующего управления роботом-манипулятором // Control Processes and Stability vol.4. СПб.: Publishing house Fedorova G. V., 2017. С. 186–191.

[9] Joel Li, Van Yang Strapdown Inertial Navigation System Based on an IMU and a Geomagnetic Sensor // Analog Dialogue. 53-03, March 2019.

[10] An Gwon, Lee Siyeong, Seo Min-Woo, Yun Kugjin, Cheong Won-Sik, Kang Suk-Ju. (2018). Charuco Board-Based Omnidirectional Camera Calibration Method. Electronics. doi: 10.3390/electronics7120421

[11] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Generation of fiducial marker dictionaries using Mixed Integer Linear Programming, Pattern Recognition, Volume 51, 2016, Pages 481–491, ISSN 0031-3203, doi: 10.1016/j.patcog.2015.09.023

[12] Francisco J. Romero-Ramirez, R. Muñoz-Salinas, R Medina-Carnicer, Speeded up detection of squared fiducial markers, Image and Vision Computing, Volume 76, 2018, Pages 38–47, ISSN 0262-8856, doi: 10.1016/j.imavis.2018.05.004

Приложение

Глава А. Переход в заданную точку для робота Fanuc

```
ROUTINE point_input

VAR
    ctrl_in : INTEGER

BEGIN
    STATUS = SET_PORT_ATR(PORT_3, ATR_READAHD, 2)
    READ file_var (x)
    READ file_var (y)
    READ file_var (z)
    READ file_var(spd)
    READ file_var (ctrl_in::1)

    STATUS = SET_PORT_ATR(PORT_3, ATR_IA, 2)

    IF  ((UNINIT(x)) OR (UNINIT(y)) OR (UNINIT(z))
    OR (UNINIT(ctrl_in)) OR (UNINIT(spd))) THEN
        ctrl_in = 1
     ctrl = 1
    ELSE
        point_w.x = x
        point_W.y = y
        point_w.z = z
        point_w.w = 180
        point_w.p = 0
        point_w.r = 0
        IF ((spd>0) AND (spd<200)) THEN
           $SPEED = spd
        ENDIF
```

```
        CHECK_EPOS((point_w), $UFRAME, $UTOOL, STATUS)
        MOVE TO point_w
    ENDIF
END point_input
```

Глава В. Код клиента Python

```
import socket
import time


class Fanuc:
    def __init__(self, ip: str, port: int):
        self.socket = socket.socket()
        self.socket.connect((ip, port))

    def go_2_point_xyz(self, x, y, z):
        coord_str = (str(x) + ' ' + str(y) + ' ' + str(z) + ' 1').
        self.socket.send(coord_str)

    def go_2_point_xyzwpr(self, x, y, z, w, p, r):
        coord_str = (str(x) + ' ' + str(y) + ' '
                     + str(z) + ' ' + str(w) + ' '
                     + str(p) + ' ' + str(r) + ' 1').encode()
        self.socket.send(coord_str)

    def __del__(self):
        self.socket.close()


if __name__ == "__main__":
    ip_robot = "localhost"
```

```
port_robot = 9090
Fanuc = Fanuc(ip_robot, port_robot)
time.sleep(1)

for x in range(100, 1000, 100):
    Fanuc.go_2_point_xyz(x, 500, 900)
    time.sleep(0.5)
```