Visual Gesture Controlled IoT Car

Remember the opening scene of movie 'Project Almanac'? Controlling a drone with hand? Make it yourself, and to simplify, let's control a CAR

Difficulté **Moyen**

Durée **3 heure(s)**

Catégories **Énergie, Robotique, Transport**

Coût **50 EUR (€)**

Sommaire

Introduction

Have you watched the movie 'Project Almanac'? Which was released in the year 2015. If not, then let me brief you a scene about it.
In the movie, the main character wishes to get into MIT and therefore, builds a project for his portfolio. The project was about a drone, that could be controlled using a 2.4GHz remote controller, but when the software application on the laptop was run, the main character was seen controlling the drone with his hands in the air! The software application used a webcam to track the the movement of the character's hand movements.

Matériaux

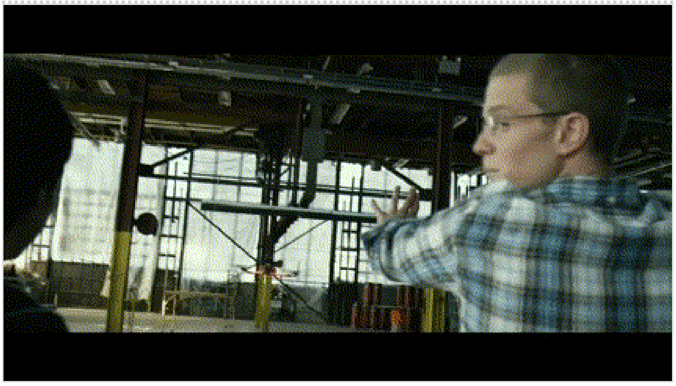
Outils

https://www.hackster.io/akshayansinha/visual-gesture-controlled-iot-car-590603#code

Étape 1 - Introduction

Have you watched the movie 'Project Almanac'? Which was released in the year 2015. If not, then let me brief you a scene about it.

In the movie, the main character wishes to get into MIT and therefore, builds a project for his portfolio. The project was about a drone, that could be controlled using a 2.4GHz remote controller, but when the software application on the laptop was run, the main character was seen controlling the drone with his hands in the air! The software application used a webcam to track the the movement of the character's hand movements.



Étape 2 - Custom PCB on your Way!

Modern methods of development got easier with software services. For hardware services, we have limited options. Hence PCBWay gives the opportunity to get custom PCB manufactured for hobby projects as well as sample pieces, in very little delivery time

Get a discount on the first order of 10 PCB Boards. Now, PCBWay also offers end-to-end options for our products including hardware enclosures. So, if you design PCBs, get them printed in a few steps!

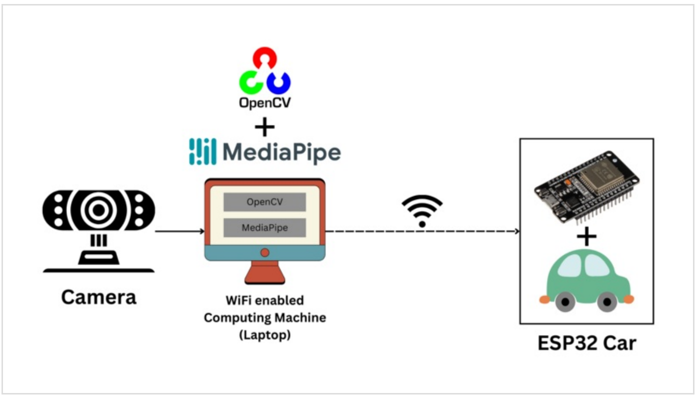
Étape 3 - Getting Started

As we already saw, this technology was well displayed in the movie scene. And the best part is, in 2023 it is super easy to rebuild it with great tools like OpenCV and MediaPipe. We will control a machine but with a small change in the method, than the one the character uses to let the camera scan his fingers.

He used color blob stickers on his fingertips so that the camera could detect those blobs. When there was a movement in the hands, which was visible from the camera, the laptop sent the signal to the drone to move accordingly. This allowed him to control the drone without any physical console.

Using the latest technological upgrades, we shall make a similar, but much simpler version, which can run on any embedded Linux system, making it portable even for an Android system. Using OpenCV and MediaPipe, let us see how we can control our 2wheeled battery-operated car, over a Wi-Fi network with our hands in the air!

Étape 4 - OpenCV and MediaPipe



OpenCV is an open-source computer vision library primarily designed for image and video analysis. It provides a rich set of tools and functions that enable computers to process and understand visual data. Here are some technical aspects.

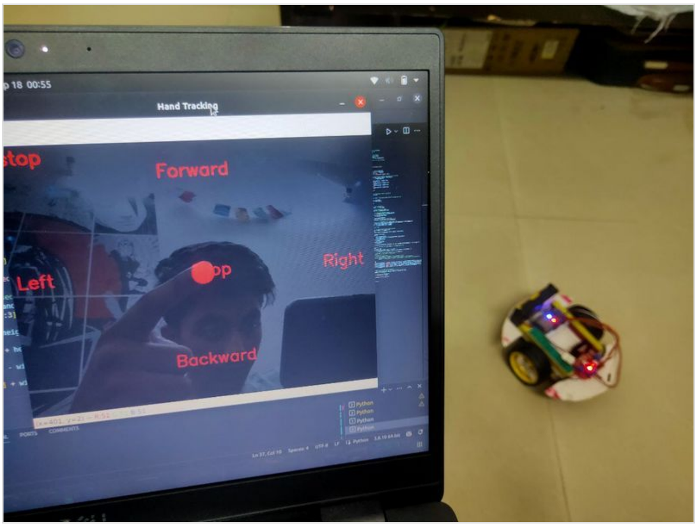
- Image Processing: OpenCV offers a wide range of fuctions for image processing tasks such as filtering, enhancing, and manipulating images. It can perform operations like blurring, sharpening, and edge detection.
- Object Detection: OpenCV includes pre-trained models for object detection, allowing it to identify and locate objects within images or video streams. Techniques like Haar cascades and deep learning-based models are available.
- Feature Extraction: It can extract features from images, such as keypoints and descriptors, which are useful for tasks like image matching and recognition.
- Video Analysis: OpenCV enables video analysis, including motion tracking, background subtraction, and optical flow.

MediaPipe is an open-source framework developed by Google that provides tools and building blocks for building various types of real-time multimedia applications, particularly those related to computer vision and machine learning. It's designed to make it easier for developers to create applications that can process and understand video and camera inputs. Here's a breakdown of what MediaPipe does:

- Real-Time Processing: MediaPipe specializes in processing video and camera feeds in real-time. It's capable of handling live video streams from sources like webcams and mobile cameras.
- Cross-Platform: MediaPipe is designed to work across different platforms, including desktop, mobile, and embedded devices. This makes it versatile and suitable for a wide range of applications.
- Machine Learning Integration: MediaPipe seamlessly integrates with machine learning models, including TensorFlow Lite, which allows developers to incorporate deep learning capabilities into their applications. For example, you can use it to build applications that recognize gestures, detect facial expressions, or estimate the body's pose.
- Efficient and Optimized: MediaPipe is optimized for performance, making it suitable for real-time applications on resource-constrained devices. It takes advantage of hardware acceleration, such as GPU processing, to ensure smooth and efficient video processing.

From above if you have noticed, this project will require one feature from each of these tools, to be able to make our project work. Video Analysis from OpenCV and HandTracking from MediaPipe. Let us begin with the environment to be able to work seamlessly.

Étape 5 - Hand Tracking and Camera Frame UI



As we move ahead, we need to know how to use OpenCV and Mediapipe to detect hands. For this part, we shall use the Python library.
Make sure you have Python installed on the laptop, and please run below command to install the necessary libraries -
Run the command to install the libraries -

```
python -m pip install opencv-python mediapipe requests numpy
```

To begin with the the control of car from the camera, let us understand how it will function -

- The camera must track the hands or fingers to control the movement of the car. We shall track the index finger on the camera for that.
- Based on the location of finger with reference to the given frame, there will be forward, backward, left, right and stop motion for the robot to function.
- While all the movements are tracked on real time, the interface program should send data while reading asynchronously.

To perform the above task in simple steps, methods used in the program have been simplified in beginner's level. Below is the final version!
As we see above, the interface is very simple and easy to use. Just move your index finger tip around, and use the frame as a console to control the robot. Read till the end and build along to watch it in action!

Étape 6 - Code - Software

Now that we know what the software UI would look like, let us begin to understand the UI and use HTTP request to send signal to the car to make actions accordingly.
Initializing MediaPipe Hands

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_drawing = mp.solutions.drawing_utils
```

Here, we initialize the MediaPipe Hands module for hand tracking. We create instances of mp.solutions.hands and mp.solutions.drawing_utils, which provide functions for hand detection and visualization.
Initializing Variables

```
tracking = False
hand_y = 0
hand_x = 0
prev_dir = ""
URL = "http://projectalmanac.local/"
```

In this step, we initialize several variables that will be used to keep track of hand-related information and the previous direction.
A URL is defined to send HTTP requests to the hardware code of ca
Defining a Function to Send HTTP Requests

```
def send(link):
    try:
        response = requests.get(link)
        print("Response ->", response)
    except Exception as e:
        print(f"Error sending HTTP request: {e}")
```

This step defines a function named send that takes a link as an argument and sends an HTTP GET request to the specified URL. It prints the response or an error message if the request fails.

These are the initial setup steps. The following steps are part of the main loop where video frames are processed for hand tracking and gesture recognition. I'll explain these steps one by one:

MediaPipe Hands Processing

```
ret, frame = cap.read()
frame = cv2.flip(frame, 1)
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = hands.process(rgb_frame)
```

Inside the loop, it captures a frame from the camera (cap.read()) and flips it horizontally (cv2.flip) to mirror the image.
The code converts the captured frame to RGB format (cv2.cvtColor) and then uses the MediaPipe Hands module to process the frame (hands.process) for hand landmark detection. The results are stored in the results variable.

Hand Landmarks and Tracking

```
if results.multi_hand_landmarks:
    hand_landmarks = results.multi_hand_landmarks[0]
    index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
    hand_y = int(index_finger_tip.y * height)
    hand_x = int(index_finger_tip.x * width)
    tracking = True
```

This section checks if hand landmarks are detected in the frame (results.multi_hand_landmarks). If so, it assumes there's only one hand in the frame and extracts the y-coordinate of the index finger tip. It updates hand_y and hand_x with the calculated coordinates and sets tracking to True.

Direction Calculation

```
frame_center = (width // 2, height // 2)
if trackin
    direction = find_direction(frame, hand_y, hand_x, frame_center)
    if(direction != prev_dir):
            try:
                link = URL+direction
                http_thread = threading.Thread(target=send, args=(link,))
                http_thread.start()
            except Exception as e:
                print(e)
            prev_dir = direction
            print(direction)
```

In this step, the code calculates the center of the frame and, if tracking is active, it uses the find_direction function to calculate the direction based on the hand's position. The direction is stored in the direction variable.
We used current direction and previous direction variables. It helps in keeping a semaphore of sending only one HTTP request for every change in command. Then overall store it in a single URL to send the HTTP request.

Visualization

```
opacity = 0.8
cv2.addWeighted(black_background, opacity, frame, 1 - opacity, 0, frame)
cv2.imshow("Project Almanac", frame)
```

If tracking is active, this section of the code adds visual elements to the frame, including a filled circle representing the index finger tip's position and text indicating the detected direction.

The code blends a black background with the original frame to create an overlay with adjusted opacity. The resulting frame is displayed in a window named "Project Almanac".

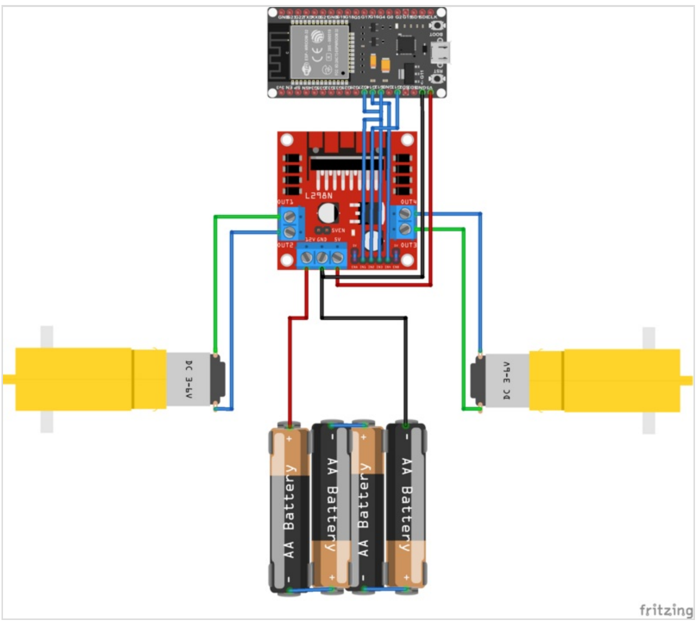
Étape 7 - Code - Hardware

Now that we are done with the software side code, let us look into
the software side code -

Importing Libraries:

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WebServer.h>
```

In this section, the code includes necessary libraries for WiFi communication (WiFi.h), setting up mDNS (ESPmDNS) for local network naming, and creating a web server using the WebServer library.



Defining Pin Constants:

```
int LeftA = 33;   // IN1
int LeftB = 25;   // IN2
int RightA = 26;  // IN3
int RightB = 27;  // IN4
```

Here, the code defines constants for pin numbers corresponding to motor control pins (presumably for a robotic project). These pins will control the movement of motors.

Setting Up Wi-Fi Credentials:

```
const char* ssid = " ";      // Enter SSID here
const char* password = " ";  // Enter Password here
```

You need to fill in your Wi-Fi network's SSID and password here to connect the ESP8266 device to your local Wi-Fi network.

Configuring Motor Control Pins:

```
 pinMode(LeftA, OUTPUT);
 pinMode(LeftB, OUTPUT);
 pinMode(RightA, OUTPUT);
 pinMode(RightB, OUTPUT);
 pinMode(2, OUTPUT);
```

In this part, the code sets the motor control pins (LeftA, LeftB, RightA, RightB) as OUTPUT pins, presumably to control motors for a robotic project. It also sets pin 2 as an OUTPUT, possibly for controlling an indicator LED.

Connecting to Wi-Fi:

```
Serial.begin(115200);
 delay(100);
 Serial.println("Connecting to ");
 Serial.println(ssid);

 // Connect to your local Wi-Fi network
 WiFi.begin(ssid, password);

 // Check if the device is connected to the Wi-Fi network
 while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.print(".");
 }

 // Display connection status and IP address
 Serial.println("");
 Serial.println("WiFi connected..!");
 Serial.print("Got IP: ");
 Serial.println(WiFi.localIP());

 digitalWrite(2, HIGH); // Turn on a blue LED to indicate a connected Wi
Fi
```

This part of the code initiates a connection to the specified Wi-Fi network using the provided SSID and password. It waits until the device successfully connects to the Wi-Fi network and then displays the IP address. Additionally, it turns on an LED on pin 2 to indicate a successful connection.

Setting up mDNS (Multicast DNS):

```
if (!MDNS.begin("projectalmanac")) {
    Serial.println("Error setting up MDNS responder!");
    while(1) {
      delay(1000);
    }
  }
  Serial.println("mDNS responder started");
```

Here, the code sets up mDNS with the hostname "projectalmanac." This allows the device to be reachable on the local network using the hostname instead of an IP address.

Defining HTTP Server Endpoints:

```
  server.on("/", handle_OnConnect);
  server.on("/left", left);
  server.on("/right", right);
  server.on("/forward", forward);
  server.on("/backward", backward);
  server.on("/stop", halt);
  server.onNotFound(handle_NotFound);
```

This part defines different HTTP server endpoints that can be accessed via URLs. For example, "/left" will trigger the left function when accessed.

Starting the Web Server:

```
server.begin();
  Serial.println("HTTP server started");
  MDNS.addService("http", "tcp", 80);
}
```

The code starts the web server, making it available for handling HTTP requests on port 80. It also registers the HTTP service with mDNS.

Handling Client Requests:

```
void loop() {
  server.handleClient();
}
```

In the loop function, the server continuously handles client requests, responding to various endpoints defined earlier.

HTTP Request Handling Functions: The code defines several functions (forward, backward, left, right, halt, handle_OnConnect, handle_NotFound) that are called when specific endpoints are accessed. These functions are responsible for controlling motors and responding to client requests. The HTML page provides information about available commands and instructions for interacting with the device.

Étape 8 - Project Almanac in action!

Now that we have understood the code sequence, let us see the work!

We can further add more features if you'd like to. Rest, the UI is simple enough to handle, which comes with not many features, but important one's.