ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В. ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КУРСОВАЯ РАБОТА

«ИНТЕРФЕЙС ЗАПИСИ ВИДЕО НА RASPBERRY PI. РЕШЕНИЕ ЗАДАЧИ ПОЗИЦИОНИРОВАНИЯ ОБЪЕКТА С ПОМОЩЬЮ СИСТЕМ КОМПЬЮТЕРНОГО ЗРЕНИЯ»

Выполнил:
студент 204 группы
Маракулин Андрей Павлович

Научный руководитель:
к.ф.-м. н. доц.
Зубюк Андрей Владимирович

Москва
2020 год

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

Повсеместное распространение автоматизированных транспортных средств, дронов, квадрокоптеров, самоуправляемых машин ставит перед инженерами, математиками и физиками задачу определения их точного местоположения в пространстве. Для этого используются разнообразные датчики: системы глобального позиционирования GPS и ГЛОНАСС, ультразвуковые датчики, радары, лидары и видеокамеры. Последние являются наиболее распространёнными в качестве основных средств ориентации в окружающем пространстве. Однако, чтобы данные, принимаемые с видеокамеры, приобрели смысловую значимость для обрабатывающего их компьютера, нужно их интерпретировать. Этим занимается научная дисциплина под названием «Компьютерное зрение».

В данной курсовой работе рассмотрен процесс получения и записи видеопотока с камеры микрокомпьютером Raspberry Pi, реализованный на языке программирования Python. Распознавание заранее известных объектов с помощью библиотеки компьютерного зрения OpenCV, математическое описание методов распознавания, определение координат микрокомпьютера относительно этих объектов, а также создание локального веб-сервера для передачи всей необходимой информации на сторонние компьютеры с помощью Wi-Fi.

2. РЕАЛИЗАЦИЯ

2.1.Описание технической части

Для записи видео и обработки видеоинформации был выбран микрокомпьютер Raspberry Pi 3B. Благодаря компактности, он отлично подходит для поставленной задачи, его можно разместить на многих транспортных средствах, например, на квадрокоптере.

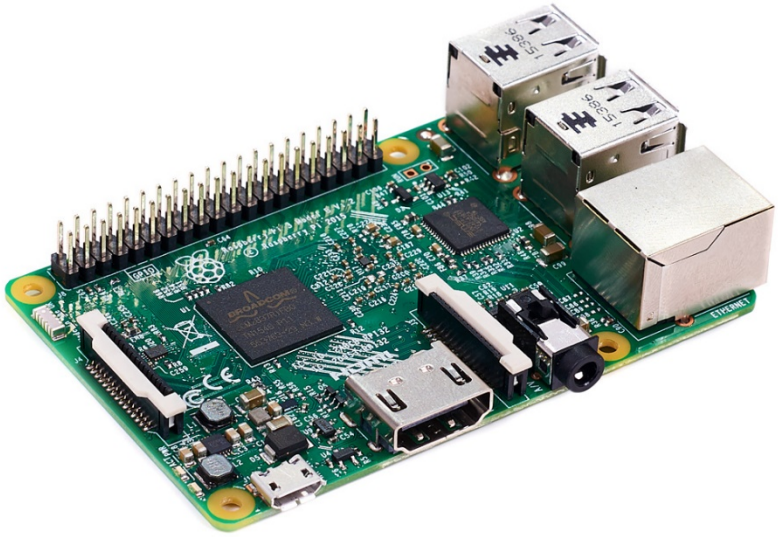

Рис. 1. Raspberry Pi 3B

К Raspberry подключается USB-веб-камера, с разрешением изображения 620 на 480 пикселей. Или любая цифровая видеокамера.

Рис. 2. Стандартная внешняя веб-камера

2.2.Настройка необходимого ПО

Для решения задачи была установлена рекомендуемая операционная система Raspbian с официального сайта: https://www.raspberrypi.org/. Был установлен интерпретатор Python 3.6.8 с официального сайта: https://www.python.org/

Большое удобство для разработки представляет менеджер пакетов pip, поставляемый с интерпретатором, с его помощью были установлены библиотеки и фреймворки:

OpenCV – свободная библиотека компьютерного зрения,

Flask – фреймворк для запуска веб-сервера с помощью python,

Imutils – удобные функции для работы с видеопотоком,

Numpy – библиотека для работы с матрицами и массивами,

Код разрабатываемого проекта был помещен под контроль версий git с синхронизацией на удалённом репозитории в системе redmine в ветке dev-marakulin. Смотрите приложение.

2.3.Разработка системы для записи видеопотока.

Для записи видео были созданы объекты соответствующих классов из OpenCV, позволяющие быстро получить доступ к видеопотоку камеры.

```
cap = cv2.VideoCapture(0)

t = time.time()
timestamp = time.strftime("%H.%M.%S", time.gmtime(t))
if not (os.path.exists('videos')):
    os.mkdir('videos')
name = f'videos/output_{timestamp}.avi'

out = cv2.VideoWriter(name, cv2.VideoWriter_fourcc(*'MPEG'), 30.0, (640,480))
```

Каждые 1/30 секунды кадр **frame** читается и записывается в поток вывода **out** с использование кодека MPEG указанного выше. Использование кодека позволяет сжать видеофайл до приемлемых размеров по сравнению с покадровой записью. Смотрите файл videorecording.py в приложении.

```
ret, frame = cap.read()
out.write(frame)
```

2.4.Создание и запуск веб-сервера

Веб-сервер был создан с помощью фреймворка Flask. В коде python с помощью декораторов объявляются несколько веб-страниц:

```
app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/text_stream')
def text_stream():
    ......

@app.route("/video_feed")
def video_feed():
    ......
```

Основная страница **index.html** была создана заранее и объединяет страницы **textstream** – страница с данными о координатах видеокамеры и **videofeed** – страница с обновляемым видеопотоком. Смотрите файл webserver.py в приложении.

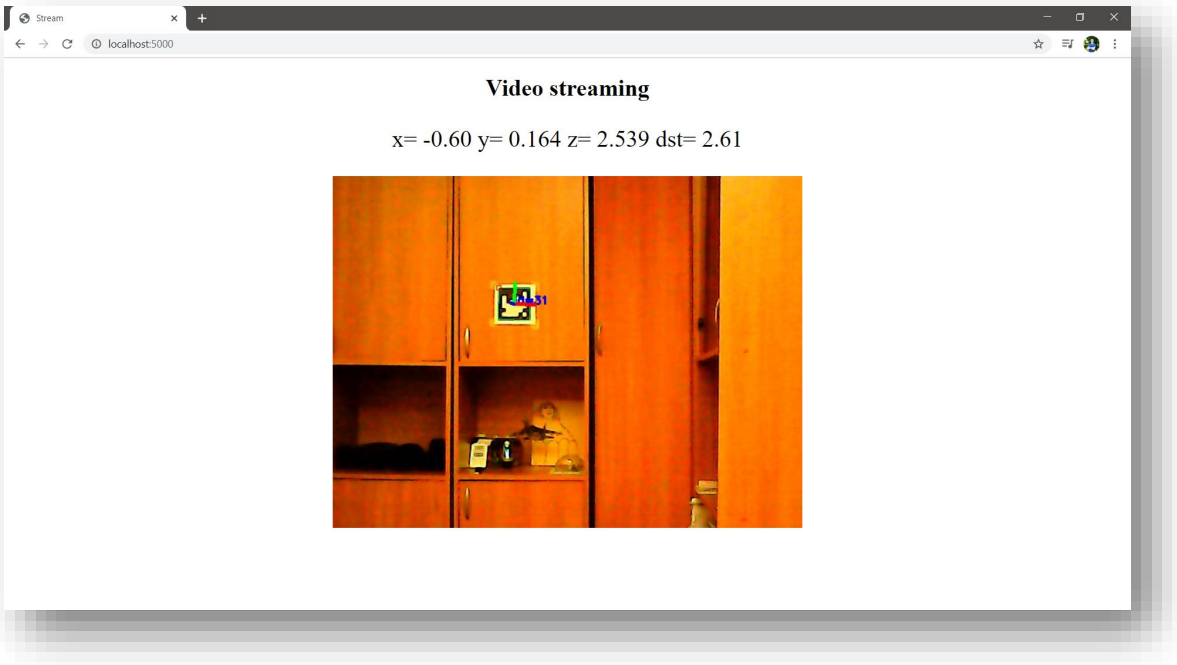

Рис. 3. Результат работы запущенного веб-сервера.

2.5.Калибровка камеры, получение коэффициентов искажения.

Любая камера имеет внутренние и внешние параметры, влияющие на формирование изображения. Внешние параметры, связанные с ориентацией (поворотом и смещением) камеры относительно некоторой мировой системы координат и внутренние параметры, например, фокусное расстояние, оптический центр и коэффициенты радиального искажения объектива.

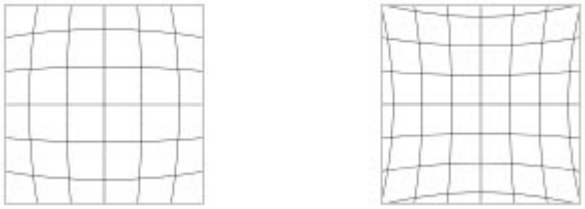

Рис. 4. а) бочкообразное искажение; б) подушкообразное искажение

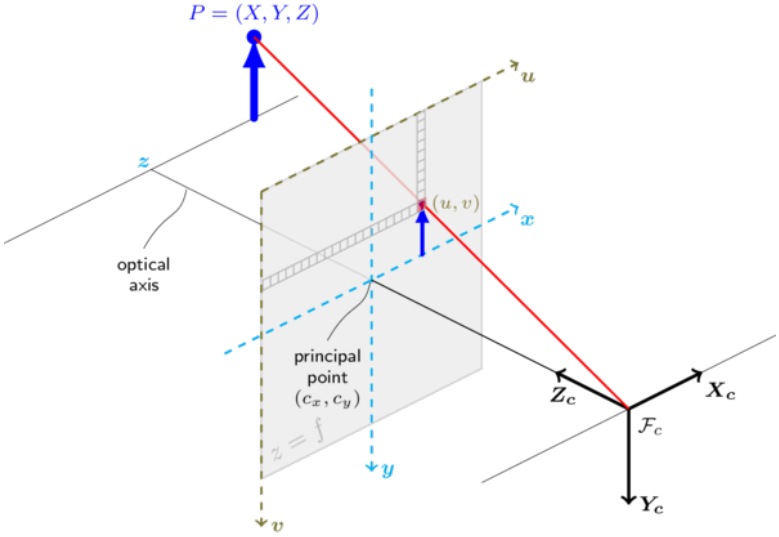

Рис. 5. Проецирование точки на плоскость изображения

Уравнение, которое связывает трехмерную точку (X_w, Y_w, Z_w) в мировых координатах с её проекцией в координатах изображения (u, v), имеет следующий вид:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix},$$

где **P** — матрица проекции размером 3 × 4, состоящая из двух частей: внутренней матрицы **K**, которая содержит внутренние параметры и внешнюю матрицу

([**R**|**T**]), которая является комбинацией матрицы вращения **R** размером 3 × 3 и вектора смещения **T** размером 3×1:

$$\mathbf{P} = \mathbf{K}\left[\mathbf{R}|\mathbf{T}\right]$$

Матрица **K** называется внутренней матрицей камеры:

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

Где f_x, f_y – фокусные расстояния по **x** и по **y**, c_x, c_y – координаты оптического центра изображения.

Цель калибровки камеры – нахождение матрицы **K**, матрицы вращения **R** и вектора смещения **T**. Эти параметры можно найти, используя стандартные средства OpenCV методом шахматной доски. Суть метода в том, что при достаточном количестве изображений шахматной доски, сфотографированных с разных углов, можно определить, насколько искажены её границы при разных точках наблюдения.

Рис. 6. Шахматная доска с найденными внутренними сторонами

Был написан вспомогательный скрипт, определяющий коэффициенты данных матриц на основе двух функций:

```
cv2.findChessboardCorners()
cv2.calibrateCamera()
```

Смотрите файл calibration.py в приложении.

2.6.Создание системы распознавания маркеров ArUco.

Маркеры ArUco – специальные контрастные квадратные изображения, удобные для распознавания системами компьютерного зрения. Изображения кодируют в себе свой уникальный идентификатор id. Поэтому под распознаванием будем понимать нахождение координат маркера на произвольной фотографии и определение его id. Если разместить маркер известного размера на посадочной площадке квадрокоптера, то с помощью установленной на нём системы можно найти его координаты относительно площадки и использовать их для автоматической посадки.

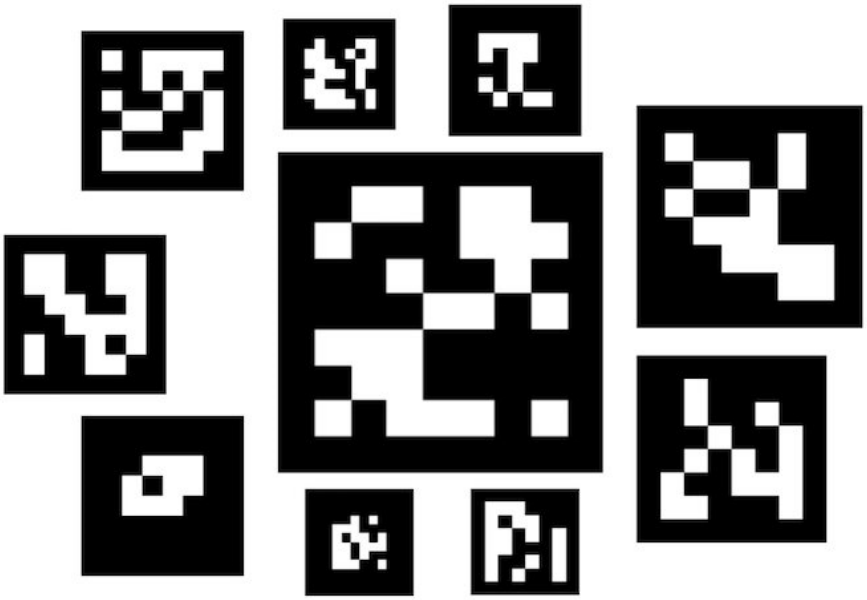

Рис. 7. Примеры маркеров ArUco

В стандартных средствах OpenCV уже существует функция detectMarkers, результатом работы которой являются координаты углов распознанных маркеров и их id. Входными данными являются изображение и словарь маркеров для поиска. Согласно описанию в документации, функция использует следующую цепочку методов:

1. применение адаптивного порогового фильтра,
2. нахождение контуров и отбор кандидатов,
3. трансформация перспективы,
4. извлечение битовой кодировки и сравнение с набором маркеров.

Пороговый фильтр – метод, преобразующий яркость изображения к бинарной, разделяя яркость выше порога и ниже порога по заранее определённому значению. *Адаптивный пороговый фильтр* ориентируется на яркость блока соседних пикселей и нечувствителен к глобальным перепадам яркости.

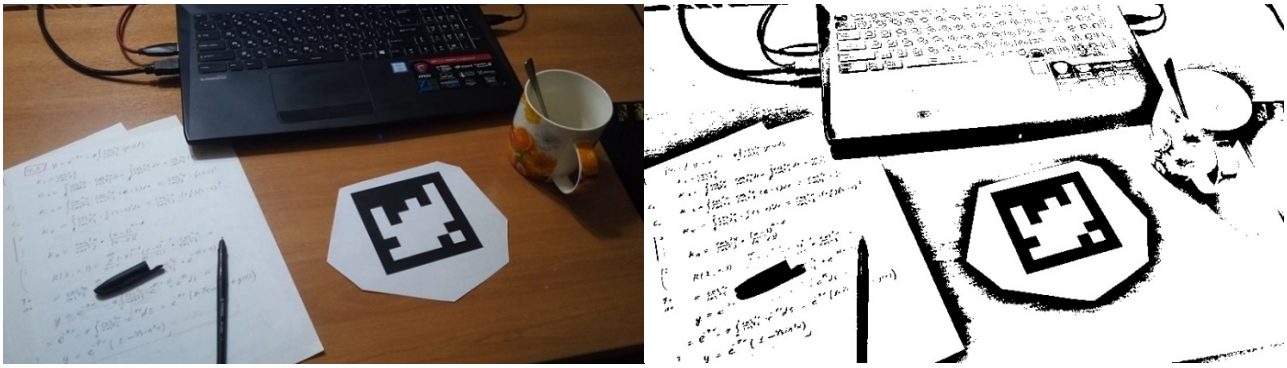

Рис. 8. а) Оригинальное изображение б) С адаптивным фильтром

Для нахождения контуров используется функция *FindContours* из OpenCV, далее контуры аппроксимируются, и отбираются наиболее подходящие варианты.

Рис. 9. Найденные контуры

Чтобы сравнить изображения в найденных контурах со словарём маркеров, недостаточно найти контур, вероятно он будет повёрнут относительно камеры или камера будет под углом к маркеру. Поэтому для сравнения изображение необходимо сначала «исправить». Для этого производится *трансформация перспективы* – операция, приводящее изображение, обведённое четырёхугольным контуром, к прямому квадратному виду.

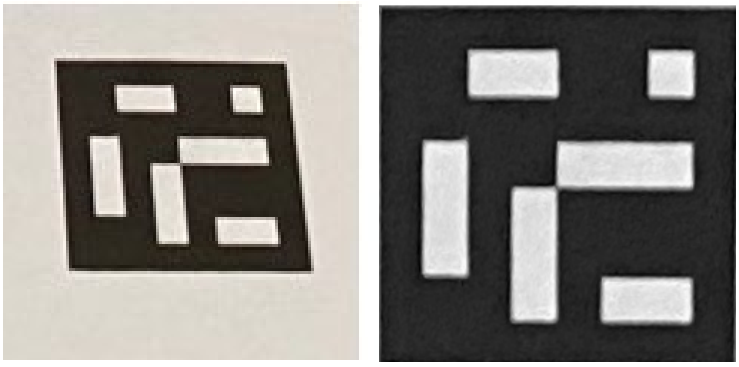

Рис. 10) Трансформация перспективы

Последним шагом в распознавании является извлечение битовой кодировки из квадратного изображения. После трансформации перспективы может случиться так, что ячейки маркера располагаются не идеально ровно, а немного захватывая территорию соседних ячеек, поэтому для извлечения используются центральные зоны ячеек.

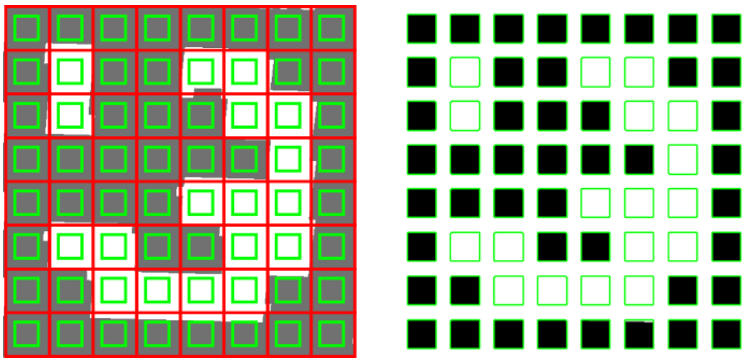

Рис. 11) Извлечение кодировки

2.7. Нахождение координат камеры относительно маркера

Теперь, когда в нашем распоряжении координаты углов маркеров и их id, можно определить координаты камеры относительно маркеров. Если на изображении найдено несколько маркеров, то для выбора центра мировой системы координат разумно выбрать маркер с наименьшим id, расположив оси OX, OY в плоскости маркера с пересечением в центре, а ось OZ ортогонально вверх.

В OpenCV существует функция estimatePoseSingleMarkers, однако, во-первых, она определяет координаты маркера на кадре, а во-вторых, даёт результат в формате вектора вращения **R** и смещения **T**. Для использования этой

функции как раз и пригождаются матрицы калибровочных коэффициентов, найденные ранее.

```
rvec, tvec, _ = estimatePoseSingleMarkers(Corners, m_len, cameraMatrix, distCoeffs)
```

Направление вектора вращения rvec совпадает с направлением оси вращения, а его длина равна углу поворота.

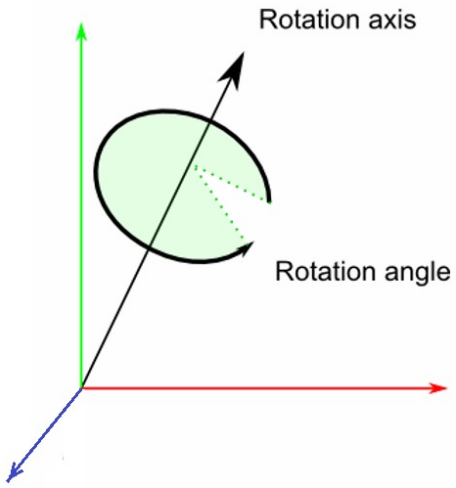

Рис. 12. Вектор вращения

Координаты центра маркера на кадре и мировые координаты связаны следующим соотношением:

$$s\ \mathbf{m}' = \mathbf{A}\ \left[\mathbf{R}|\mathbf{T}\right]\ \mathbf{M}'$$

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

- (X,Y,Z) – координаты в мировой системе,
- (u,v) координаты (в пикселях) на кадре,
- A – матрица коэффициентов искажения,
- s – масштабный коэффициент,
- R – матрица поворотов,
- T – вектор смещения.

Так как связь искомых координат имеет в себе матрицу поворотов, а в нашем распоряжении есть только вектор поворота, то его следует преобразовать к матрице поворота согласно следующим соотношениям:

$$\theta = norm(r)$$

$$r = r/\theta$$

$$R = \cos\theta I + (1 - \cos\theta) r r^T + \sin\theta \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$$

Таким образом, перейти от **R** и **T** к координатам камеры в декартовой системе координат позволяет следующее выражение:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R^T(-T)$$

Распознавание и нахождение координат камеры реализовано в файле cam_pos_detector.py (смотрите в приложении).

2.8. Объединение сервера, записи видео и распознавания.

Веб-сервер с трансляцией в реальном времени, запись видеопотока и нахождение координат камеры было объедено в единую систему в файле webserver.py. Пример работы веб-сервера.

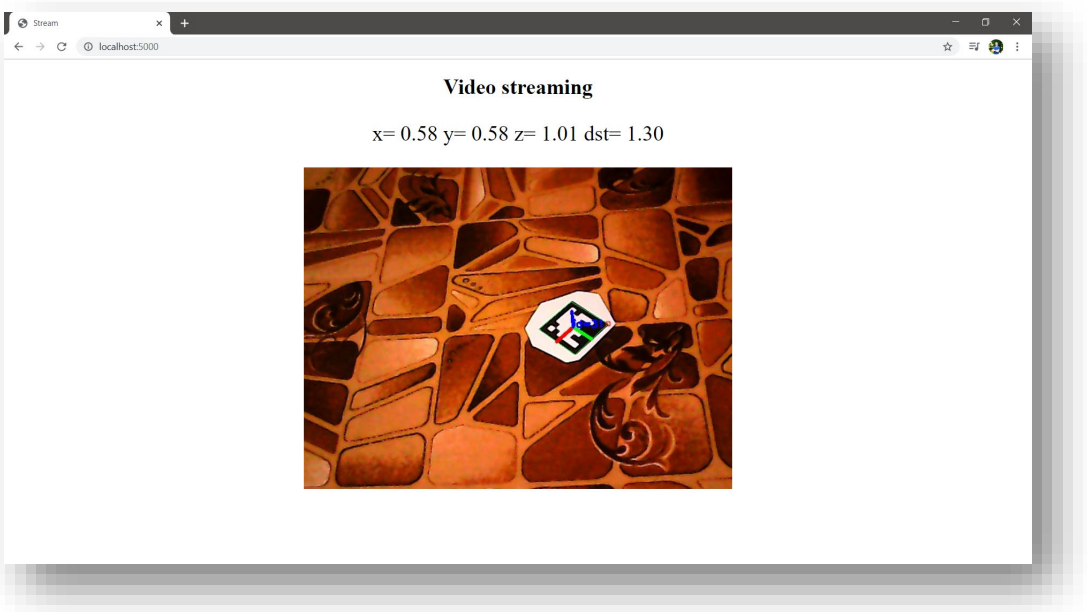

Рис. 13. Веб-страница запущенного сервера с координатами камеры

В центре транслируется изображение с камеры в реальном времени, на нём рисуются оси и грани распознанного маркера. Выше, над трансляцией, отображаются найденные координаты камеры относительно маркера и расстояние до центра маркера.

Рис. 14. Часть веб-страницы запущенного сервера

Незначительные колебания камеры вызывают колебания найденного расстояния приблизительно в 2-3% от абсолютного значения. Реальное расстояние отличается от найденного в пределах 3%. Таким образом, точность системы позволяет использовать её на практике.

3. ЗАКЛЮЧЕНИЕ

В рамках курсовой работы была разработана система определения местоположения камеры с помощью компьютерного зрения с возможностью записи видеопотока и трансляции его на другие компьютеры с помощью локальной сети Wi-Fi.

С технической стороны было разработано компактное и лёгкое устройство, основанное на Raspberry Pi, которое можно прикрепить ко множеству летательных аппаратов, роботов, беспилотных автомобилей.

Маркеры ArUco могут размещаться на посадочной площадке, парковочном месте, опорной точке, месте назначения и указывать беспилотным средствам их относительное положение для ориентации в пространстве.

Система распознавания позволяет удобно передавать полученную информацию на локальный веб-сервер, или, при подключении беспилотного аппарата к сети интернет на глобальный веб-сервер, тем самым упрощая пользователю получение информации о беспилотном средстве.

Системы автономной навигации всё больше востребованы в быстро развивающемся сегменте автономных аппаратов. В сравнении с GPS, ГЛОНАСС разработанная система в десятки раз точнее, в сравнении с радарами и лидарами, значительно дешевле. Систему можно использовать для множества задач: для автоматической парковки автомобиля, для указания места посадки квадрокоптера, для навигации БПЛА над городом, для указания тормозного пути поездов и так далее. Потенциал применения системы, а также её дальнейшей модификации очень велик.

4. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. OpenCV: Camera Calibration and 3D Reconstruction. – URL: https://docs.opencv.org/3.3.1/d9/d0c/group__calib3d.html (дата обращения: 02.04.2020).
2. Detection of ArUco Markers. – URL: https://docs.opencv.org/3.2.0/d5/dae/tutorial_aruco_detection.html (дата обращения: 02.04.2020).
3. Flask documentation. – URL: https://flask.palletsprojects.com/en/1.1.x/ (дата обращения: 08.03.2020).
4. Геометрия формирования изображений.– URL: https://waksoft.susu.ru/2020/02/23/geometriya-formirovaniya-izobrazhenij/ (дата обращения: 06.04.2020).
5. Проблема сферических искажений в камерах видеонаблюдения. Исправление искажений. – URL: http://www.absolutss.ru/Particles/art_13/ (дата обращения: 10.04.2020).
6. Adaptive Thresholding. – URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm
7. Suzuki, S., and Be, K. (1985). Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing 30, p. 32–46
8. Augmented Reality using ArUco Markers in OpenCV. – URL: https://www.learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/ (дата обращения: 28.04.2020).
9. Моделирование и распознавание 2D/3D образов. – URL: https://api-2d3d-cad.com/euler_angles_quaternions/ (дата обращения: 04.05.2020).
10. Calculate X, Y, Z Real World Coordinates from Image Coordinates using OpenCV. – URL: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/ (дата обращения: 06.05.2020).
11. Кватернионы. И. Л. Кантор, А. С. Солодовников Гиперкомплексные числа — М.: Наука, 1973. — 144 с.

5. ПРИЛОЖЕНИЕ

Поскольку количество строк кода велико, то в тексте представлены лишь основные функции и концепции.

Описание главных файлов репозитория:

1. calibration.py – скрипт для получения калибровочных коэффициентов камеры, коэффициенты записываются в calibration_data.json. И используются в других файлах.
2. aruco_generator.py – скрипт для получения изображения конкретного маркера и его последующей печати.
3. aruco_simple_detector.py – скрипт распознающий маркеры и рисующий на изображении их контуры
4. cam_pos_detector.py – скрипт распознающий маркеры и определяющий позицию камеры относительно маркеров
5. videorecording.py – скрипт записывающий видеопоток с камеры в видеофайлы в папку videos
6. webserver.py – скрипт, запускающий веб-сервер на локальном хосте на порте 5000, с объединённой системой распознавания маркеров, записи видеопотока и определения координат камеры.