Муниципальное общеобразовательное учреждение "Гимназия №3"

Индивидуальный практико-ориентированный проект

Тема «Алгоритмы отслеживания, выделения и вычисления расстояния между перемещающимися объектами в видеопотоке в качестве средства управления в реальном времени»

Работу выполнил
ученик 10 А класса
МОУ "Гимназия №3"
Коробов Алексей Сергеевич
Руководитель:
Цветков Дмитрий Юрьевич,
Преподаватель радиоэлектроники в
ГОАУ ДО ЯО ЦДЮТТ.

Ярославль, 2023

Оглавление:

I. Введение

Актуальность.

В наше время видеокамеры активно используются для наблюдения за людьми на улицах, охраны частной собственности. Их вешают вдоль дорог для слежки за соблюдением правил дорожного движения. Во многих передовых странах эти камеры обучили различать людей в реальном времени, используя машинное зрение и нейросети.

Я решил написать проект на вышеупомянутую мной тему по двум причинам. Во первых, я хочу подробнее изучить методы анализа видеопотока, ведь в связи с увеличением количества камер в нашей и других странах, в будущем возникнет потребность в специалистах по данной теме. Во вторых, используя полученные при изучении знания, я хочу доказать, что камеры можно использовать не только для слежки и наблюдения, но и в качестве инструмента управления каким-либо устройством (начиная от погрузчика, заканчивая точными хирургическими манипуляторами). По вопросу использования камер в качестве средства управления было найдено крайне мало информации в сети интернет, следовательно, этот проект может стать первой структурированной работой на эту тему.

Цель работы: Доказать пригодность использования алгоритмов отслеживания, выделения и вычисления расстояния между перемещающимися объектами в видеопотоке в качестве средства дистанционного управления.

Задачи работы:

1. Изучить информацию с официального сайта mediapipe (mediapipe – фреймворк, содержащий алгоритмы компьютерного зрения от Google) и OpenCV ((Open Source Computer Vision Library) — это открытая библиотека для работы с алгоритмами компьютерного зрения, машинным обучением и обработкой изображений), а также из других источников.

2. Попробовать написать свой алгоритм, анализирующий видеопоток в реальном времени, используя информацию и примеры, полученные после выполнения первой задачи.
3. Создать устройство, управляемое посредством написанного алгоритма.
4. Соединить программную и физическую часть проекта.
5. Осуществить проверку готового устройства (устройство должно управляться

посредством изменения картинки (положения руки на ней), получаемой веб-камерой, в реальном времени).

Гипотеза: Итоговый продукт проекта окажется функционален, что докажет возможность использования алгоритмов отслеживания, выделения и вычисления расстояния между перемещающимися объектами в видеопотоке в качестве средства управления в реальном времени.

Метод исследования: моделирование, измерение.

II. Основная часть

1) Что такое компьютерное и машинное зрение, чем они отличаются и где используются?

Для незнающего человека может показаться, что машинное зрение и компьютерное зрение это различные названия одной и той же технологии, однако это не совсем так. Компьютерное зрение — это область компьютерных наук, которая стремится расширить возможности компьютеров по идентификации и определению объектов и людей на изображениях и видео. В свою очередь машинное зрение – это применение технологии компьютерного зрения в различных сферах человеческой жизни, с целью автоматизации процессов, напрямую связанных с обработкой зрительной информации. Другими словами, машинное зрение – это одна из сфер применения технологии компьютерного зрения. Чаще всего, машинное зрение используется для автоматизации процесса производства товаров на фабриках и заводах, однако с каждым днем оно всё больше интегрируется в повседневную жизнь. Передвижение беспилотных автомобилей (например, автомобилей марки Tesla), функционирование систем наблюдения и контроля (распознавание людей, номерных знаков машин), работа приложений-переводчиков, автоматический анализ медицинских изображений (рентген, томография, УЗИ) – все эти процессы не могут функционировать без вышеупомянутого машинного зрения.

2) Что такое Mediapipe и OpenCV, как они помогают рядовым разработчикам в создании алгоритмов машинного зрения?

Как я ранее говорил, в настоящее время отрасль машинного зрения активно развивается, и многие крупные компании, так или иначе связанные с нею, заинтересованы в упрощении и структурировании программ, отвечающих за прием и обработку фото и

видеоинформации. Множество сторонних разработчиков и просто программистов-любителей, желающих поподробнее изучить эту прогрессирующую отрасль, также нуждались в простых и эффективных инструментах, помогающих в разработке вышеописанных программ.

В связи с этими двумя факторами “на заре” развития машинного зрения корпорация Intel в 2006 году представила OpenCV (Open Source Computer Vision Library) - это программная библиотека компьютерного зрения и машинного обучения с открытым исходным кодом. Она была создана для обеспечения общей инфраструктуры для приложений компьютерного зрения и ускорения использования машинного восприятия в коммерческих продуктах. Эта библиотека упрощает использование и модификацию кода для коммерческих организаций. Библиотека содержит более 2500 оптимизированных алгоритмов, которые включают в себя полный набор как классических, так и современных алгоритмов компьютерного зрения и машинного обучения. Эти алгоритмы могут использоваться для обнаружения и распознавания лиц, идентификации объектов, отслеживания движущихся объектов, извлечения 3D-моделей объектов, поиска похожих изображений из базы данных изображений, отслеживания движения глаз, распознавания пейзажей и т.д. Сообщество пользователей OpenCV насчитывает более 47 тысяч человек, а количество загрузок превышает 18 миллионов. Библиотека широко используется в компаниях, исследовательских группах и правительственных органах.

Одновременно с этим компания Google занимается разработкой собственного фреймворка под названием Mediapipe (Фреймворк (с англ. framework — «каркас, структура») — заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код, он задает структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта). MediaPipe - это фреймворк созданный для упрощения и ускорения обработки данных временных рядов, таких как видео, аудио и др. Разработчик может создать прототип, не углубляясь в написание алгоритмов и моделей машинного зрения, используя существующие компоненты. Эта структура может использоваться для различных приложений для обработки изображений и мультимедиа таких как обнаружение объектов, распознавание лиц, отслеживание рук, отслеживание нескольких рук и сегментация волос. MediaPipe поддерживает различные аппаратные и операционные платформы, такие как Android, iOS и Linux. Изначально он был разработан для анализа видео и аудио на YouTube в режиме реального времени (и право на его использование было только у корпорации Google). В

2019 году после публичного релиза MediaPipe открыл новый мир возможностей для исследователей и разработчиков. Ключевыми особенностями Mediapipe от OpenCV являются улучшенная проработанность и оптимизация алгоритмов (программы, написанные при помощи решения от Google стабильно работают даже в умных холодильниках), отвечающих за анализ фото и видеопотока, а также куда больший выбор встроенных вспомогательных инструментов.

Не смотря на то, что OpenCV и Mediapipe обладают схожим функционалом, они фактически не являются конкурентами друг для друга. В большинстве случаев в программах используются оба решения вместе.OpenCV в реальном времени считывает и преобразует видеопоток для его дальнейшей обработки каким-либо алгоритмом Mediapipe (изначально создавался для обработки уже готовых выложенных видео на YouTube, из-за чего не может самостоятельно считывать видео в реальном времени (например, с веб-камеры)). После этого выбранный алгоритм работает с принятым видеопотоком.

3) Написание алгоритма с использованием инструментария OpenCV и Mediapipe.

Начать анализ написанной мною программы предлагаю с самой главной её части, а именно со считывания и обработки получаемой с веб-камеры информации (скриншот представлен в Приложении №1). В этом отрывке кода мы создаём объект захвата видео cap при помощи одной из функций OpenCV. Также мы создаем объекты hands (руки) и draw (рисунок) для определения и отрисовки “скелета” ладони в реальном времени, используя встроенные в Mediapipe нейросети.

В зацикленной части моей программы идёт анализ и считывание с созданных ранее объектов. Большинство важных моментов этого отрывка программы я описал в виде комментариев в коде (скриншот представлен в Приложении №2). В комментариях упоминается рисунок 1, данный рисунок содержится в Приложении №3.

Перейдём к функции, отвечающей за калибровку получаемых координат, которая представлена в Приложении №4. Комментарии, оставленные мной в программном коде, дают достаточное представление о работе данной функции.

Остальная часть моей программы отвечает за подготовку и отправку данных по последовательному порту в Arduino. Эти данные будут в дальнейшем преобразованы в углы для сервоприводов моего манипулятора при помощи функции get_instruction() (см. Приложение №5). После получения этих инструкций мы отправляем уже готовые углы

сервоприводам по последовательному порту при помощи функции write_instruction() (см. Приложение №6).

На картинке, представленной в Приложении №7, мы можем наблюдать нахождение длин каждого из пальцев (в дальнейшем совокупность их длины, а также углов между ними будет преобразована в углы для манипулятора при помощи самописной функции get_instruction()).

3) Создание простейшего манипулятора.

Для данного проекта я собрал устройство, по типу своей конструкции относящееся к промышленным манипуляторам с пятью степенями свободы. На рисунке, представленном в Приложении №8, можно увидеть схему промышленного манипулятора с шестью степенями свободы. Однако из-за желания уделить больше времени программной части проекта, которая является основной, в итоговом варианте я решил оставить только 5 осей (основную кинематику промышленного манипулятора), однако в программе учтена шестая ось, а также возможность установки захвата. Фотография моего манипулятора представлена в Приложении №9.

Питание манипулятора осуществляется от сетевого зарядного устройства с напряжением 5V и выходной силой тока 2.4A. Мой манипулятор построен на основе платформы Arduino. Arduino – это инструмент для проектирования электронных устройств, более плотно взаимодействующих с окружающей физической средой, чем стандартные персональные компьютеры, которые фактически не выходят за рамки виртуальности. Arduino применяется для создания электронных устройств с возможностью приема сигналов от различных цифровых и аналоговых датчиков, которые могут быть подключены к нему, и управления различными исполнительными устройствами. Проекты устройств, основанные на Arduino, могут работать самостоятельно или взаимодействовать с программным обеспечением на компьютере, что собственно и требуется в моем случае. Мой манипулятор использует плату Arduino Uno (см. Приложение №10), а программа для приёма инструкций от функции write_instruction(), написанная в Arduino IDE (Arduino IDE — это программная среда разработки, использующая C++ и предназначенная для программирования всех плат ряда Arduino), представлена на скриншоте (см Приложение №11).

4) Соединение программной и физической части, проверка работоспособности моего устройства.

Подключаем все провода от сервоприводов (находятся в узлах моего манипулятора) к Arduino через расширительную плату, которая позволяет подвести к моторам дополнительное питание от зарядного устройства (см. Приложение №12). После чего плату Arduino Uno подключаем к компьютеру по USB кабелю, в Arduino IDE выбираем порт, к которому подключена плата. В моём случае это последовательный порт COM4 (см. Приложение №13). Запоминаем этот порт. После чего в Arduino IDE загружаем программу в плату, получаем надпись “Загрузка завершена” (см. Приложение №14), что означает отсутствие синтаксических ошибок в коде программы и исправность платы Arduino UNO. Переходим к моему алгоритму отслеживания ладони. В коде программы находим строку 112, где изменяем номер порта на COM4 (см. Приложение №15). Запускаем программу. После запуска программы (shift+F10) на экране отображается информация с видеокамеры с просьбой поместить руку в область её видимости (“Show hand”). Это показано на скриншоте в Приложении №16. После того как рука попадает в область видимости запускается функция calibrate(), которая просит показать сначала ладонь, а затем кулак (см. Приложения №17 и №18). Всё это время манипулятор находится в стартовом положении, описанном в строке 115(см Приложения №19 и №20) После того как кулак продемонстрирован, нужно нажать клавишу “a”(триггер окончания калибровки), и начнётся процесс управления манипулятором в реальном времени (см. видео “Управление манипулятором”). Программа остылает в терминал те же данные, что и в последовательный порт COM4, благодаря чему мы можем отследить ошибки в её работе (см. Приложение №21). Для более комфортного управления мною были предусмотрены два жеста: при сгибе среднего пальца манипулятор опускается, при согнутом безымянном пальце манипулятор поднимается. Движения вправо, влево, вперед и назад зависят непосредственно от положения ладони.

III. Заключение

Устройство, полученное в ходе работы над темой этого проекта, является достаточно простым в управлении, а также легко настраивается под любого пользователя при помощи калибровки. Не стоит забывать, что данное устройство является лишь прототипом, из-за чего оно не лишено таких недостатков как недостаточная плавность работы, неустойчивость конструкции, что обусловлено недостатком материальных и временных ресурсов. Однако эти минусы легко поправимы, а программа, которая лежит в основе всего моего проекта, работает стабильно, понятна для стороннего программиста, разбирающегося в компьютерном зрении, дискретна, завершаема и результативна. Совокупность прототипа и программы образует мой продукт проекта, который отвечает тем задачам, которые я для себя ставил, что говорит о его функциональности. Итоговый продукт проекта оказался функционален, что доказало возможность использования алгоритмов отслеживания, выделения и вычисления расстояния между перемещающимися объектами в видеопотоке в качестве средства управления в реальном времени.

IV. Список литературы

1) Книга “Learning OpenCV 4 Computer Vision with Python” в соавторстве Джозефа Хоуза и Джо Минитино.
2) Официальный сайт Mediapipe https://developers.google.com/mediapipe
3) Официальный сайт OpenCV https://opencv.org/
4) Официальная документация для OpenCV https://docs.opencv.org/
5) Официальный репозиторий Mediapipe с примерами и пояснениями https://google.github.io/mediapipe/

V. Приложения

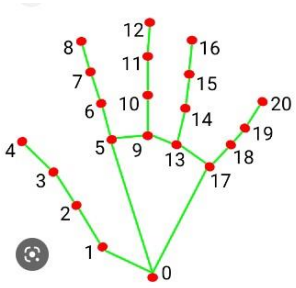
Приложение №1

```
cap = cv2.VideoCapture(0)  # Камера
ret, frame = cap.read()
(h, w) = frame.shape[:2]
hands = mp.solutions.hands.Hands(max_num_hands=1)  # Объект ИИ для определения ладони
draw = mp.solutions.drawing_utils  # Для рисования ладони
```

Приложение №2

```
while True:
    # Закрытие окна на клавишу ESC
    if cv2.waitKey(1) & 0xFF == 27:
        break
    if cv2.waitKey(1) & 0xFF == ord('r'):
        calibrate_flag = 1

    success, image = cap.read()  # Считываем изображение с камеры
    # image = cv2.flip(image)  # Отражаем изображение для корректной картинки
    imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Конвертируем в rgb
    results = hands.process(imageRGB)  # Работа mediapipe

    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks:
            x, y, z = [], [], []
            for id, lm in enumerate(handLms.landmark): #записываем в массивы координаты точек руки от нулевой до двадцатой (как на рисунке 1)
                h, w, c = image.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                x.append(lm.x * w)
                y.append(lm.y * h)
                z.append(lm.z * w)

            if len(x) == 21: #если координаты для всех точек записаны (те ладонь полностью видна), то переходим к функции калибровки
                median_filter(find_lengths(x, y, z), last_lengths)
                median_filter(find_coords(x, y, z), last_coords)
                if calibrate_flag != 3:
                    calibrate(x, y, z)
                #else:
                    #write_instruction(median_filter(find_lengths(x, y, z), last_lengths), find_coords(x, y, z))
            else:
                cv2.putText(image, 'Show hand', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0))
                #отображаем надпись "Show hand" если калибровка не произведена

            draw.draw_landmarks(image, handLms, mp.solutions.hands.HAND_CONNECTIONS)  # Рисуем ладонь
    else:
        cv2.putText(image, 'Show hand', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0))
        #отображаем надпись "Show hand" если ладонь в камере не обнаружена

    cv2.imshow("Hand", image)  #Отображаем всю картинку
```

Приложение №3



Приложение №4

```
def calibrate(x, y, z):
    global calibrate_flag, palm, fist, up, down, last_lengths
    if calibrate_flag == 1: #если в кадре видна рука, то:
        cv2.putText(image, 'Show palm', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0))  #просим продемонстрировать ладонь
        if cv2.waitKey(1) & 0xFF == ord('a'): #после нажатия клавиши A отправляем наши значения в медианный фильтр
            palm = median_filter(find_lengths(x, y, z), last_lengths)
            calibrate_flag = 2
    elif calibrate_flag == 2: #если мы уже показали ладонь, то:
        cv2.putText(image, 'Show fist', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0)) #просим продемонстрировать кулак
        if cv2.waitKey(1) & 0xFF == ord('a'): #после нажатия клавиши A отправляем наши значения в медианный фильтр
            fist = median_filter(find_lengths(x, y, z), last_lengths)
            calibrate_flag = 3
```

Приложение №5

```
def get_instruction(lengths, coords):
    global height
    th = [coords[0], coords[1]]
    index = [coords[2], coords[3]]

    if all([(lengths[i] - fist[i]) / (palm[i] - fist[i]) < 0.8 for i in range(5)]):
        print('STOP')
        return

    if (lengths[1] - fist[1]) / (palm[1] - fist[1]) < 0.8 and (lengths[2] - fist[2]) / (palm[2] - fist[2]) > 0.8:
        height += 3
    elif (lengths[2] - fist[2]) / (palm[2] - fist[2]) < 0.8 and (lengths[1] - fist[1]) / (palm[1] - fist[1]) > 0.8:
        height -= 3

    if (lengths[4] - fist[4]) / (palm[4] - fist[4]) < 0.8 \
            and (lengths[1] - fist[1]) / (palm[1] - fist[1]) < 0.8 \
            and (lengths[2] - fist[2]) / (palm[2] - fist[2]) < 0.8:
        servoAngle[-1] += 3
    elif (lengths[4] - fist[4]) / (palm[4] - fist[4]) > 0.8 \
            and (lengths[1] - fist[1]) / (palm[1] - fist[1]) < 0.8 \
            and (lengths[2] - fist[2]) / (palm[2] - fist[2]) < 0.8:
        servoAngle[-1] -= 3

    height = min(2 * hand, max(0, height))

    X, Y = w // 2, h - 100

    x, y = int(index[0]), int(index[1])
    print(x, y)
    cv2.line(image, (x, y), (X, Y), (255, 255, 255), 3)
    shift = sqrt((X - x) ** 2 + (Y - y) ** 2)
    alfa = acos((x-X) / shift)
    alfa = int(180 * alfa / pi)
    if y > Y:
        alfa = 360 - alfa
    move(shift / 5, height, alfa)
    return servoAngle
```

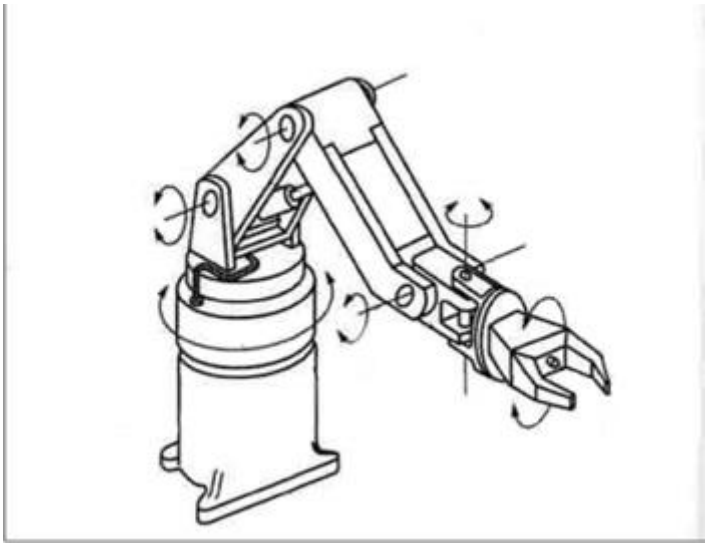
Приложение №6

```
def write_instruction(lengths, coords):
    get_instruction(lengths, coords)
    print(servoAngle)

    a = 0
    x1, y1 = 0, 0
    for i in range(0, 3):
        a += servoAngle[i]
        x2 = int(cos(a / 180 * pi) * hand) + x1
        y2 = int(-sin(a / 180 * pi) * hand) + y1

        cv2.line(image, (int(x1 * cos((servoAngle[3]) * pi / 180)) + 500, y1 + 200),
                 (int(x2 * cos((servoAngle[3]) * pi / 180)) + 500, y2 + 200), (255, 255, 255), 3)
        x3, y3 = x1, y1
        x1, y1 = x2, y2

    x1 = int(x3 * cos(servoAngle[3] * pi / 180) + 500)
    y1 = int(x3 * -sin(servoAngle[3] * pi / 180) + 400)
    x2 = x1 + int(hand * cos((servoAngle[3] + servoAngle[4]) * pi / 180))
    y2 = y1 + int(hand * -sin((servoAngle[3] + servoAngle[4]) * pi / 180))
    cv2.line(image, (500, 400), (x1, y1), (255, 255, 255), 3)
    cv2.line(image, (x1, y1), (x2, y2), (255, 255, 255), 3)
    my_serial.write(bytes([254]))
    my_serial.write(bytes([max(0, min(180, servoAngle[3]))]))
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, servoAngle[0]))]))
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, abs(servoAngle[1])))]))
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, 270 - servoAngle[2]))]))
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, servoAngle[4] + 90))]))  # надо подумать
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, servoAngle[6]))]))
    # print(my_serial.read(1), end='')
    my_serial.write(bytes([max(0, min(180, servoAngle[5]))]))
    # print(my_serial.read(1), end='')
    print(my_serial.read())
    time.sleep(0.01)
```

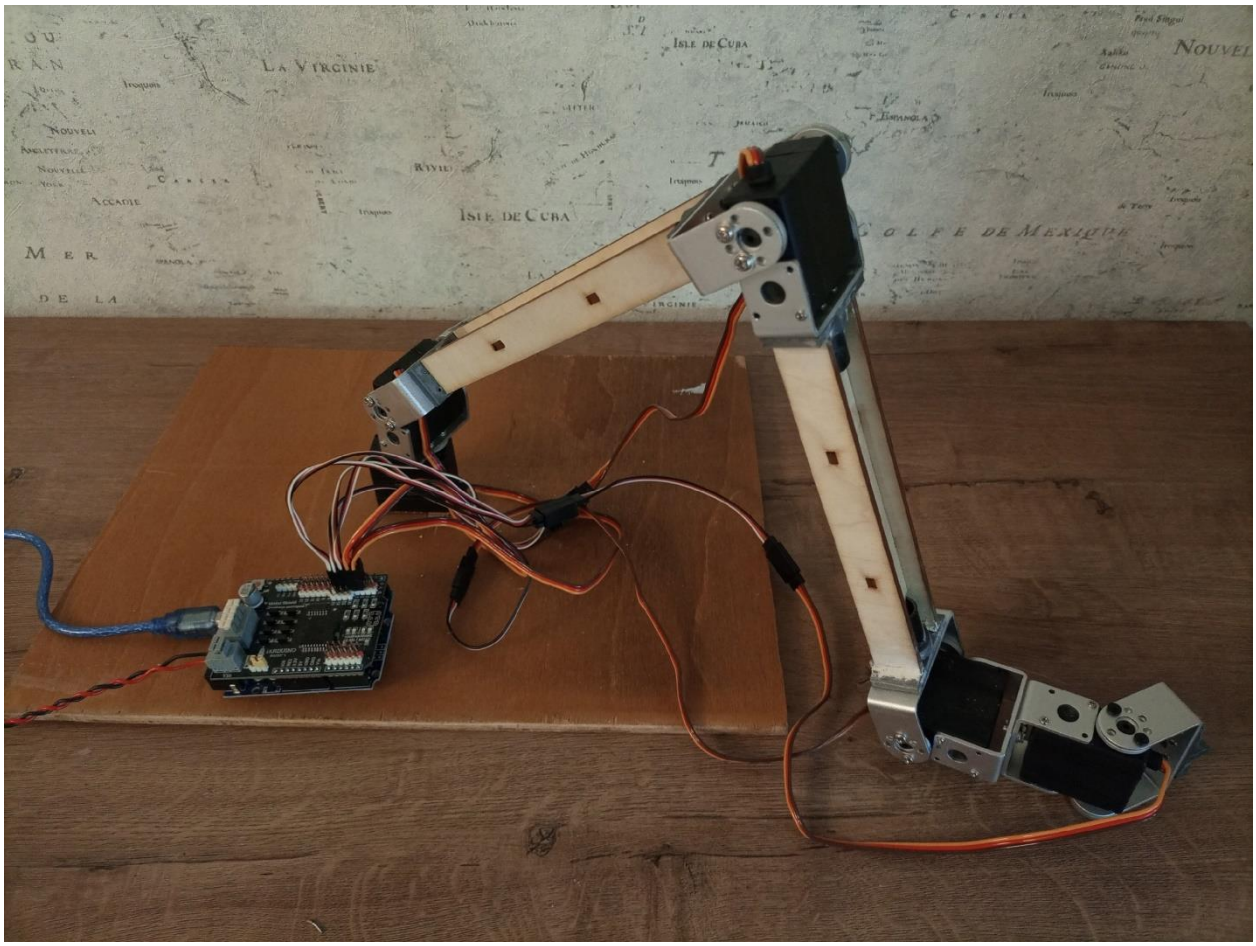
Приложение №7

```
def length_index_finger(x, y, z):
    x1 = x[8] - x[0]
    y1 = y[8] - y[0]
    z1 = z[8] - z[0]

    return hypot(x1, y1, z1) / length_hand(x, y, z)


def length_middle_finger(x, y, z):
    x1 = x[12] - x[0]
    y1 = y[12] - y[0]
    z1 = z[12] - z[0]
    return hypot(x1, y1, z1) / length_hand(x, y, z)


def length_ring_finger(x, y, z):
    x1 = x[16] - x[0]
    y1 = y[16] - y[0]
    z1 = z[16] - z[0]
    return hypot(x1, y1, z1) / length_hand(x, y, z)


def length_pinky_finger(x, y, z):
    x1 = x[20] - x[0]
    y1 = y[20] - y[0]
    z1 = z[20] - z[0]
    return hypot(x1, y1, z1) / length_hand(x, y, z)


def length_thumb_finger(x, y, z):
    x1 = x[4] - x[0]
    y1 = y[4] - y[0]
    z1 = z[4] - z[0]
    return hypot(x1, y1, z1) / length_hand(x, y, z)
```

Приложение №8



Приложение №9

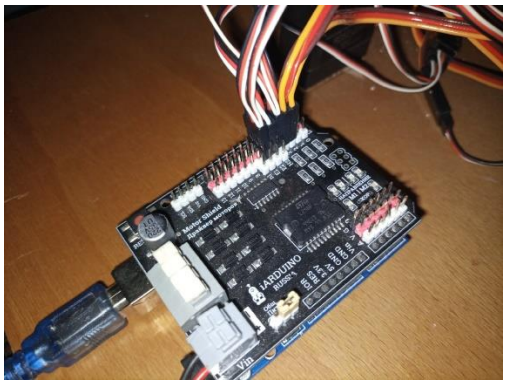


Приложение №10

Приложение №11

```
#include <ServoSmooth.h>

ServoSmooth serv[7];

uint32_t servoTimer;
uint32_t turnTimer;

void setup() {
  for (int i = 0; i < 7; ++i) {
    serv[i].attach(i + 2);
    serv[i].setSpeed(1000);
    serv[i].setAccel(0.5);
  }
   Serial.begin(115200);
}

void loop() {
  if (millis() - servoTimer >= 20) {
    servoTimer += 20;
    for (byte i = 0; i < 7; i++) {
      serv[i].tickManual();   //
    }
  }

  if (millis() - turnTimer >= 100) {
    turnTimer = millis();
    if (Serial.available() && Serial.read() == 254) {
      for (int i = 0; i < 7; ++i) {
        while (!Serial.available());
        int v = Serial.read();
        serv[i].setTargetDeg(v);
      }
      Serial.write('a');
    }
  }
}
```

Приложение №12



Приложение №13

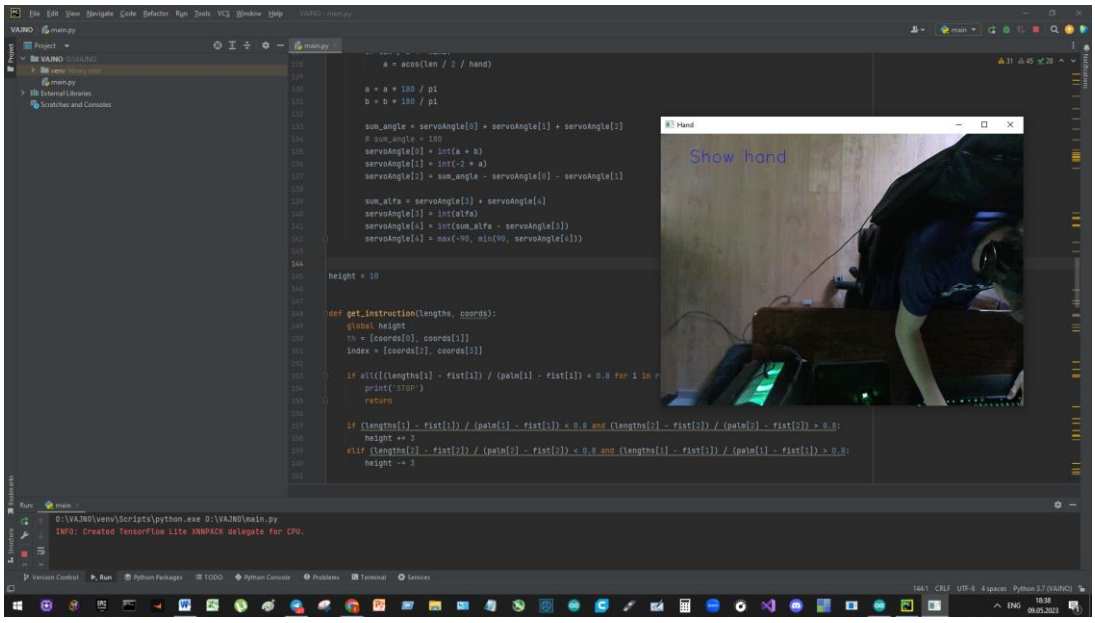
Приложение №14

```
Загрузка завершена.
Скетч использует 6550 байт (20%) памяти устройства. Всего доступно 32256 байт.
```

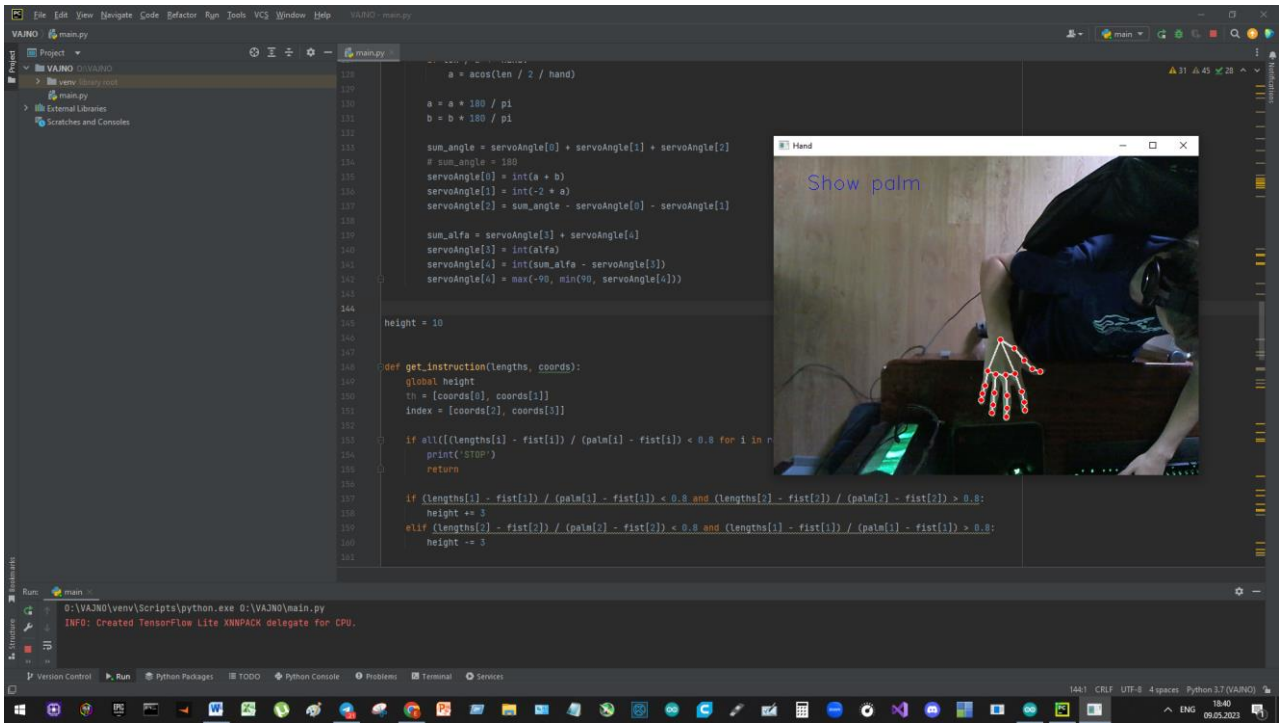
Приложение №15

```
112     my_serial = serial.Serial('COM4', 115200)
```

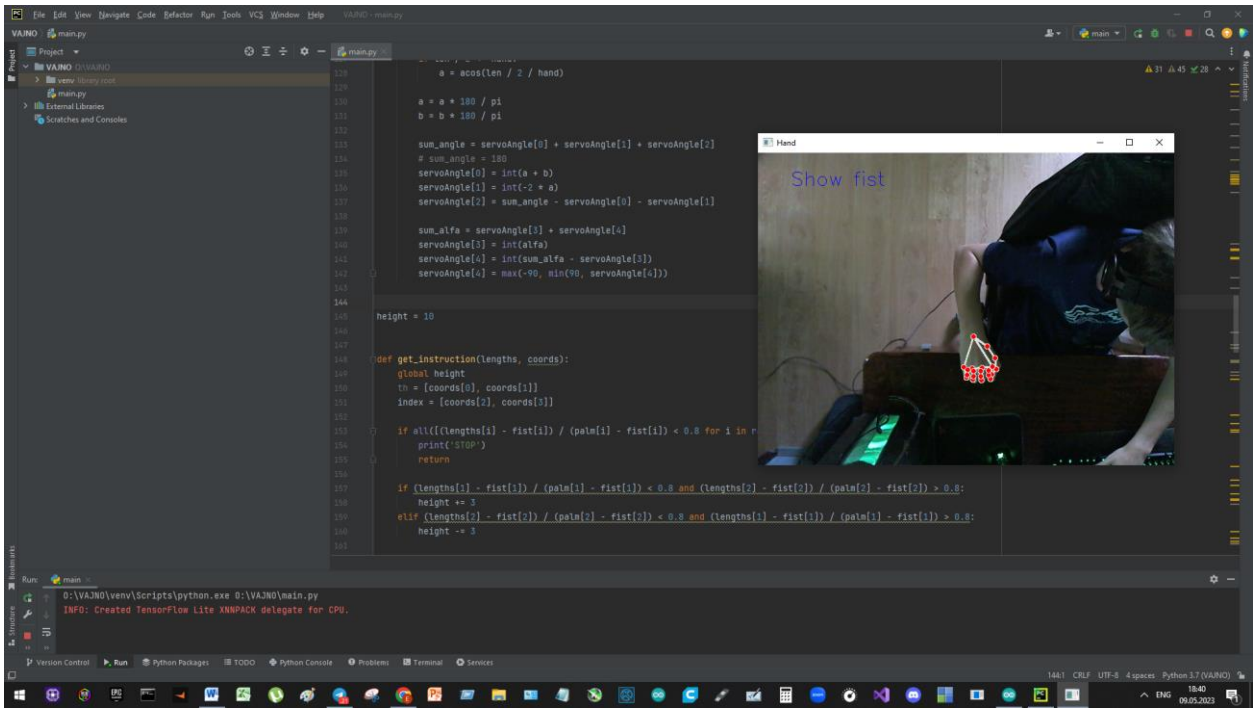
Приложение №16



Приложение №17



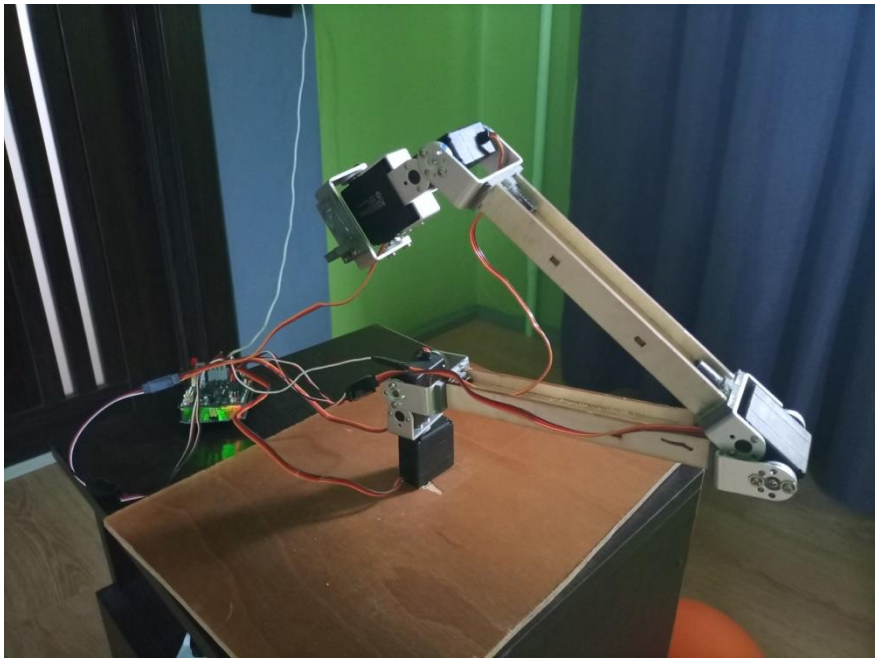
Приложение №18



Приложение №19

```
115    servoAngle = [0, -90, 270, 90, 0, 0, 90]
```

Приложение №20



Приложение №21