Helmet Motion Project

Charbel El Hajj 54238

Elie Fayad 54204

Mario Bouabboud 54381

Supervisor: Dr. Gilbert Habib

Table of Contents

I. Introduction

The goal of this Internet of Things (IoT) project is to control a turret using the motion of a person's head or by detecting humans through a camera mounted on it. The feed is then transmitted to a remote server which then displays a live footage, the person's location and the motion type with the help of a web interface.

Some applications for this project include but are not limited to:

- **Military Applications:** A military turret designed for vehicles could be controlled by the driver's helmet motion or automatically by detecting humans. This also enables the supervision of the soldier's location.
- **Security Applications:** Enabling security guards to monitor rooms more smoothly and easily using this technology and could be improved to follow suspicious individuals.
- **Medical Applications:** Doctors may utilize this technology in medical treatments and procedures. For instance, it may facilitate the Endoscopy procedure and automate health threat detections in it.

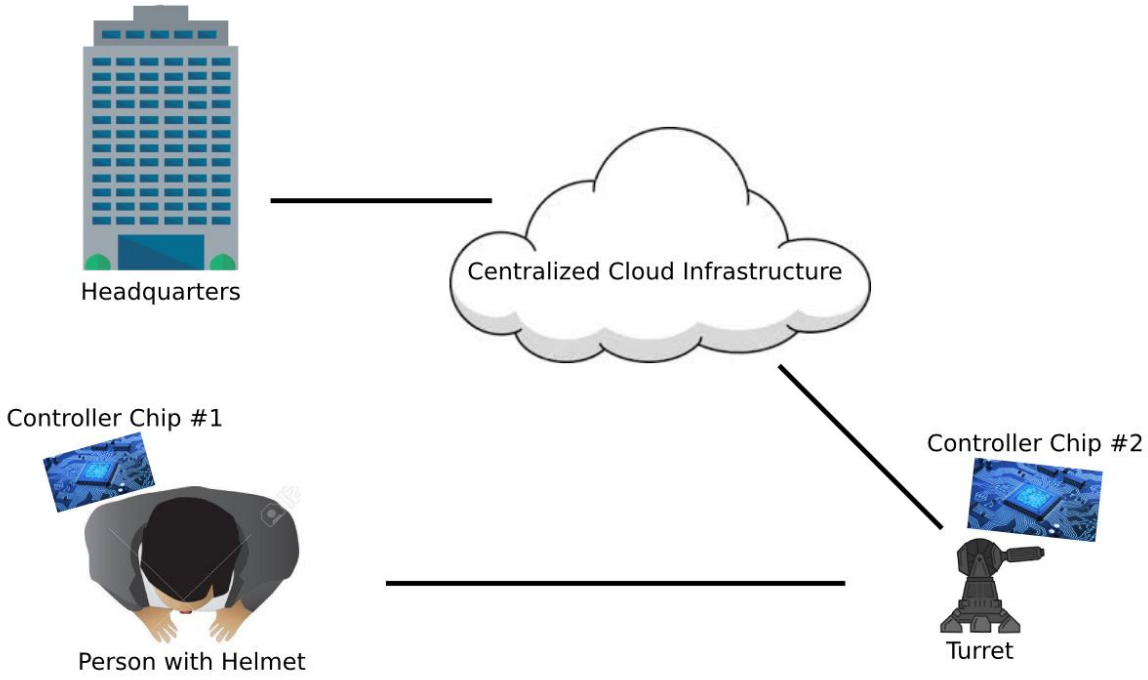


Figure 1 - Project Concept

II. Components & Platforms

➢ Controllers:

➢ **Raspberry Pi 3:** The Raspberry Pi is a low cost, minicomputer that can be plugged into a screen monitor, and uses a standard keyboard and mouse. It is a capable little device that enables people to explore computing, and to learn how to program in languages like C and Python. The Raspberry Pi is used to enable the movement of servos and motion tracking in addition to GPS signal transfer.

✓ **Arduino UNO:** Arduino is an open-source electronics platform based on easy-to-use hardware and software. Physical add-on modules enable Arduinos to read inputs (i.e. light on a sensor, ultrasonic waves, temperature fluctuations) and turn it into an output (i.e. activating a motor, turning on an LED). You can tell your board what to do by sending a set of instructions to the microcontroller on the board via the Arduino IDE using mostly C as a programming language. The Arduino is used to read the various motion and GPS input and tracking.

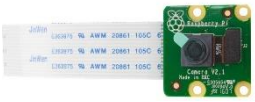
➢ Modules:

➢ **HC-06 Bluetooth Module *(3.3V – 5V)*:** A Bluetooth module designed for wireless serial communication. Once it is paired to another Bluetooth device such as PC, smart phones and tablet, its operation becomes transparent to the user. All data received through the serial input is immediately transmitted over the air. When the module receives wireless data, it is sent out through the serial interface exactly at it is received as bytes. Minor code configuration is needed in order for the module to function.

➢ **MPU6050 Gyroscope Module *(3.3V – 5V)*:** A sensor based on MEMS (micro electro mechanical systems) technology. This device helps determine orientation based on gravity, its angle of rotation and angular velocity.

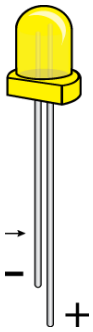
➢ **SG-90 Servo Motor *(3.0V – 7.2V)*:** Tiny and lightweight motor with high output power. Servos can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller.

- **Neo-6M GPS Module *(3.3V – 5V)*:** GPS module which provides a strong satellite search capability and can be used to determine position, time, and speed. This module does not work indoors, it must be in an open space with a clear view to the sky.



- **Pi-Cam *(250mA)*:** Image sensor custom designed add-on for Raspberry Pi, featuring a fixed focus lens.

- **Push Button:** Internally, two legs of the button are connected to each other but not connected to the opposite two creating an open circuit. When pressed, the circuit becomes connected and breaks when released.



- **Light Emitting Diode (LED – *1.8V – 3.3V*):** Small light that works with relatively little power.

c) Platforms:

- ✓ **Electronic Modules:** Provides the Arduino and the Raspberry Pi with the required inputs and outputs

- ✓ **Arduino IDE:** Uses C as a programming language to control the Arduino

- ✓ **Angular:** Front-end development platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling and integrated best practices to solve development challenges

- ✓ **Firebase:** Google back-end service that helps the quick development of high-quality apps

- ✓ **TensorFlow:** Open source machine learning library for research and production

- ✓ **OpenCV:** Open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Contains algorithms that can be used to detect and recognize faces, identify objects…

- ✓ **Python:** The programming language used on the Raspberry Pi

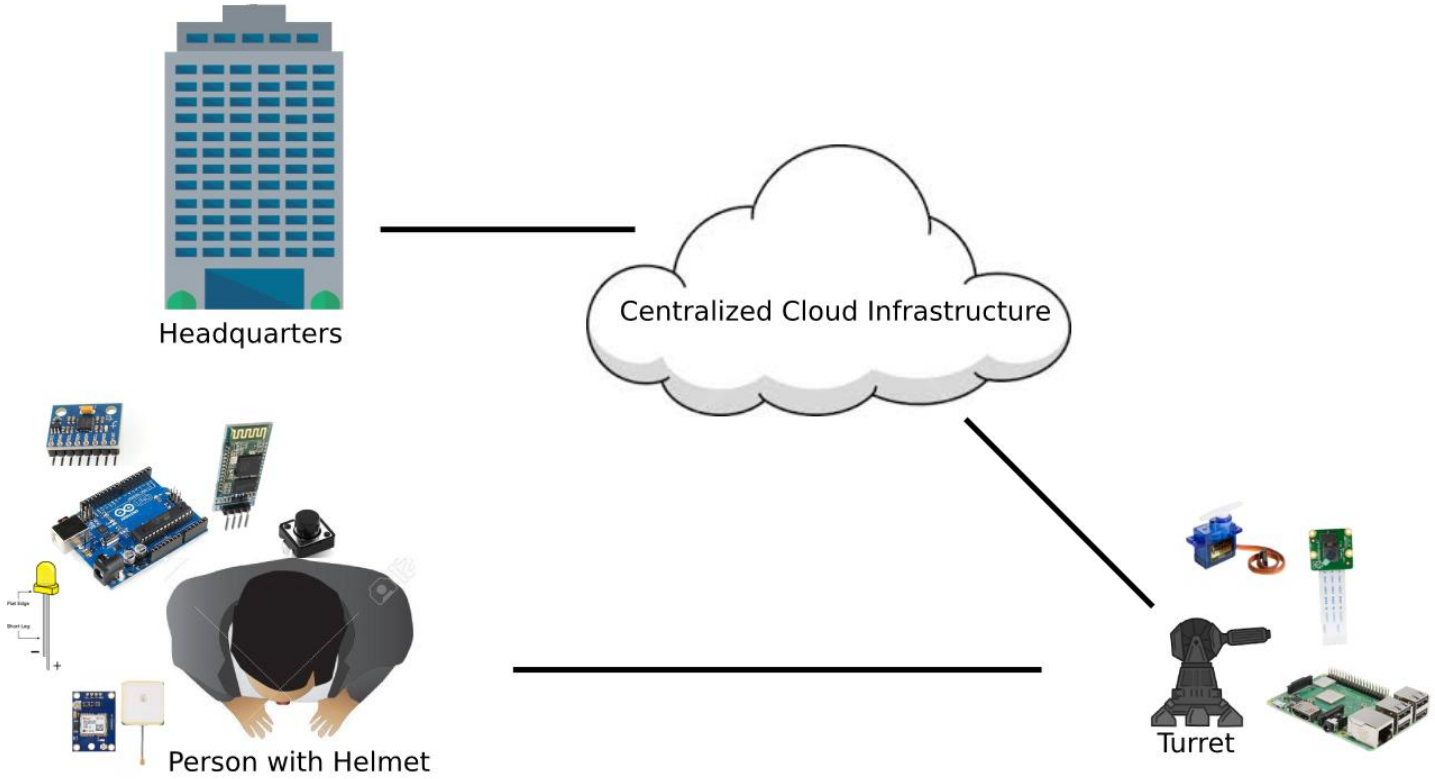


Figure 2 - Component Distribution Overview

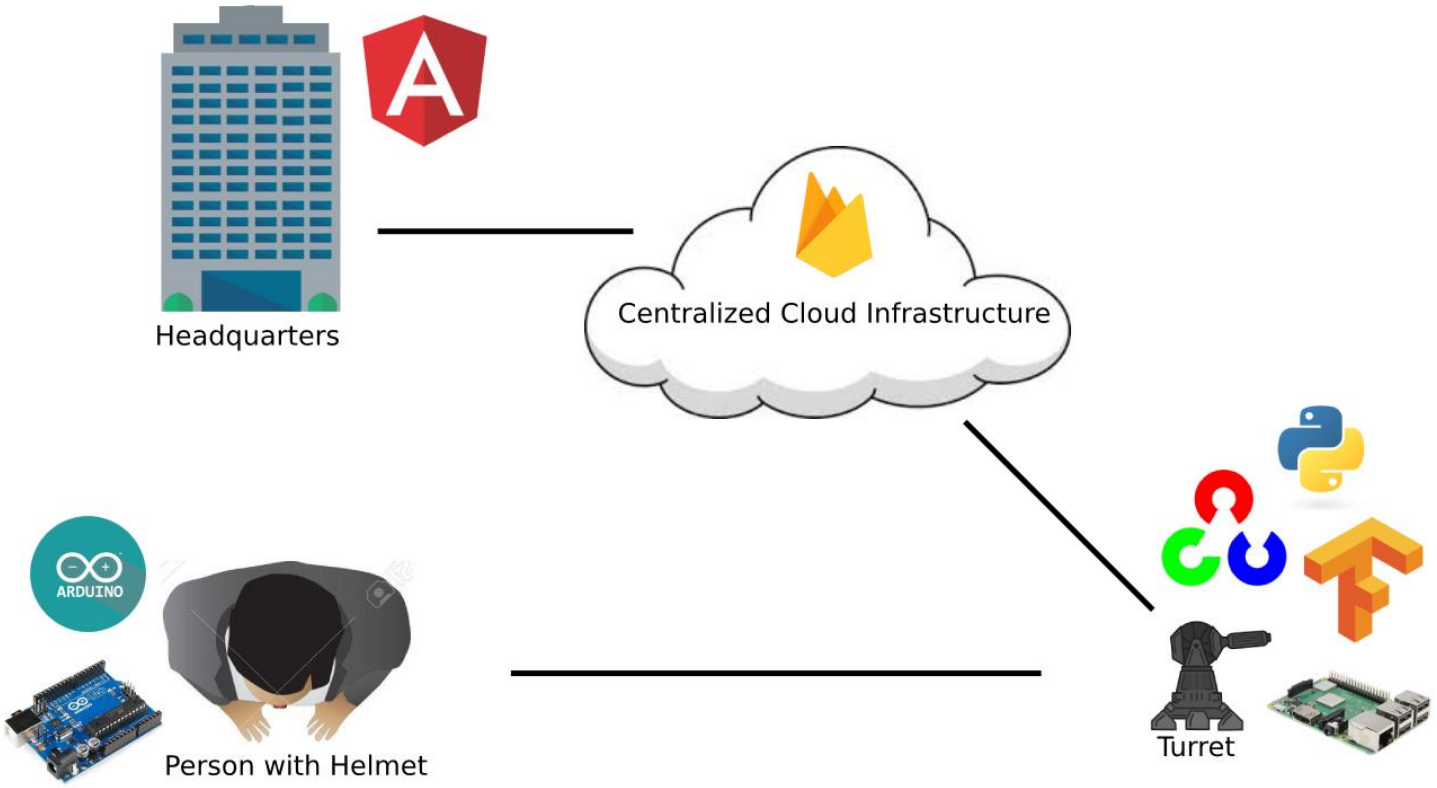


Figure 3 - Platform Distribution Overview

III. Project Structure & Connections

➢ Arduino:

1. Connections:

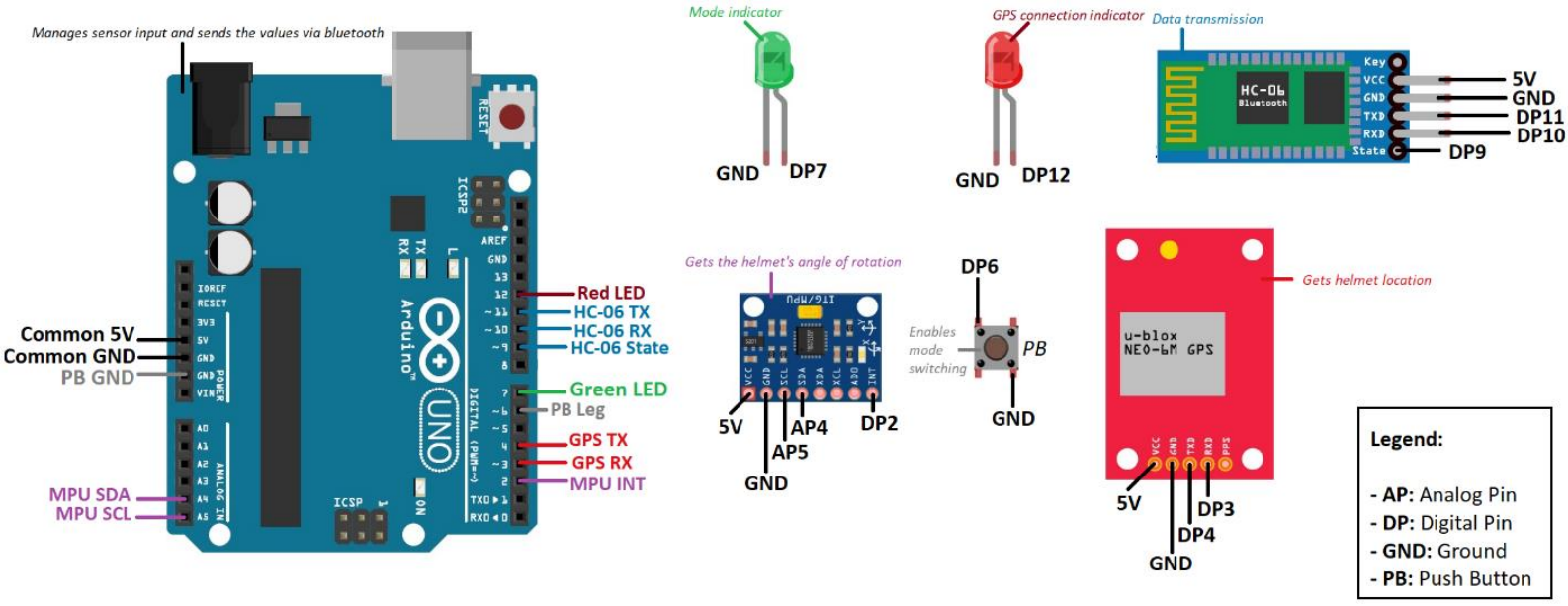


Figure 4.1 - Arduino Connections

2. Design:

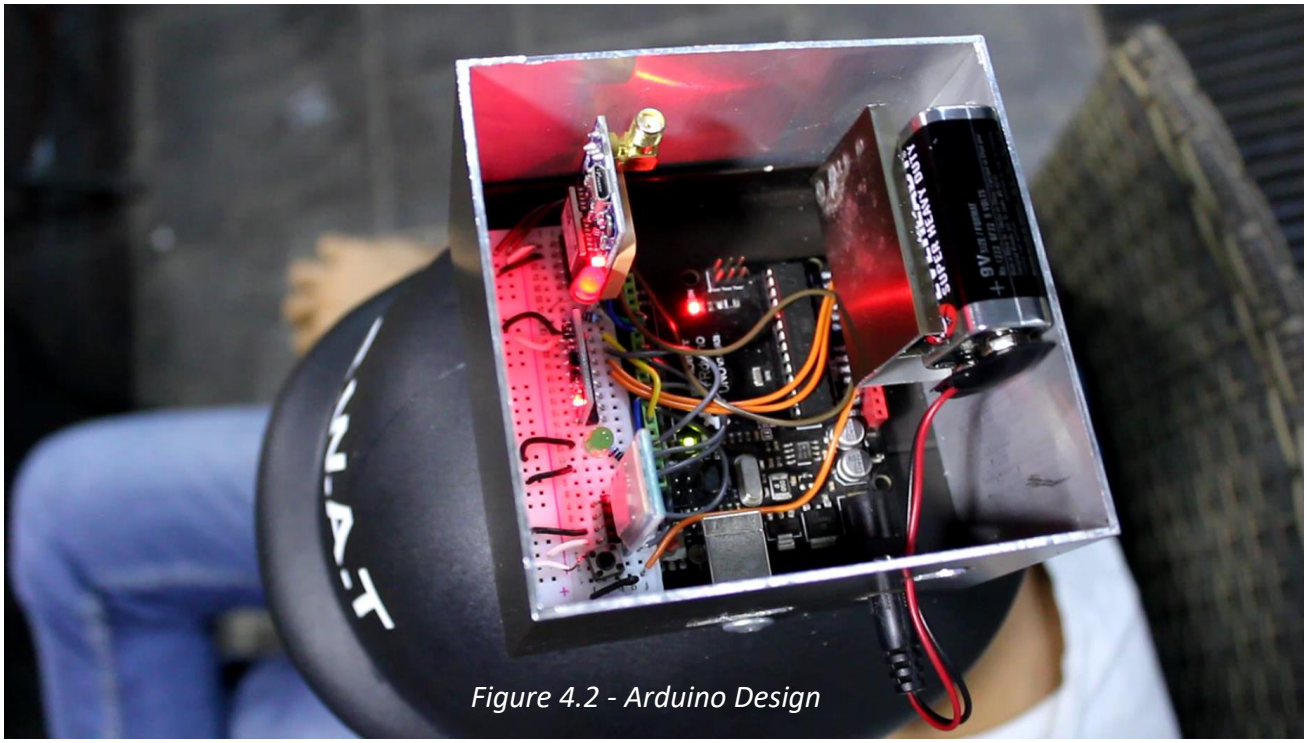


Figure 4.2 - Arduino Design

➢ Raspberry Pi:

1. Connections:

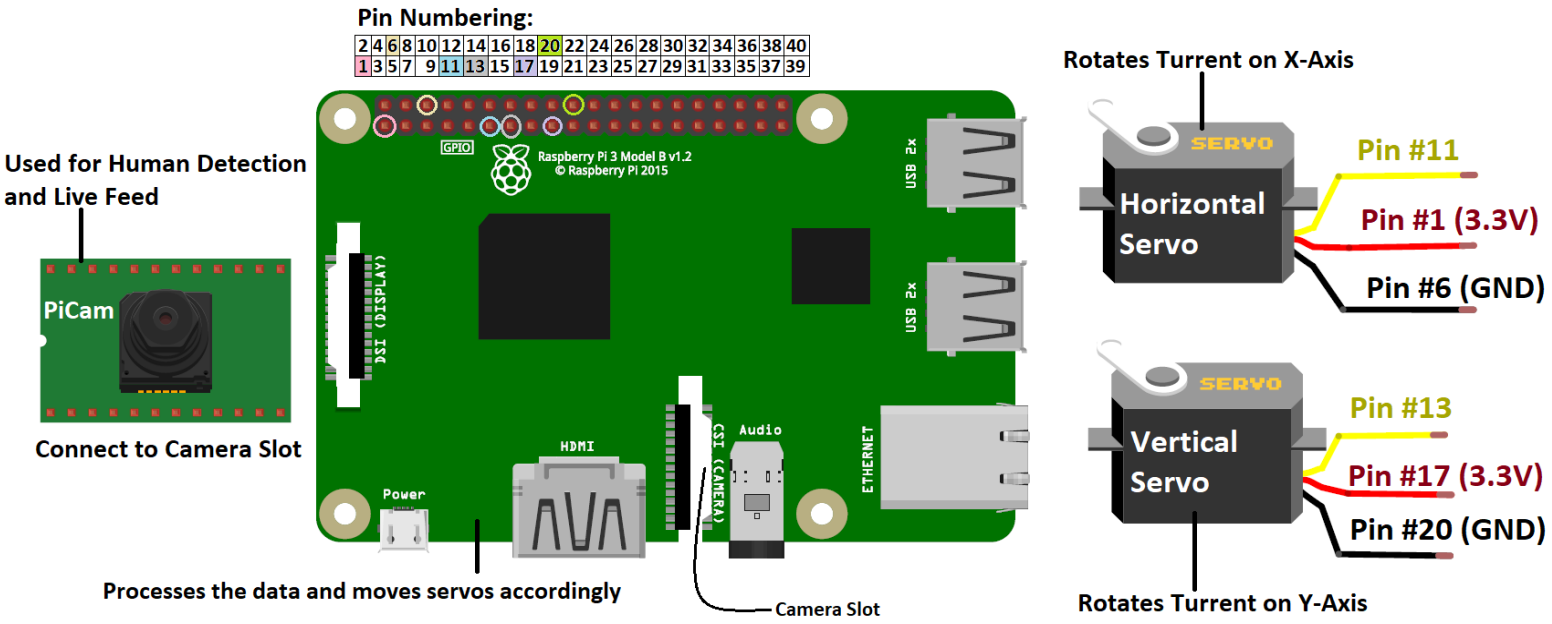


Figure 5.1 - Raspberry Pi Connections

2. Design:

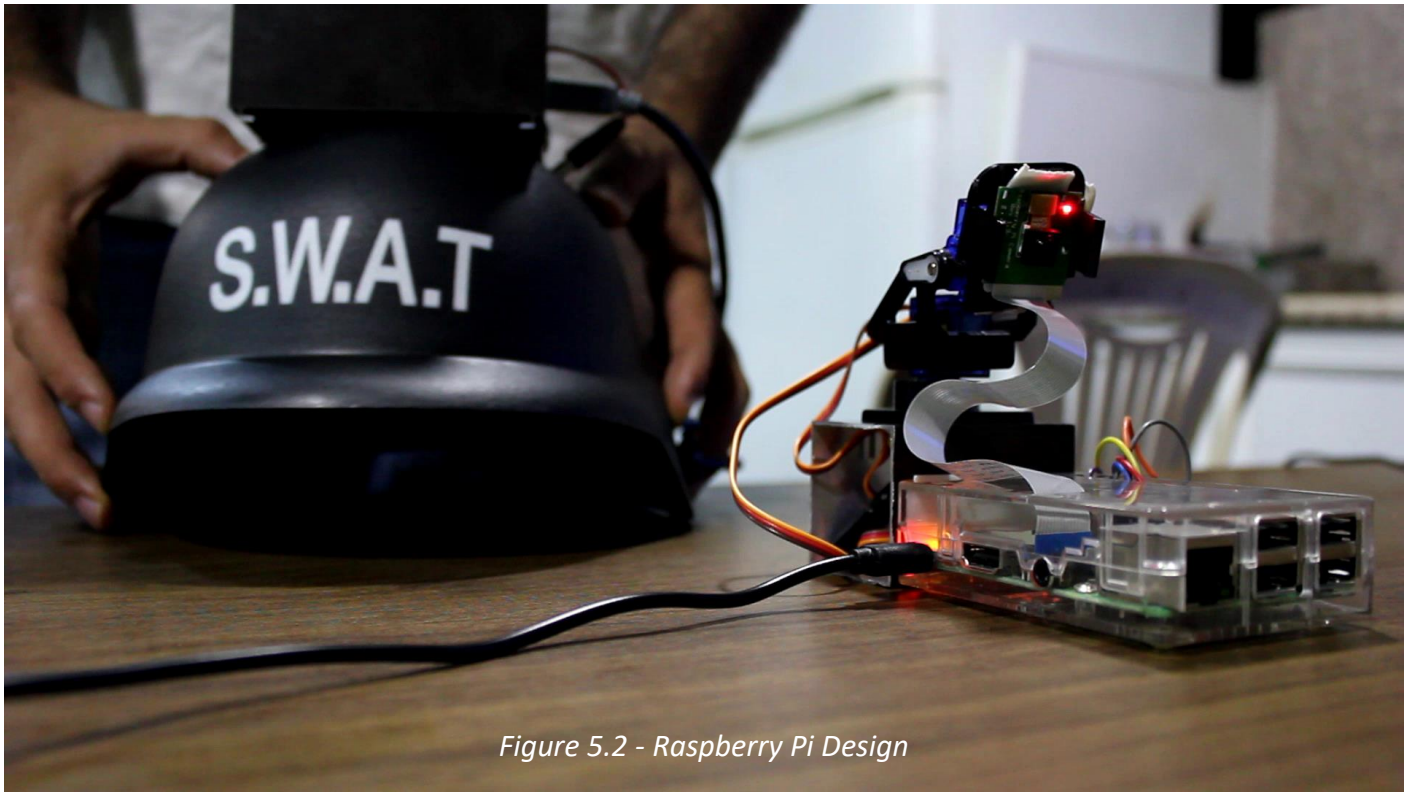


Figure 5.2 - Raspberry Pi Design

➢ Web Panel:

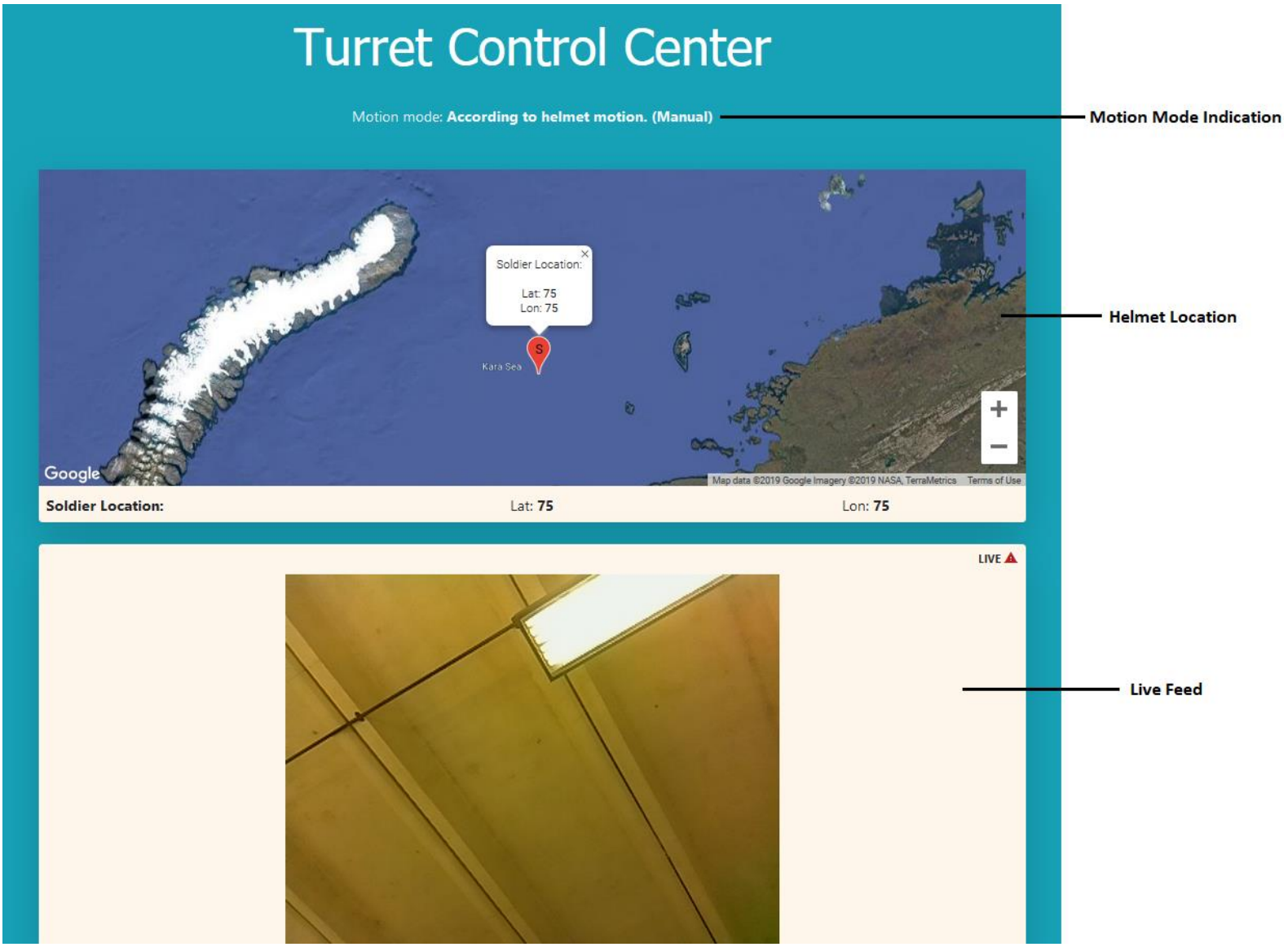


Figure 6 - Web Panel

IV. Project Challenges

Hardware Connectivity and Communication:

Being Computer Science students, there is a lack in the appropriate formation of hardware manipulation and programming. The hardest challenges were connecting the devices and modules to extract valid data and communicating between them.

- ✓ **Gyroscope Calibration and Value Input:**

  The first problem encountered was discovering how the MPU6050 module works. After learning that this module measures the acceleration and the rotation (angular velocity) on the X, Y and Z axes, an appropriate method to extract accurate angles was done.

  In order to read correct values, two methods can be applied:

  1. **Manual interaction** with the module: which consists of writing code from scratch in order to read the values of the gyroscope
  2. Interaction with the module via **specialized libraries:** which consists of using already published code in order to read the gyroscope values.

  In the beginning, several libraries were tested for the extraction of data. After applying test cases, the library "MPU6050_tockn" was chosen since it calibrates the gyroscope before usage and reads both the raw and converted data.

  Later on a second problem arose, an inaccuracy in the servos' movements was detected after ~45 seconds of continuous execution. This was due to a drifting effect that happens in the MPU6050 module.

  Gyroscopes are affected because of noise and temperature changes while accelerometers are affected due to vibration and other no-gravity accelerations. The drift is inevitable and numerous attempts to fix and reduce it were applied.

  The best solution was modifying the previously mentioned library to calculate the error rate before execution and using the obtained value in the angles' calculations. This allowed reading somewhat constant angles with slower change over time.

  Because this last method is highly dependent on the time passed since the last read, the mode change feature disrupted the process. What was done to fix it is measuring the time the person pressed the button and then subtracting this value from the original time interval when getting the new angles.

- ✓ **Servo Manipulation:**

  Some problems with the wiring of the servos were encountered on the Raspberry Pi due to the difficult pin identification. After some research, the servos started working but were jittering on a resting position. Another library had to be used which allowed **faster** and more efficient access to the Servos and stopped the malfunction.

  In the Arduino: To reduce power consumption and improve servo movement, a function was created which rounds gyroscope input before transmitting said values to the Raspberry Pi.

  In the Raspberry Pi: Angles must be converted to a Pulse Width Modulation signal (PWM signal) which is equivalent to the time (in milliseconds) of the servo rotation. For the conversion, the following formula was deduced:

SG-90 servos have a range of 180°

Minimum PWM of SG-90: 500 ms
Maximum PWM of SG-90: 2500 ms

Assume angles belong to [-90°, 90°]

PWM signal of SG-90 is linear => can combine angles and signals as points in 2D space (angle, pulse)

Line Equation: $y - y_{point} = slope * (x - x_{point})$

Suppose: A(-90, 500) & B(90, 2500)

$slope = \frac{y_B - y_A}{x_B - x_A} = \frac{100}{9}$
$\Rightarrow y - y_A = slope * (x - x_A)$
$\Rightarrow y = \frac{100}{9} * (x + 90) + 500$

Where **y=pulse** and **x=angle**

- ✓ **Bluetooth Communication:**

  The initial Bluetooth module used was the HC-05 which was then upgraded to the HC-06 module due to its inability to connect and send data to another device.

  With this upgrade, the connection between the Arduino and the Raspberry Pi was successful. In order to secure the connection, the Raspberry Pi was configured to only connect to the HC-06 Bluetooth module of the Arduino.

  In the Arduino: The data was initially sent as one packet to the Raspberry Pi using a structure which didn't work. After learning that the HC-06 module doesn't accept complex types for transfer but transmits data as bytes, the mode was sent with an *int16_t* type while the GPS coordinates and motion angles were converted to *strings* before transmission.

  In the Raspberry Pi: The sent data was read using a Bluetooth socket as bytes and then converted accordingly to the appropriate type thus getting the correct values.

- ✓ **Reading GPS Signals:**

  The first problem encountered was the incapability of the GPS module to connect to satellites in order to get the location. After conducting some research, the problem seemed to be due to the attempt of reading GPS values from an indoor location. As soon as the GPS was taken outside, a connection was made to the satellites and the data was read.

  "TinyGPS++", a library specific to facilitate the interaction with the GPS module, was used in order to check for the integrity and validity of the data and to convert the raw data to an appropriate format.

- ✓ **Using Two Serial Communication Ports on the Arduino:**

  Arduino doesn't support more than one serial port to be active at any given time. That was a problem as it is required to activate the Bluetooth and GPS serial ports for communication. This was solved by programmatically specifying which port to activate at any given time.

- ✓ **Raspberry Pi Overheat:**

  After downloading and installing TensorFlow on the Raspberry Pi and the corresponding libraries from OpenCV and running a model for image processing specific to the Raspberry Pi, a 1 FPS frame rate was reached and the Raspberry Pi started overheating after moderate use *(5 – 10 min approx.)*.

  This was solved by eliminating the image processing from the Raspberry Pi and performing the heavy code on a remote PC which receives images to perform the detection and then returns its results.

- ✓ **Mode Change Using Push Button:**

  The mode change is done on the Arduino by long pressing a push button for a certain duration. After implementing this feature, the HC-06 module stopped working on each button press. This was due to the button acting as an interrupt and a noise source which disables the Bluetooth module. In order to solve this issue, the push button was connected to a separate Ground Pin and to a section of Digital Pins different than that of the HC-06's.

Language Migration (from C to Python):

Due to the limitations of C libraries on the Raspberry Pi, i.e. the lack of control of servos and the difficulty of implementation, it was essential to migrate the code from C to Python in order to facilitate and improve the work.

Live Streaming:

The Firebase storage was used for the live stream by uploading the feed to the appropriate bucket and then displaying it using its public URL.
The stream is not very fast as all media must be encrypted and then decrypted before a successful upload.

Performing Human Detection:

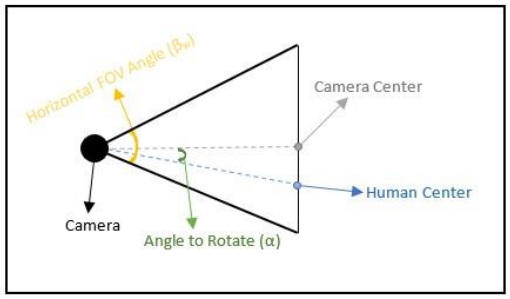
- ✓ **To setup human detection a series of steps[1] were followed:**
  1. Update the Raspberry Pi
  2. Install a pre-trained model[2], TensorFlow and OpenCV
  3. Compile and install Protobuf
  4. Setup TensorFlow directory structure
  5. Perform tests for detection. Note that the Raspberry Pi can take up to one minute to initialize and could heat up after few minutes

- ✓ **Converting coordinates to angles:**
  After successfully reading the image, it is then expanded and processed against the specified model which gives the confidence score, classes of detected objects, number of objects detected and boxes surrounding each object. The results are then filtered to keep boxes of detected humans with a confidence score greater than 60%. After filtering, the coordinates are converted to angles as follows:

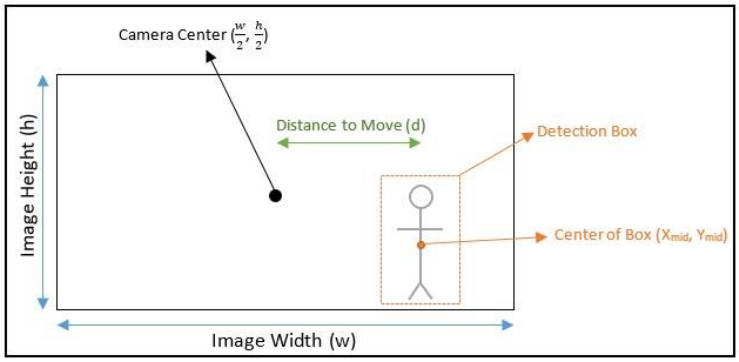
Lower left corner coordinates — Upper right corner coordinates

Boxes are arrays which contain: $[Y_{min}, X_{min}, Y_{max}, X_{max}]$

This formula demonstrates how to get the Horizontal Angle, same logic can be applied for the Vertical Angle.



Camera – Top View



Camera View

We can conclude from the following: $r = \frac{\alpha}{\beta} = \frac{x_{mid} - \frac{w}{2}}{w} => \alpha = \frac{\left(x_{mid} - \frac{w}{2}\right) \times \beta}{w}$

[1] Steps from: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi

[2] Install from: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. The model *ssd_mobilenet_v1_coco* was used in this project.

Designing the Prototype:

Due to the long shipping time of the stand, a problem was encountered regarding the prototype's design. Time was running out, and when the stand finally arrived, it was broken. The stand was then fixed as much as possible and is functioning properly if no external pressure is exerted on it.

As for the Arduino, a simple aluminum box was designed and mounted on the helmet which contains the Arduino circuit and all the components needed for the project.

Code Execution on Boot:

In order to run the Raspberry Pi code on boot for the automatic connection and activation with the Arduino and the other platforms, the following steps were executed:

1) Access the "rc.local" file on the Raspberry Pi by running the command: "sudo nano \etc\rc.local"
2) Modify the file to include a bash command that calls the script(s) to execute on boot

V. Code Snippets

➢ Arduino:

```
//get new gyro values
mpu6050.update();

x = roundAnglex((double)(mpu6050.getAngleX()));
//since values should belong to [-180, 0]
y = roundAngley((double)(mpu6050.getAngleY())) - 90;
```

Snippet 1.1 - Gyroscope Input (using custom library)

```
while (digitalRead(button_pin) == LOW) {
  //wait 100ms and increment the duration
  delay(100);
  btnPressLength = btnPressLength + 100;

  //check for changes
  if (btnPressLength >= timer_auto) {
    mode = mode_auto;
  } else if (btnPressLength >= timer_manual) {
    mode = mode_manual;
  }

  changeLedState();
}
```

Snippet 1.2 - Button Input (Change Mode)

```
//test if any data is available
while (GPSSerial.available() > 0) {
  //encode the data to the appropriate format used in the GPS library
  if (gps.encode(GPSSerial.read())) {
    //if GPS not connected to any satellites
    if (gps.satellites.value() <= 0){
      gps_valid = false;
      break;
    }
    //if there are satellites and location is valid, extract and update lat and lon
    else if (gps.location.isValid()) {
      latVal = gps.location.lat();
      lonVal = gps.location.lng();
      //specify that the data from the GPS module started being read
      gps_valid = true;
      break;
    }
  }
}
```

Snippet 1.3 - GPS Input

```
//activate the BLE serial port
activateBLE();

//send mode
BTSerial.write(mode);

//send x&y if needed
if(mode == mode_manual){
  //convert x to const char* for transmission
  //coordBuffSize >= number_of_characters_&_numbers(x) + 1(for '\0') + 1(for '-' if needed)
  char resultx[coordBuffSize];
  const char* xConv = dtostrf(x, coordBuffSize, prec, resultx);

  //send size
  BTSerial.write(strlen(xConv));
  //send x
  BTSerial.write(xConv);

}else if(mode == mode_auto){
  //wait until raspberry can accept data to send
  int sendData;
  do{
     sendData = BTSerial.read();
  }while(sendData == -1);
}
//rest of code not included
//activate the GPS serial port (thus deactivating the BLE port)
activateGPS();
```

Snippet 1.4 - Send Data via HC-06

➢ Raspberry Pi + Remote PC:

```
##Mac Address of HC-06 module in order to only connect to it
bd_addr = "00:[redacted]00:06"
portBLE = 1
def reset_ble(connected=False):
    global sockBLE

    if connected:
        sockBLE.close()

    connectedBLE = False
    ##Connect to arduino
    while(not connectedBLE):
        # Create a socket object
        sockBLE = bluetooth.BluetoothSocket( bluetooth.RFCOMM )
        ##sock.connect_ex returns 0 is connectoin is successful or the errnum in case of failure
        c = sockBLE.connect_ex((bd_addr, portBLE))
        if(c == 0 ):
            connectedBLE = True
    print(" Waiting to connect bluetooth. Status: ", c)
reset_ble()
```

Snippet 2.1 - Connecting to Arduino (via Bluetooth & HC-06 module)

```
x = ""
xVal = 0
i = 0
##Read length of x
data = sockBLE.recv(1)
##Convert xlength from bytes to int
xlen = ord(str(data,'utf-8'))

##Get the x value
##Concept:
# 1: Read every number of the x separately
# 2: Convert the bytes to strings
# 3: Add them to an accumulator variable which will then become the x when the proccess finishes
while ( i < xlen ):
    data = sockBLE.recv(1)
    xSection = str(data, 'utf-8')
    x = x + xSection
    i = i + 1
xVal = float(x.replace(" ",""))
```

Snippet 2.2 - Receiving Data from Bluetooth

```
##Give permission to variable pi to get access to the local GPIO pins
pi = pigpio.pi()

##Function to convert x-angle to an appropriate format for the motion of the servo responsible for the horizontal motion
##Rounding occurs from angle to pulse (The range is 500 to 2500)
def MoveServoXAngle(angleX):
    global oldx
    oldx=angleX

    pulseX = round(((angleX + 90) * (100 / 9)) + 500, 1)

    print(" Raw X: ",angleX)

    #precausions in case angles values are out of range
    if pulseX>2450:
        pulseX=2450

    if pulseX<550:
        pulseX=550

    ##Moving the horizontal servo by writing the converted value to the GPIO #17
    pi.set_servo_pulsewidth(pinX, pulseX)
```

Snippet 2.3 - Servo Motion

```
#get the result of detection (angles)
lenX=int(sockPC.recv(1).decode("utf-8"))
X=int(sockPC.recv(lenX).decode("utf-8"))

print("X: ", X)
X = oldx - X

MoveServoXAngle(X)
```

Snippet 2.4 - Rotation angle modification from automatic mode result

```
##Use a service account
##Use private key from file .json
cred = credentials.Certificate('helmetmotionfi[redacted].json')

##Initialize firebase_admin and get access to the storage
firebase_admin.initialize_app(cred, {
    'storageBucket': 'helmetmotion[redacted].appspot.com'
})

##Get access to the database
db = firestore.client()

##Get access to the storage
bucket = storage.bucket()
```

Snippet 2.5 - Raspberry Pi & Firebase Connection

```
##Get access to the element in firebase storage
blob=bucket.blob(imagePath)

##Uploading the image and getting the image url after
##making the blob public to use the url in a remote computer
blob.upload_from_filename(imagePath)

blob.make_public()
url = blob.public_url

#Creating thread to make stream faster
UploadImThread = Thread(target=UpdateLink, args=(url, ))
UploadImThread.start()
```

Snippet 2.6 - Image Upload to Firebase Storage (Live Feed)

```
##Update the URL in the db to chnage the image in the web
##Referencing the URL document for the update ('u' is to transform the string to frpm bytes to unicode)
doc_url = db.collection(u'Stream').document(u'feed')

##Updating the field
doc_url.set({
    u'url':url,
})
```

Snippet 2.7 - Updating Public URL Value in Firebase

```
# Load the Tensorflow model into memory and start session
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)

# Get index of box where human probability is the highest and greater than 60%
# Class of humans = 1
def bScoreOnlyHumans(classes, scores):
  j=-1
  max=-1
  for i in range(0,99):
    if classes[i] == 1 and scores[i]>=0.6:
      if scores[i] > max:
        max = scores[i]
        j=i
  return j


# Each box represents a part of the image where a particular object was detected
def getX(box):
    x = (box[1] + box[3])/2 * width

    x = ((x-width/2) / width) * 40

    return int(np.int16(x.item()))
```

```
image = cv2.imread(PATH_TO_IMAGE)
height, width, channels = image.shape

image_expanded = np.expand_dims(image, axis=0)

# Perform the actual detection by running the model with the image as input
(boxes, scores, classes, num) = sess.run(
    [detection_boxes, detection_scores, detection_classes, num_detections],
    feed_dict={image_tensor: image_expanded})

maxIndex=bScoreOnlyHumans( np.squeeze(classes), np.squeeze(scores))

boxesqueezed=np.squeeze(boxes)
if maxIndex==-1:
    x=0
    y=0
    print('no human detected')
else:
    print('human detected')
    #calculates coordinates for the center of the "box" surrounding the human
    x=getX(boxesqueezed[maxIndex])
    y=getY(boxesqueezed[maxIndex])

#Send length X
c.send(str(len(str(x))).encode("utf-8"))
#Send X
#print(str(x))
c.send(str(x).encode("utf-8"))
```

Snippet 2.8 - Human Detection in Remote PC

- **Web Panel:**

```
firebase: {
  apiKey: 'AIza                                    EM',
  authDomain: 'helmetmot                           .firebaseapp.com',
  databaseURL: 'https://helmetmotionf              .firebaseio.com',
  projectId: 'helmetmo                             3719',
  storageBucket: 'helmetmotion                     .appspot.com',
  messagingSenderId: '81        809'
}
```

Snippet 3.1 - Firebase Authentication Criteria

```
imports: [
  AngularFireModule.initializeApp(environment.firebase),
  AngularFirestoreModule,
  AngularFireStorageModule
],
```

Snippet 3.2 - Connection to Firebase Database and Importing Firebase Angular SDKs (angularfire2)

```
imports: [
  AgmCoreModule.forRoot({
    apiKey: 'AIzaSy                                  CDng'
  })
],
```

Snippet 3.3 – Specifying Google Maps API Key

```
//initialize GPS variables
private initGPS():void{

      //reference the collection
      this.GPSRef = this.dbfs.collection(this.colGPS);

      //reference the document
      this.coordDoc = this.GPSRef.doc<GPSCoordinates>(this.docGPS);

      //get the values and listen for changes
      this.coords = this.coordDoc.valueChanges();
}
```

Snippet 3.4 - Access GPS Coordinates and Listen to Changes

```
//get lat and lon of the soldier
this.fbs.getCoords().subscribe((coordsObs) => {
  this.coords = coordsObs;

  this.lat = this.coords.lat;
  this.lon = this.coords.lon;
});
```

Snippet 3.5 - Get GPS Coordinates from Firebase

```
<agm-map
  [latitude]="lat"
  [longitude]="lon"
  [streetViewControl]="disableOpt"
  [rotateControl]="enableOpt"
  [mapTypeId]="enableSatelliteView"
  [zoom]="zoomLevel">

</agm-map>
```

Snippet 3.6 – Customize and Display Google Maps Map