ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовой проект

по дисциплине «Объектно-ориентированное программирование»

«Разработка системы автоматического отслеживания теннисного мяча на базе робота OmegaBot с использованием Raspberry Pi и Arduino»

Выполнил
студент гр.
3331506/20401 _____ *(подпись)* Скорнякова Е.А.
Юлик В.М.

Работу принял _____ *(подпись)* Ананьевский М.С.

Санкт-Петербург
2025

Оглавление

Техническое задание

Тема:

Разработка системы автоматического трекинга и сопровождения теннисного мяча на базе роботизированной платформы «OmegaBot» с использованием компьютерного зрения и микроконтроллеров.

Цель проекта:

Создать программно-аппаратный комплекс, способный в реальном времени обнаруживать теннисный мяч, определять его координаты и обеспечивать движение робота для его сопровождения.

Задачи:

- Реализовать алгоритм распознавания окружностей (теннисного мяча) с помощью библиотеки OpenCV.
- Обеспечить передачу данных между Raspberry Pi (обработка изображения) и Arduino (управление моторами).
- Разработать систему управления движением робота на основе полученных координат мяча.

Ожидаемый результат:

Робот должен автономно определять положение теннисного мяча в поле зрения камеры и двигаться так, чтобы удерживать его в центре кадра.

1. Введение

Современная робототехника активно развивается в направлении автономных систем, способных взаимодействовать с динамическими объектами в реальном времени. Одной из ключевых задач в этой области является трекинг подвижных целей, который находит применение в спортивной аналитике, роботизированных играх и системах автоматизации.

В данной работе рассматривается разработка системы автоматического сопровождения теннисного мяча на базе робота OmegaBot. В качестве технической основы используются два микроконтроллера: Raspberry Pi (для обработки видеопотока и детектирования мяча) и Arduino (для управления двигателями).

Актуальность проекта обусловлена необходимостью создания эффективных алгоритмов компьютерного зрения для работы в реальном времени, а также интеграции программных и аппаратных компонентов в единую систему.

2. Теоретическая часть

2.1 HSV

HSV — это цветовая модель, разделяющая цвет (Hue), насыщенность (Saturation) и яркость (Value). Она особенно полезна в компьютерном зрении для выделения объектов по цвету, поскольку позволяет более эффективно фильтровать цвета, чем RGB. В OpenCV преобразование из BGR в HSV осуществляется с помощью cv2.cvtColor().

Мы используем: cvtColor(*img, hsv, COLOR_BGR2HSV);

Описание функции на официальном сайте библиотеки:

```
void cv::cvtColor(
    InputArray src,      // Входное изображение (BGR)
    OutputArray dst,     // Выходное изображение (HSV)
    int code,            // Код преобразования: COLOR_BGR2HSV
    int dstCn = 0        // Число каналов выходного изображения (0 = авто)
);
```

2.2 Объект Mat

Mat — это основной контейнер для хранения матриц изображений в OpenCV. Он представляет собой n-мерный плотный массив, содержащий данные изображения и метаинформацию, такую как размер, количество каналов и тип данных. Это позволяет эффективно управлять изображениями и выполнять различные операции обработки.

2.3 Маски

Маска в OpenCV — это бинарное изображение, где белые пиксели (значение 255) указывают на области интереса, а черные (значение 0) — на игнорируемые области. Маски используются для выделения или подавления определенных частей изображения, например, при фильтрации цвета или применении морфологических операций.

2.4 Цветовая фильтрация

Цветовая фильтрация позволяет выделить объекты определенного цвета на изображении. Обычно изображение преобразуется в HSV-пространство, после чего с помощью функции cv2.inRange() создается маска для заданного

диапазона оттенков. Затем, используя cv2.bitwise_and(), можно извлечь интересующие области.

Мы используем: inRange(hsv, lower, upper, mask);

Описание функции на официальном сайте библиотеки:

```
void cv::inRange(
    InputArray src,      // Входное изображение (HSV)
    InputArray lowerb,   // Нижняя граница диапазона (Scalar)
    InputArray upperb,   // Верхняя граница диапазона (Scalar)
    OutputArray dst      // Выходная бинарная маска
);
```

2.5 Морфологические преобразования

Морфологические операции, такие как эрозия и дилатация, применяются к бинарным изображениям для удаления шума, заполнения пробелов и улучшения структуры объектов. Они основаны на применении структурирующего элемента (ядра) к изображению. OpenCV предоставляет функции cv2.erode(), cv2.dilate(), cv2.morphologyEx() для выполнения этих операций.

Мы использовали morphologyEx(mask, mask, MORPH_OPEN, kernel);

Описание функции на официальном сайте библиотеки:

```
void morphologyEx(
    InputArray src,
    OutputArray dst,
    int op,
    InputArray kernel,
    Point anchor = Point(-1,-1),
    int iterations = 1,
    int borderType = BORDER_CONSTANT,
    const Scalar& borderValue = morphologyDefaultBorderValue()
);
```

Используемые нами аргументы функции представлены в таблице 1.

Аргумент	Тип	Описание
src	InputArray	Входное изображение (в вашем случае — бинарная маска mask).
dst	OutputArray	Выходное изображение (здесь результат записывается обратно в mask).
op	int	Тип морфологической операции (MORPH_OPEN — открытие).

kernel	InputArray	Структурирующий элемент (в вашем случае — квадратная матрица 5×5).

Рассмотрим более подробно используемый нами тип морфологической операции MORPH_OPEN. Операция состоит из двух последовательных действий:

1. Эрозия (Erosion):

Удаляет мелкие белые объекты (шум) и уменьшает границы основного объекта.

2. Дилатация (Dilation):

Восстанавливает размер основного объекта после эрозии.

Итог: MORPH_OPEN = Дилатация (Эрозия(src)).

Удаляет шумы и разрывы, сохраняя форму крупных объектов.

2.6 Поиск контура

Контуры представляют собой границы объектов на изображении. В OpenCV функция cv2.findContours() используется для обнаружения контуров на бинарных изображениях.

Мы использовали cv::findContours(*mask, contours, cv::RETR_TREE, cv::CHAIN_APPROX_SIMPLE);

Описание функции на официальном сайте библиотеки:

```
void cv::findContours(
    InputArray image,                  // Входное бинарное изображение
    OutputArrayOfArrays contours,      // Выходной список контуров
    OutputArray hierarchy,             // Иерархия контуров
    int mode,                          // Режим поиска (RETR_TREE и др.)
    int method,                        // Метод аппроксимации (CHAIN_APPROX_SIMPLE)
    Point offset = Point()             // Смещение контуров
);
```

Используемые нами аргументы функции представлены в таблице 2.

Аргумент	Тип	Описание
mask	InputArray	Входное изображение (в вашем случае — бинарная маска mask).

contours	std::vector<std::vector< cv::Point>>	выходной параметр, вектор векторов точек
cv::RETR_TREE	int	Режим поиска. Возвращает иерархию контуров (например, контуры внутри других контуров).
cv::CHAIN_APPROX_SIMPLE	int	Метод аппроксимации: сжимает контур, оставляя только ключевые точки (например, для прямоугольника вернёт 4 угла вместо всех пикселей границы).

2.7Аппроксимация контура минимальной охватывающей окружностью

Аппроксимация (приближение) объекта минимальной охватывающей окружностью — это нахождение наименьшей по радиусу окружности, которая полностью охватывает заданный контур (набор точек). Что необходимо для определения центра теннисного мячика.

Мы использовали cv::minEnclosingCircle(contour, center, radius);

Описание функции на официальном сайте библиотеки:

```
void cv::minEnclosingCircle(
    InputArray points,       // Входной контур (вектор точек)
    Point2f& center,         // Выходной параметр: центр окружности
    float& radius            // Выходной параметр: радиус окружности
);
```

Используемые нами аргументы функции представлены в таблице 3.

Аргумент	Тип	Описание
contour	InputArray	Входной вектор точек контура (в нашем случае контур)
center	std::vector<std::vector< cv::Point>>	Выходной параметр, содержащий координаты центра окружности

radius	float	Режим поиска. Возвращает иерархию контуров (например, контуры внутри других контуров).

2.8 Взаимодействие с камерой с помощью библиотеки libcamera

Работа с камерой через libcamera в C++ состоит из следующих этапов:

1. Инициализация камеры и менеджера устройств

Пример кода:

```
CameraManager cm;
cm.start();
```

Пояснение:

CameraManager управляет списком подключённых камер. Метод start() инициализирует систему libcamera и позволяет получить доступ к камерам.

2. Выбор и захват камеры\

Пример кода:

```
gCamera = cm.cameras()[0];
gCamera->acquire();
```

Пояснение:

Из массива камер выбирается первая доступная (индекс [0]).

Вызов acquire() блокирует использование камеры другими процессами.

3. Конфигурация потока видеоданных

Пример кода:

```
auto config = gCamera->generateConfiguration({StreamRole::Viewfinder});
config->at(0).pixelFormat = formats::RGB888;
config->at(0).size.width = 1920;
config->at(0).size.height = 1080;
config->at(0).bufferCount = 4;
gCamera->configure(config.get());
gConfig = &config->at(0);
```

Пояснение:

- Создается конфигурация потока с ролью Viewfinder — предназначено для отображения.
- Устанавливаются формат пикселей (RGB888), разрешение, количество буферов.
- Камера настраивается через configure().

4. Выделение буферов для кадров

Пример кода:

```
FrameBufferAllocator allocator(gCamera);
for (StreamConfiguration &cfg : *config)
    allocator.allocate(cfg.stream());
```

Пояснение:

FrameBufferAllocator выделяет память для хранения кадров (кадровые буферы), которые будут использоваться для передачи данных изображения.

5. Создание и добавление буферов в запросы

Пример кода:

```
std::vector<std::unique_ptr<Request>> requests;
for (StreamConfiguration &cfg : *config) {
    for (const auto &buffer : allocator.buffers(cfg.stream())) {
        auto request = gCamera->createRequest();
        request->addBuffer(cfg.stream(), buffer.get());
        requests.push_back(std::move(request));
    }
}
```

Пояснение:

Request — структура, описывающая одну операцию захвата изображения.

Каждому буферу сопоставляется Request, который добавляется в очередь захвата.

6. Запуск камеры и отправка запросов

Пример кода:

```
gCamera->start();
for (auto &req : requests)
    gCamera->queueRequest(req.get());
```

Пояснение:

Камера запускается.

Все подготовленные запросы помещаются в очередь на выполнение.

7. Обработка полученного кадра — requestComplete

Пример кода:

```
gCamera->requestCompleted.connect(requestComplete);
```

Пояснение:

Устанавливается функция-обработчик requestComplete, которая вызывается при завершении обработки очередного кадра.

Внутри этой функции:

- Создаётся объект cv::Mat с использованием указателя на данные буфера:
- cv::Mat frame(height, width, CV_8UC3, data);
- Выполняется resize кадра.
- Применяется цветовая маска и извлекается контур.
- Вычисляется ошибка координат центра объекта относительно центра кадра.
- Значения передаются через send_to_arduino().

8. Завершение работы

Пример кода:

```
gCamera->stop();
gCamera->release();
cm.stop();
```

Пояснение:

Камера останавливается, освобождается, завершается работа CameraManager.

2.9 Особенности привода платформы

Дифференциальный привод — это один из самых распространённых и простых в реализации типов приводов для мобильных роботов. Он обеспечивает высокую манёвренность и точность управления, что делает его популярным в образовательных, исследовательских и промышленных проектах.

Принцип работы дифференциального привода:

Дифференциальный привод включает два ведущих колеса, каждое из которых управляется отдельным мотором. Управление движением осуществляется за счёт изменения скоростей вращения левого и правого колёс.

- Прямолинейное движение: оба колеса вращаются с одинаковой скоростью и в одном направлении.

- Поворот: колёса вращаются с разной скоростью; направление поворота зависит от того, какое колесо вращается быстрее.
- Разворот на месте: колёса вращаются с одинаковой скоростью, но в противоположных направлениях.

2.10 ПИД-регулятор

ПИД-регулятор (пропорционально-интегрально-дифференциальный регулятор) — это механизм обратной связи, используемый для поддержания заданного уровня параметра системы. Он состоит из трех компонентов:

- Пропорциональный (P): реагирует на текущую ошибку.
- Интегральный (I): учитывает накопленную ошибку во времени.
- Дифференциальный (D): прогнозирует будущую ошибку на основе скорости изменения.

ПИД-регуляторы широко применяются в промышленности для управления температурой, скоростью, давлением и другими параметрами.

3. Описание аппаратной части

В данном проекте для создания системы автоматического трекинга используется роботизированная платформа «OmegaBot». В нее входят:

- Микрокомпьютер Raspberry Pi 3+;
- Модуль видеокамеры Raspberry Pi Camera Module V2;
- Коллекторные двигатели;
- Микроконтроллер Arduino Uno.

3.1 Raspberry Pi 3+

Это мини-компьютер размером с банковскую карту, но с возможностями полноценного ПК. Он используется в этом проекте как "мозг", который обрабатывает изображение с камеры, находит на нем теннисный мяч и определяет его координаты. Все вычисления происходят прямо на борту Raspberry Pi, без подключения к ноутбуку или облаку.

На Raspberry Pi установлена операционная система (подробнее в разделе «Установка»), а также библиотеки для компьютерного зрения — в частности, OpenCV. Именно через неё происходит распознавание объекта.

3.2 Raspberry Pi Camera Module V2

Эта камера подключается к специальному CSI-порту на плате и передаёт видеопоток напрямую, с минимальной задержкой. Она может снимать в высоком разрешении, но для быстроты работы в нашем проекте кадры уменьшаются до 640×480.

3.3 Коллекторные двигатели

Для движения робота используются коллекторные электродвигатели постоянного тока (DC motors). Данные двигатели не умеют вращаться с маленькой скорость, как шаговые, из-за чего точность позиционирования уменьшается, но зато у них простое управление (подавать ШИМ сигнал на драйвер через цифровой пин на микроконтроллере).

3.4 Arduino Uno

Arduino отвечает за движение робота. Микроконтроллер подключен к моторам робота через драйвер и подаёт на них управляющие сигналы ШИМ — поворачивать, ехать вперёд или назад.

Когда Raspberry Pi находит координаты мяча, он отправляет их через USB порт на Arduino.

4. Установка

4.1 Установка Raspberry Pi OS Bookworm

Для начала скачиваем официальный загрузчик Raspberry Pi Imager (рисунок 1) с сайта raspberrypi.com и при помощи него записал операционную систему на SD-карту. В качестве ОС я выбрал Raspberry Pi OS Bookworm (32-bit) — это последняя стабильная версия.

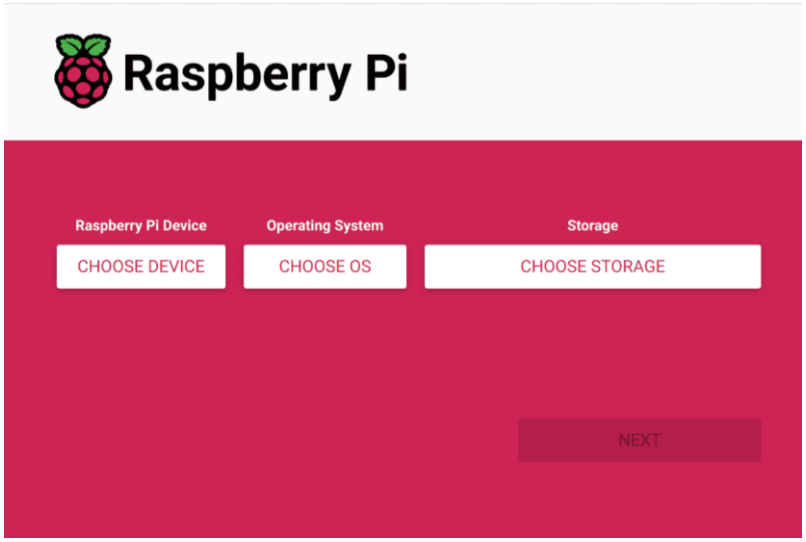


Рисунок 1 – Официальный загрузчик образа для RPi

4.2 Первый запуск Raspberry Pi и обновление системы

Вставляем SD-карту в слот на нижней стороне Raspberry Pi. Подключаем USB-адаптер питания, клавиатуру, мышь и монитор через HDMI.

После подачи питания Raspberry Pi автоматически включается — загорается красный светодиод (питание) и зелёный (работа SD-карты).

На первом экране появляется мастер первоначальной настройки, после которого мы переходим на рабочий стол нашего микрокомпьютера (рисунок 2).

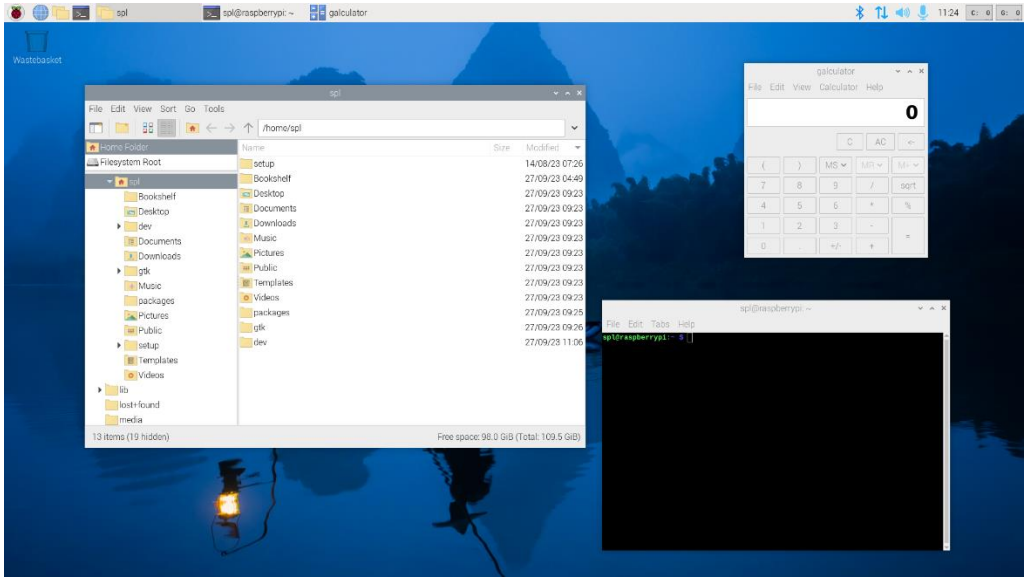


Рисунок 2 – Рабочий стол RPi OS Bookworm

После успешного запуска сразу проводим полное обновление системы, так как Raspberry Pi OS регулярно получает новые версии пакетов.

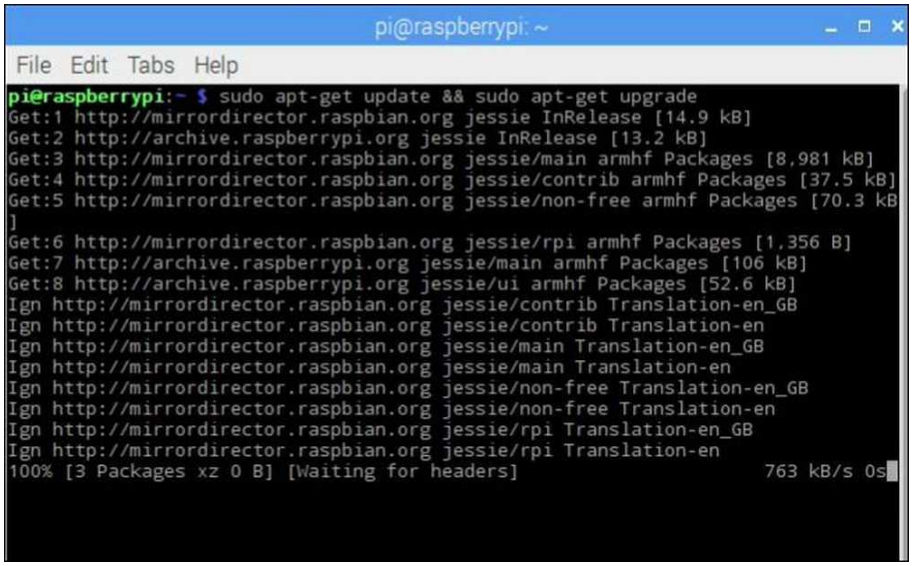


Рисунок 3 – Обновление системы

Raspberry Pi, после данного процесса, готов к установке всех нужных библиотек и работе с камерой и Arduino без ошибок, связанных с устаревшими компонентами.

4.3 Подключение камеры Raspberry Pi Camera Module V2

Камера Raspberry Pi Camera V2 подключается к плате Raspberry Pi с помощью плоского шлейфа (FFC) и CSI-разъёма (Camera Serial Interface).

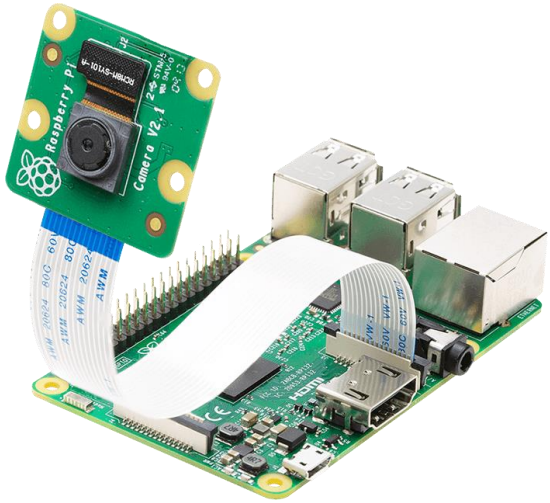


Рисунок 4 – Подключение камеры

Так как мы используем Raspberry Pi OS Bookworm и хотим работать через libcamera и OpenCV — необходимо отключить программный стек raspicam. Для этого переходим в настройки Raspberry Pi и выбираем пункт Interface Options → Legacy Camera → No.

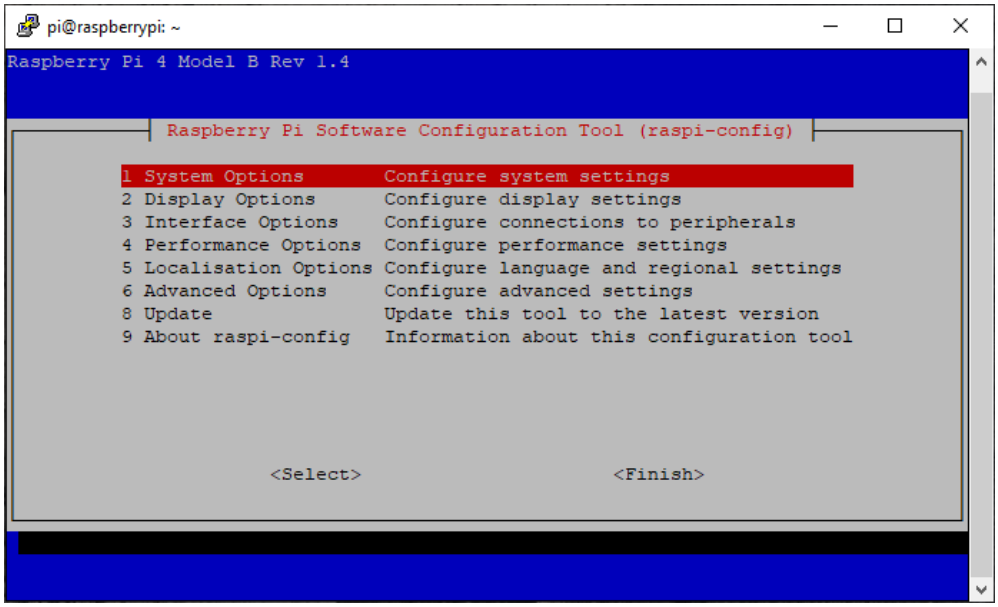


Рисунок 5 – Настройка камеры

После перезагрузки системы убеждаемся, что камера обнаружена. Для этого в терминале выполняем команду:

```
libcamera-hello --list-cameras
```

ниже выводится сообщение с информацией о камере, сенсоре и разрешении.

4.4 Установка OpenCV

Библиотеку будем устанавливать вручную из исходников. Открываем терминал в домашней папке пользователя. Далее клонируем репозиторий с основной библиотекой:

```
git clone https://github.com/opencv/opencv.git
```

в результате создается папка opencv/ , в которой находятся все исходные файлы библиотеки.

Клонируем дополнительный модуль:

```
git clone https://github.com/opencv/opencv_contrib.git
```

После загрузки исходников переходим к этапу сборки библиотеки OpenCV из исходников. Сначала переходим в папку opencv/ , где лежит основной код, и создаём там подкаталог build/, чтобы в ней провести сборку:

```
cd ~/opencv
mkdir build
cd build
```

Теперь настраиваем сборку OpenCV: включаем оптимизацию компиляции под производительность; указываем, куда установить OpenCV после сборки; подключаем дополнительные модули из opencv_contrib; включаем аппаратные ускорения для ARM-процессоров; подключаем поддержку многопоточности через библиотеку TBB; включаем поддержку видеопотока через Video4Linux и т. д.:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
      -D ENABLE_NEON=ON \
      -D ENABLE_VFPV3=ON \
      -D WITH_TBB=ON \
      -D WITH_V4L=ON \
      -D WITH_QT=OFF \
      -D BUILD_EXAMPLES=OFF ..
```

Далее запускаем сборку (компиляция занимает около 2-3 часов):

```
make -j4
```

Когда сборка завершена без ошибок, запускаем установку и обновляем системный кэш библиотек:

```
sudo make install
sudo ldconfig
```

5. Распознавание

Система распознавания теннисного мяча (Приложение 2) реализована на базе библиотеки OpenCV и включает следующие этапы:

5.1 Захват и предварительная обработка кадра

- Получение видеопотока с камеры через libcamera.
- Преобразование кадра в формат RGB и уменьшение разрешения до 640×480 для оптимизации вычислений.
- Цветовая фильтрация (сегментация мяча)
- Преобразование изображения из RGB в HSV-пространство для более устойчивого выделения цвета,
- Применение бинарной маски с заданными границами цветового диапазона (подбирались экспериментально с помощью вспомогательной программы с трекбарами (Приложение1)),
- Использование морфологических операций (открытие — MORPH_OPEN) для устранения шумов и сглаживания контуров;

5.2 Выделение контура мяча

- Поиск всех контуров на бинарной маске с помощью функции findContours(),
- Выбор наибольшего контура (по площади) для исключения ложных срабатываний;

5.3 Определение координат центра и радиуса

- Аппроксимация контура минимальной охватывающей окружностью (minEnclosingCircle(),
- Фильтрация объектов по радиусу (игнорирование слишком мелких деталей),
- Визуализация результата: отрисовка окружности и центра на исходном кадре (для отладки);

5.4 Расчёт ошибки позиционирования

- Сравнение координат центра мяча с центром кадра (целевой точкой),

- Передача отклонений (error_x, error_y) и радиуса мяча на Arduino для управления роботом;

6. Управление роботом

6.1. Аппаратная конфигурация

Управление движением робота OmegaBot реализовано на базе микроконтроллера Arduino, который:

- Получает данные об отклонении мяча от центра кадра (error_x, error_y, radius) от Raspberry Pi через UART-интерфейс.
- Управляет двухканальным H-мостом для регулировки скорости и направления моторов:
- Левый мотор: PWM-сигнал (пин 6), направление (пин 7).
- Правый мотор: PWM-сигнал (пин 5), направление (пин 4).

6.2. Алгоритм управления на основе ПИД-регулятора

Для плавного сопровождения мяча реализован ПИД-регулятор (пропорционально-интегрально-дифференциальный), работающий по отклонению по оси X:

Расчёт ошибки:

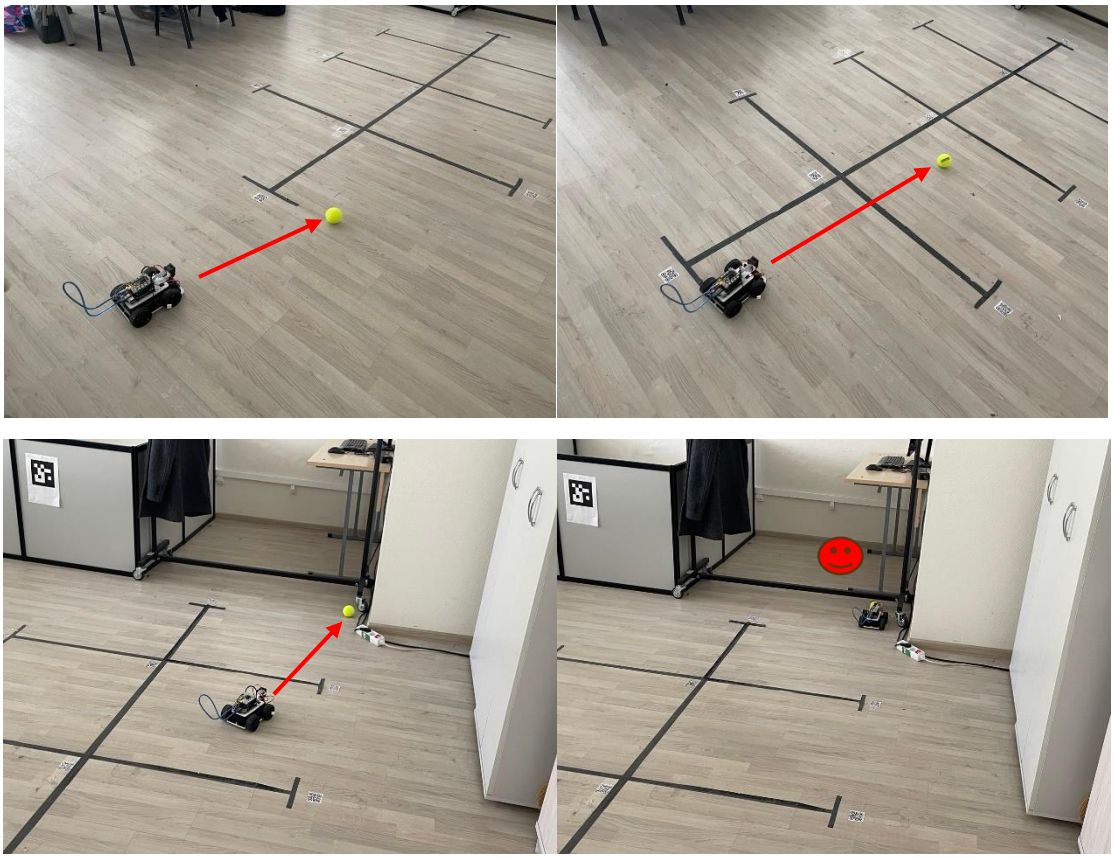
- error_x — разность между текущим положением мяча и центром кадра (в пикселях).

6.3 Особенности реализации

- Дифференциальное управление: Разность скоростей моторов обеспечивает поворот робота в сторону мяча,
- Коэффициенты ПИД: Подобраны экспериментально (Kp = 5.0, Kd = 4.0),
- Минимальная скорость: Нижний порог (70) исключает "мёртвую зону" при малых ошибках;

7. Результат

В результате была разработана система автоматического трекинга и сопровождения теннисного мяча на базе роботизированной платформы OmegaBot. Реализация алгоритмов компьютерного зрения на Raspberry Pi в связке с микроконтроллером Arduino Uno позволила создать работоспособную и адаптируемую систему, способную функционировать в реальном времени.



Созданный комплекс способен не только отслеживать теннисный мяч, но и быть легко перенастроен для сопровождения других объектов — благодаря гибкости алгоритма цветовой фильтрации и анализа контуров.

8. Заключение

В процессе выполнения проекта мы значительно расширили свои знания в области компьютерного зрения, обработки изображений и микроконтроллерного управления. Научились работать с библиотекой OpenCV на языке C++, разбираться в принципах цветовой фильтрации и контурного анализа, а также настраивать видеопоток с камеры Raspberry Pi.

В дальнейшем планируются следующие улучшения для более стабильной работы системы:

- Добавление функции предсказания траектории объекта с использованием фильтра Калмана;
- Переход на нейросетевые модели для более надёжного распознавания объектов;
- Подключение к микрокомпьютеру через VNC Server для удаленной отладки.

Приложения

Приложение 1. Программа для подбора HSV-коэффициентов

```
#include <libcamera/libcamera.h>
#include <libcamera/camera_manager.h>
#include <libcamera/framebuffer_allocator.h>
#include <libcamera/request.h>

#include <opencv2/opencv.hpp>

#include <memory>
#include <thread>
#include <atomic>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

using namespace libcamera;
using namespace std;
using namespace cv;

shared_ptr<Camera> gCamera;
StreamConfiguration *gConfig;
atomic<bool> running{true};

int hMin = 0, sMin = 0, vMin = 0;
int hMax = 179, sMax = 255, vMax = 255;

void *mmapBuffer(const FrameBuffer::Plane &plane) {
    return mmap(nullptr, plane.length, PROT_READ, MAP_SHARED,
plane.fd.get(), 0);
}

void munmapBuffer(void *map, const FrameBuffer::Plane &plane) {
    munmap(map, plane.length);
}

void createTrackbars() {
    namedWindow("HSV Controls", WINDOW_NORMAL);
```

```
    createTrackbar("Hue Min", "HSV Controls", &hMin, 179);
    createTrackbar("Hue Max", "HSV Controls", &hMax, 179);
    createTrackbar("Sat Min", "HSV Controls", &sMin, 255);
    createTrackbar("Sat Max", "HSV Controls", &sMax, 255);
    createTrackbar("Val Min", "HSV Controls", &vMin, 255);
    createTrackbar("Val Max", "HSV Controls", &vMax, 255);
}

void requestComplete(Request *request) {
    createTrackbars();
    const FrameBuffer *buffer = request->buffers().begin()->second;
    void *data = mmapBuffer(buffer->planes()[0]);

    int width = gConfig->size.width;
    int height = gConfig->size.height;

    Mat frame(height, width, CV_8UC3, data);
    Mat rgb = frame.clone();

    Mat hsv, mask;
    cvtColor(rgb, hsv, COLOR_BGR2HSV);

    Scalar lower(hMin, sMin, vMin);
    Scalar upper(hMax, sMax, vMax);
    inRange(hsv, lower, upper, mask);
    Mat result;
    bitwise_and(rgb, rgb, result, mask);



    imshow("Camera", rgb);
    imshow("Mask", mask);
    imshow("Result", result);

    if (waitKey(1) == 27)
        running = false;

    munmapBuffer(data, buffer->planes()[0]);
```

```
    request->reuse(Request::ReuseBuffers);
    gCamera->queueRequest(request);
}

int main() {
    CameraManager cm;
    cm.start();

    gCamera = cm.cameras()[0];
    gCamera->acquire();
    gCamera->requestCompleted.connect(requestComplete);

    auto config = gCamera->generateConfiguration({StreamRole::Viewfinder});
    config->at(0).pixelFormat = formats::RGB888;
    config->at(0).size.width = 640;
    config->at(0).size.height = 480;
    config->at(0).bufferCount = 4;
    gCamera->configure(config.get());
    gConfig = &config->at(0);

    FrameBufferAllocator allocator(gCamera);
    for (StreamConfiguration &cfg : *config)
        allocator.allocate(cfg.stream());

    vector<unique_ptr<Request>> requests;
    for (StreamConfiguration &cfg : *config) {
        for (const auto &buffer : allocator.buffers(cfg.stream())) {
            auto request = gCamera->createRequest();
            request->addBuffer(cfg.stream(), buffer.get());
            requests.push_back(move(request));
        }
    }

    gCamera->start();
    for (auto &req : requests)
        gCamera->queueRequest(req.get());

    while (running)
```

```
        this_thread::sleep_for(chrono::milliseconds(10));

    gCamera->stop();
    gCamera->release();
    cm.stop();

    destroyAllWindows();
}
```

Приложение 2. Исходный код системы распознавания

```
#include <libcamera/libcamera.h>
#include <libcamera/camera_manager.h>
#include <libcamera/framebuffer_allocator.h>
#include <libcamera/request.h>

#include <opencv2/opencv.hpp>

#include <memory>
#include <thread>
#include <atomic>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <termios.h>


std::shared_ptr<libcamera::Camera> gCamera;
libcamera::StreamConfiguration *gConfig;
std::atomic<bool> running{true};

const std::string ARDUINO_PORT = "/dev/ttyACM0";
int serial_fd = -1;

bool setup_serial() {
    serial_fd = open(ARDUINO_PORT.c_str(), O_WRONLY | O_NOCTTY | O_SYNC);
    if (serial_fd < 0) {
        perror("ERROR: Can't open Arduino port");
        return false;
```

```
        }

        struct termios tty;
        if (tcgetattr(serial_fd, &tty) != 0) {
            perror("Error from tcgetattr");
            close(serial_fd);
            return false;
        }

        cfsetospeed(&tty, B115200);
        cfsetispeed(&tty, B115200);

        tty.c_cflag = (tty.c_cflag & ~CSIZE) | CS8;
        tty.c_iflag &= ~(IXON | IXOFF | IXANY | IGNBRK);
        tty.c_lflag = 0;
        tty.c_oflag = 0;
        tty.c_cc[VMIN] = 1;
        tty.c_cc[VTIME] = 1;
        tty.c_cflag |= (CLOCAL | CREAD);
        tty.c_cflag &= ~(PARENB | PARODD | CSTOPB | CRTSCTS);

        if (tcsetattr(serial_fd, TCSANOW, &tty) != 0) {
            perror("Error from tcsetattr");
            close(serial_fd);
            return false;
        }

        return true;
    }

    void send_to_arduino(float err_x, float err_y, float rad) {
        if (serial_fd < 0) return;

        char buffer[96];
        snprintf(buffer,    sizeof(buffer),    "X%.2f,Y%.2f,R%.2f\n",    err_x,
err_y,rad);

        write(serial_fd, buffer, strlen(buffer));
```

```
        fsync(serial_fd);
    }

    void *mmapBuffer(const libcamera::FrameBuffer::Plane &plane) {
        return mmap(nullptr, plane.length, PROT_READ, MAP_SHARED,
plane.fd.get(), 0);
    }


    void munmapBuffer(void *map, const libcamera::FrameBuffer::Plane &plane) {
        munmap(map, plane.length);
    }

     cv::Mat getting_mask(cv::Mat* img)
     {
        if (!img || img->empty()) return {};

        using namespace cv;

        Mat hsv, mask;

        cvtColor(*img, hsv, COLOR_BGR2HSV);
        Scalar lower(28, 76, 107);
        Scalar upper(54, 255, 255);
        inRange(hsv, lower, upper, mask);
        Mat kernel = getStructuringElement(MORPH_RECT, Size(5, 5));
        morphologyEx(mask, mask, MORPH_OPEN, kernel);

        return mask;

     }


    std::vector<cv::Point> getting_largest_countours(cv::Mat* mask){
        std::vector<std::vector<cv::Point>> contours;

        cv::findContours(*mask, contours, cv::RETR_TREE,
cv::CHAIN_APPROX_SIMPLE);
```

```
        if (contours.empty()) return{};
        auto largest_contour = *std::max_element(
                contours.begin(),
                contours.end(),
                [](const        std::vector<cv::Point>&        a,        const
std::vector<cv::Point>& b) {
                    return cv::contourArea(a) < cv::contourArea(b);
                });
        return largest_contour;
    }


    std::vector<float>  getting_center(std::vector<cv::Point>  contour,cv::Mat*
frame)
    {
        cv::Point2f center;
        float radius;
        if (contour.empty()){
            return std::vector<float>{0.0f, 0.0f, 250.0f};
        }

        cv::minEnclosingCircle(contour, center, radius);
        if (radius >= 20) {
            cv::circle(*frame, center, (int)radius, cv::Scalar(0, 255, 0), 2);
            cv::circle(*frame, center, 3, cv::Scalar(0,0,255), -1);
        }
        if (radius < 20){
         return std::vector<float>{0.0f, 0.0f, 250.0f};
        }

        float x = center.x;
        float y = center.y;
        std::vector<float> coordinates = {x,y,radius};
        return coordinates;
    }


    void requestComplete(libcamera::Request *request) {
```

```
        const libcamera::FrameBuffer *buffer = request->buffers().begin()->second;
        void *data = mmapBuffer(buffer->planes()[0]);

        int width = gConfig->size.width;
        int height = gConfig->size.height;

        cv::Mat frame(height, width, CV_8UC3, data);
        cv::Mat rgb = frame.clone();
        cv::resize(rgb, rgb, cv::Size(640, 480));

        cv::Mat mask =  getting_mask(& rgb);
        std::vector<cv::Point> contour =  getting_largest_countours(&mask);
        std::vector<float> coordinates = getting_center(contour, &rgb);
        // find error//
        float frame_center_x = 640 / 2.0f;
        float frame_center_y = 480 / 2.0f;

        float error_x = coordinates[0] - frame_center_x;
        float error_y = coordinates[1]- frame_center_y;
        float radius = coordinates[2];

        //std::cout << error_x << "," << error_y << "\n";
        send_to_arduino(error_x, error_y, radius);

        //cv::imshow("Camera", rgb);

        if (cv::waitKey(1) == 27)
            running = false;

        munmapBuffer(data, buffer->planes()[0]);
        request->reuse(libcamera::Request::ReuseBuffers);
        gCamera->queueRequest(request);
    }


    int main() {
```

```
using namespace libcamera;

if (!setup_serial()) return 1;

CameraManager cm;
cm.start();

gCamera = cm.cameras()[0];
gCamera->acquire();
gCamera->requestCompleted.connect(requestComplete);

auto config = gCamera->generateConfiguration({StreamRole::Viewfinder});
config->at(0).pixelFormat = formats::RGB888;
config->at(0).size.width = 1920;
config->at(0).size.height = 1080;
config->at(0).bufferCount = 4;

gCamera->configure(config.get());
gConfig = &config->at(0);

FrameBufferAllocator allocator(gCamera);
    for (StreamConfiguration &cfg : *config)
        allocator.allocate(cfg.stream());

    std::vector<std::unique_ptr<Request>> requests;
    for (StreamConfiguration &cfg : *config) {
        for (const auto &buffer : allocator.buffers(cfg.stream())) {
            auto request = gCamera->createRequest();
            request->addBuffer(cfg.stream(), buffer.get());
            requests.push_back(std::move(request));
        }
    }


gCamera->start();
for (auto &req : requests)
    gCamera->queueRequest(req.get());
```

```
    while (running)
        std::this_thread::sleep_for(std::chrono::milliseconds(10));

    cv::destroyAllWindows();
    gCamera->stop();
    gCamera->release();
    cm.stop();

    if (serial_fd >= 0) close(serial_fd);
    return 0;

}
```

Приложение 3. Программа управления роботом

```
#define LEFT_PWM  6
#define LEFT_DIR  7
#define RIGHT_PWM 5
#define RIGHT_DIR 4

const float Kp = 5.0;
const float Kd = 4.0;

float prev_error_x = 0;
const int base_speed = 100
unsigned long lastUpdate = 0;
const unsigned long timeout = 1000;

void setup() {
  pinMode(LEFT_PWM, OUTPUT);
  pinMode(LEFT_DIR, OUTPUT);
  pinMode(RIGHT_PWM, OUTPUT);
  pinMode(RIGHT_DIR, OUTPUT);

  Serial.begin(115200);
```

```
  Serial.println("Tracker ready");
}

void setMotorSpeed(int left, int right) {
  digitalWrite(LEFT_DIR, left >= 0);
  digitalWrite(RIGHT_DIR, right >= 0);
  analogWrite(LEFT_PWM, constrain(abs(left), 0, 200));
  analogWrite(RIGHT_PWM, constrain(abs(right), 0, 200));
}

void stopMotors() {
  analogWrite(LEFT_PWM, 0);
  analogWrite(RIGHT_PWM, 0);
}

void loop() {
  //setMotorSpeed(80, 240);
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    data.trim();

    int x_pos = data.indexOf('X');
    int y_pos = data.indexOf('Y');
    int R_pos = data.indexOf('R');
    int comma_pos = data.indexOf(',');

    if (x_pos == 0 && comma_pos > 1 && y_pos > comma_pos) {
       float error_x = data.substring(x_pos + 1, comma_pos).toFloat();
       float radius = data.substring(R_pos + 1).toFloat();
      if (radius >= 220){
        stopMotors();
      }
      else {

        float P_x = Kp * error_x;

        float D_x = Kd * (error_x - prev_error_x);
        prev_error_x = error_x;
```

```
        float correction_x = P_x + I_x + D_x;

        int left_speed = base_speed - correction_x;
        int right_speed = base_speed + correction_x;

        left_speed = constrain(left_speed, 70, 200);
        right_speed = constrain(right_speed, 70, 200);

        setMotorSpeed(left_speed, right_speed);

        lastUpdate = millis();
      }
    }
  }
  if (millis() - lastUpdate > timeout) {
    stopMotors();
  }
}
```

Литература

1. OpenCV: Computer Vision Library. Documentation: morphologyEx() [Электронный ресурс]. – URL: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f (дата обращения: 01.06.2024).
2. Гонсалес Р., Вудс Р. Цифровая обработка изображений : [пер. с англ.] / Р. Гонсалес, Р. Вудс. — 4-е изд. — Москва : Техносфера, 2022. — 1168 с. — ISBN 978-5-94836-677-7.
3. Ерофеев А.А. Теория автоматического управления : учебник для вузов / А.А. Ерофеев. — Санкт-Петербург : Политехника, 2021. — 416 с. — ISBN 978-5-7325-1204-3.
4. Bradski G. Learning OpenCV: Computer Vision with the OpenCV Library / G. Bradski, A. Kaehler. — 1st ed. — Sebastopol : O'Reilly Media, 2008. — 555 p. — ISBN 978-0-596-51613-0.
5. Szeliski R. Computer Vision: Algorithms and Applications / R. Szeliski. — 2nd ed. — London : Springer, 2022. — 925 p. — ISBN 978-1-84882-934-3. — DOI: 10.1007/978-1-84882-935-0.
6. Libcamera: Open Source Camera Stack [Электронный ресурс] // Official Documentation. — URL: https://libcamera.org (дата обращения: 10.06.2024).