Final Project Report - PerceptoBot

ECE 558-Fall 2018
Instructor- Prof. Roy Kravitz

Shubham Vasaikar, Mrunal Hirve
12/7/2018

Demo: https://youtu.be/kpluHYzKbks

Repository: https://github.com/ECE558-fall2018/FinalProj-shubhamv_mrunalh

Contents

Description 2

Design 2

Hardware 3

- Architecture 3
- Components 3
- Hardware Connections 3
- Implementation Details 4
  - Reading Sensor data 4
  - Capturing Video Stream 4
  - Using the camera for 2 tasks 4
  - Face Detection 5
- PerceptoBot 6

Software 7

- Firebase Design 7
- Android Application 8
  - Application Layout 8
  - Application Implementation Details 9

Stretch Goals 11

Challenges Faced 11

- Weight Imbalance of Robot 11
- Python-Firebase Interface 11
- Open CV installation 11

New Learning outcomes 11

- Hardware 11
- Software 11

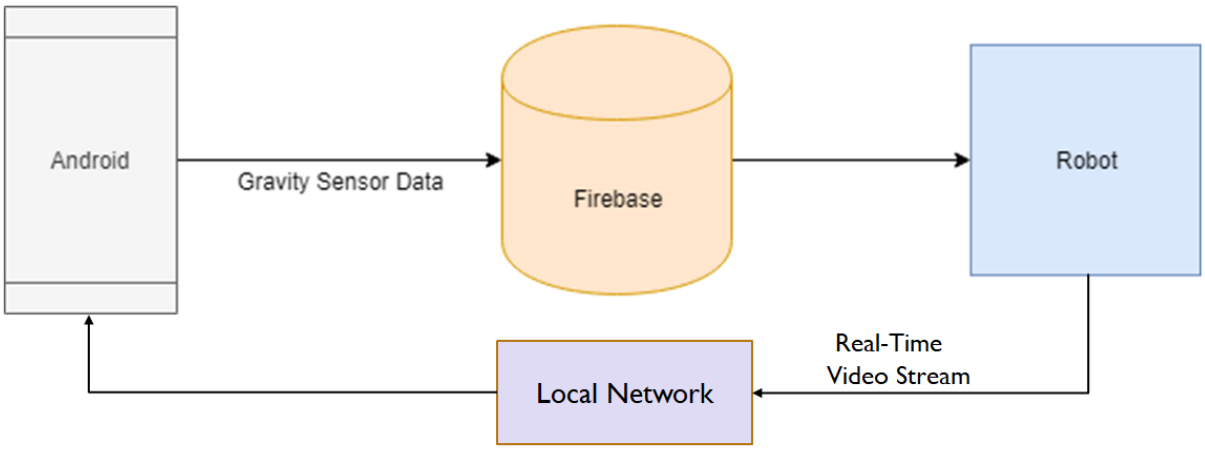
Description

The name PerceptoBot is derived from Perception: meaning ability to see or hear. PerceptoBot currently has the ability to detect and follow human faces.

We have designed a 2-wheeled robot that is controlled via an Android app using movement sensors (accelerometer and gyroscope) on the phone. The camera also sends a stream of live video to the android application.

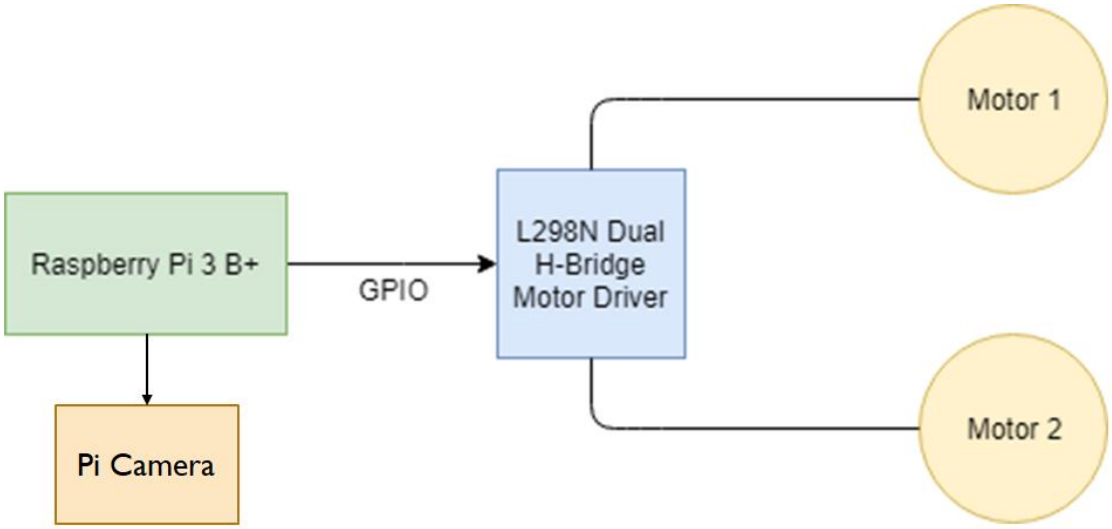
Design

As a high-level overview of the current project, the app continually monitors and sends sensor values to the Firebase Realtime Database. The robot is listening on the sensor attributes in firebase and makes movements accordingly. The Pi camera mounted on PerceptoBot sends a live video stream over the local network to the application.



Hardware

Architecture



Components

- Robot chassis to mount a Raspberry Pi 3 B+.
- 2 wheels
- A motor driver
- 2 DC Motors
- 2 battery packs (one for the RPi and one to power the motors)
- Pi Camera
- The RPi runs Rasbian OS and connects to the motor driver using the GPIO pins.

Hardware Connections

RPI3B+	**L298N**		**L298N**	**DC MOTOR**
GPIO 7	IN 1		OUT 1	Vcc
GPIO 8	IN 2		OUT2	GND
GPIO 9	IN 3		OUT3	Vcc
GPIO 10	IN 4		OUT4	GND
GND	GND			

Implementation Details

Reading Sensor data

The code that reads the sensor values and runs the motors is written in Python and uses the pyrebase module to listen for changes in firebase data. We are using the inbuilt gpiozero library to control the robot.

```
from gpiozero import Robot
import pyrebase

firebase = pyrebase.initialize_app(config)
db = firebase.database()

def linear_handler(message):
    if (db.child("direction").get().val()):
        robot.forward(message['data'])
    else:
        robot.backward(message['data'])

linear_stream = db.child("linearAcc").stream(linear_handler)
```

Capturing Video Stream

We tried to capture video streams using Picamera along with OpenCV over the RTSP protocol. However, it was very unreliable and consumed a lot of resources on the RPi while also being slow. Therefore, we decided to use a pre-baked solution to stream video over the network. We settled on “pistreaming” by waveform80 (the creator of the picamera python module). This solution allows us to have low latency streams over http rather than rtsp. There are 2 advantages to this – the stream can be viewed on any browser, and the stream has very low latency while maintaining high quality. We are displaying the stream in the app using a WebView.

Using the camera for 2 tasks

After video streaming, face detection was a little tricky as the camera was now already being used for video streaming. So, to use the camera for face detection at the same time, we wrote a thread that will accept the same camera object that is being used by the video stream. This thread will then run in parallel to the video stream, while both of them can access the camera capture.

```
print('Initializing broadcast thread')
output = BroadcastOutput(camera)  # Video stream
broadcast_thread = BroadcastThread(output.converter, websocket_server)
print("Initialize video object")
video = Video(camera, cascPath="./cascade.xml") # Face detection
```

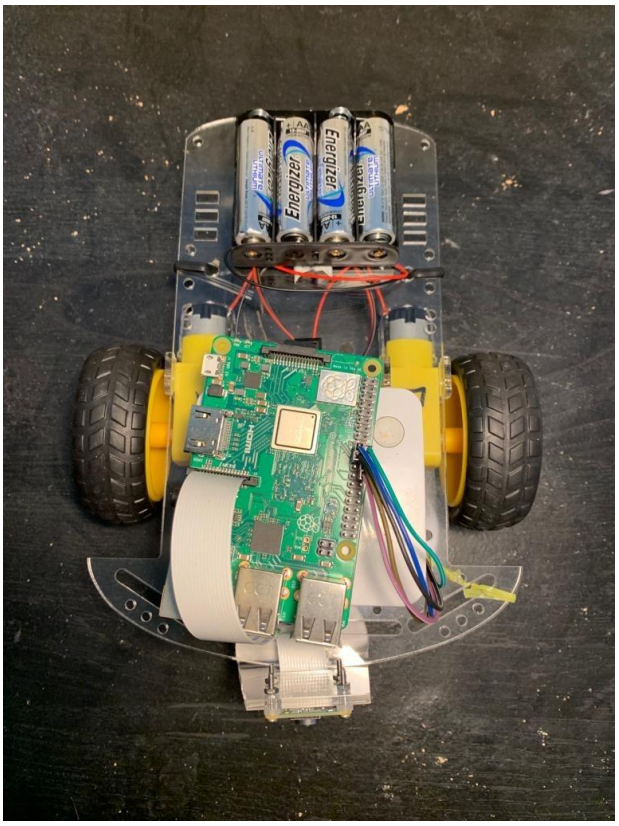
In this code snippet, the camera object is being used to initialize both, the Video Stream thread, and the Face Detection thread.

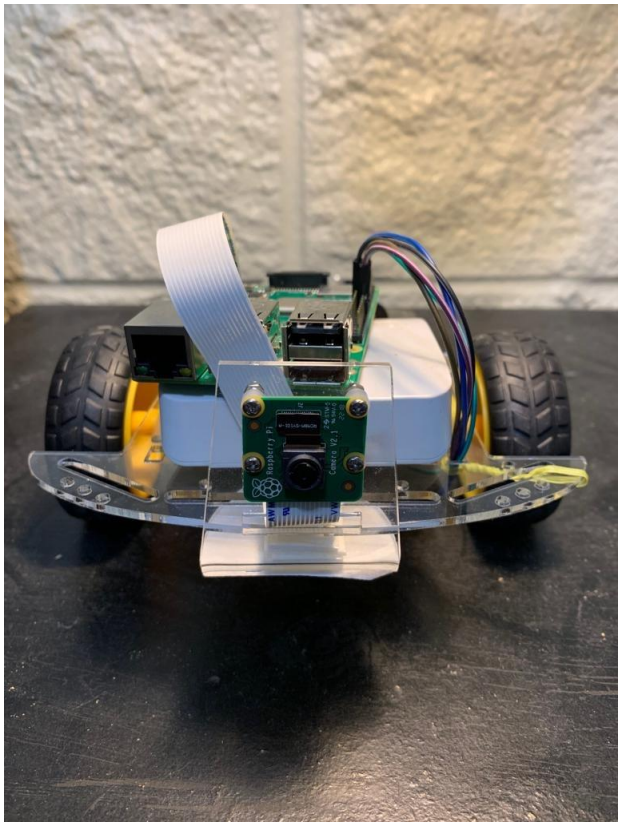
Face Detection

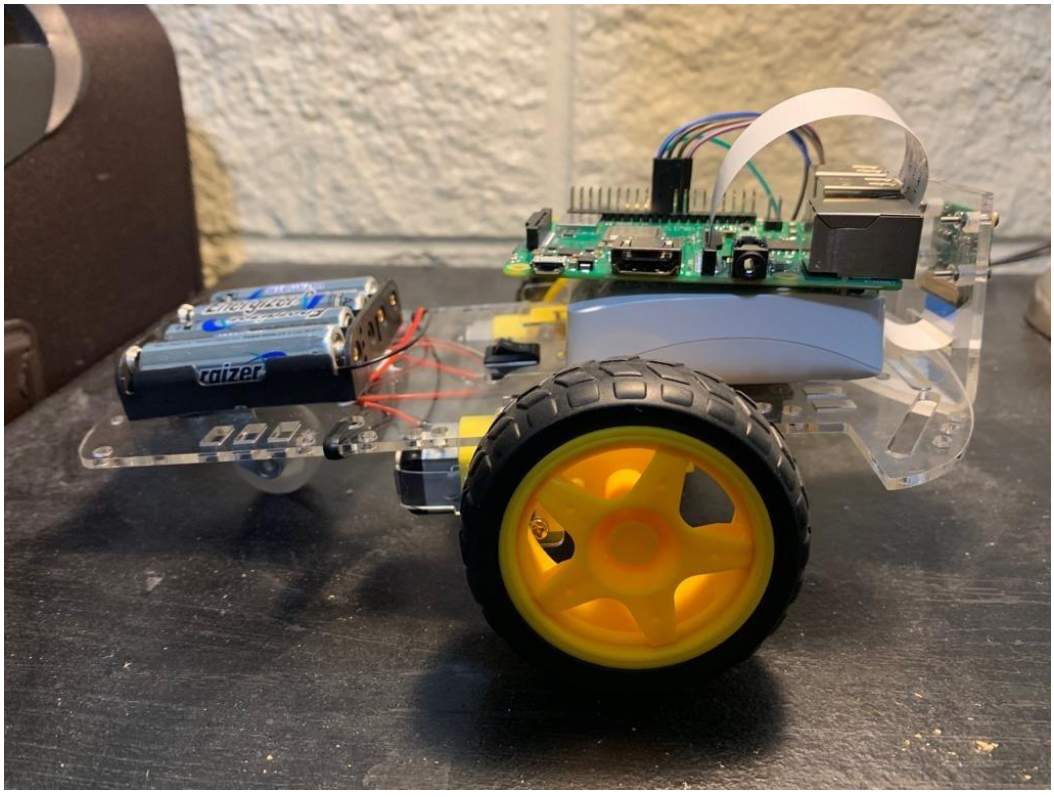
We used OpenCV to perform face detection. Face detection is performed via a previously trained classifier that looks for faces in an image. We extract frames from the camera, then perform some pre-processing on it before passing it on to OpenCV's Cascade Classifier. This Cascade Classifier needs to be given an XML file as input. In this case, we used the HAAR frontal face classifier. When a frame from the camera is passed to the classifier, it returns the coordinates of the position of the face in the frame. We can then use these coordinated to check if the user is moving out of frame. If the user approaches the edge of the frame on either side, the robot will turn so that the face is in the frame again.

```
faceCascade = cv2.CascadeClassifier(self.cascPath) # The classifier

...

faces = faceCascade.detectMultiScale( # faces contains an array of
    frame,                            # coordinates of the detected
    scaleFactor=1.1,                  # face
    minNeighbors=5,
    minSize=(30, 30),
    flags=cv2.CASCADE_SCALE_IMAGE
)
```

PerceptoBot

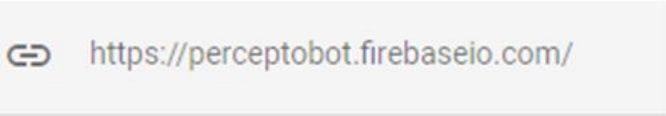


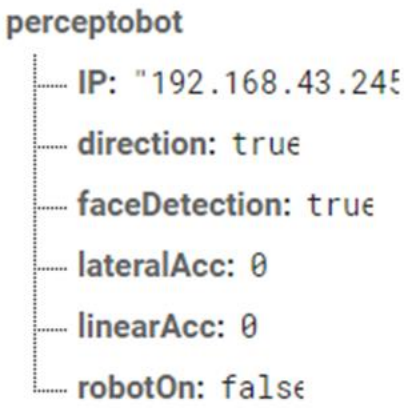




Software

Firebase Design





We are using Firebase as a Backend-as-a-Service to store data on cloud. Firebase sends and transmits real-time data over the network from the application to the RPi and vice versa.

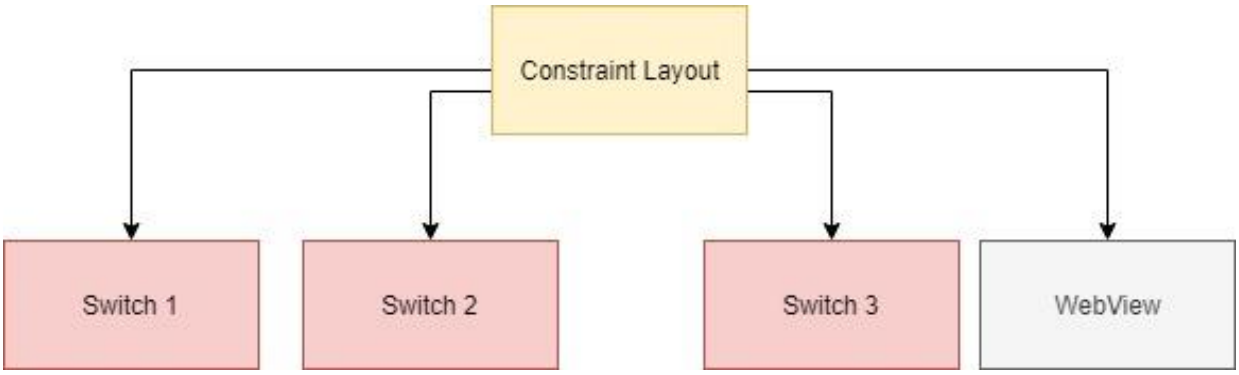
Firebase DB includes these fields:

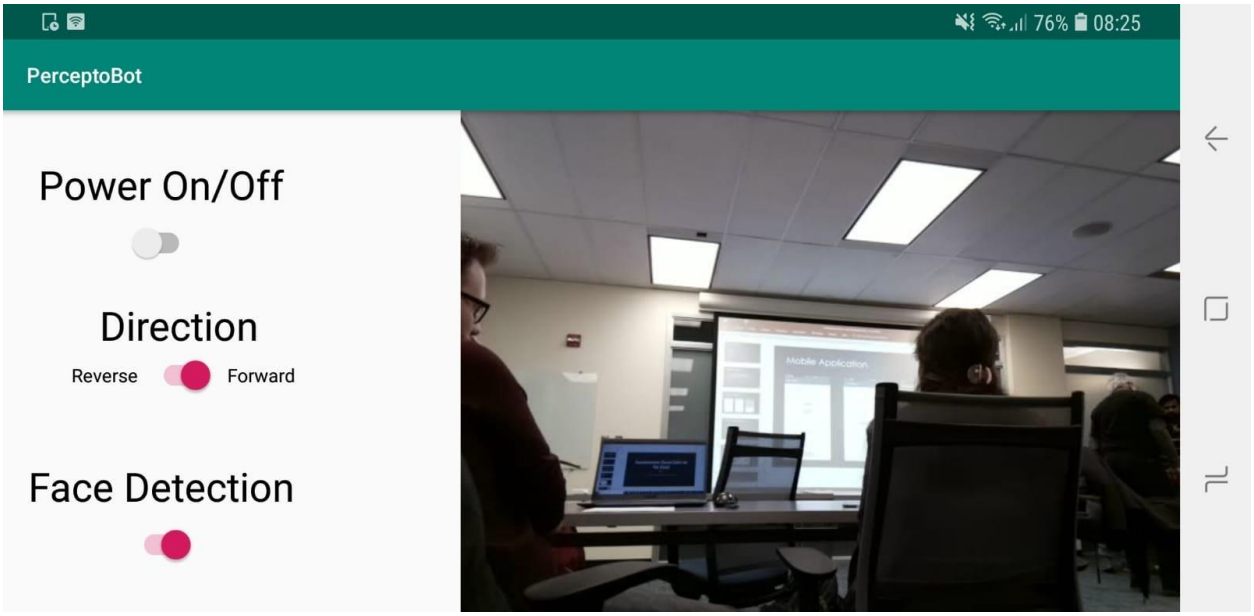
- IP: The RPi sends its local wlan0 IP to Firebase so that the app can open to the webview to that address.
- robotOn: Boolean flag, when set to true allows the robot to move in linear and lateral directions.
- direction: Boolean flag. When set to true robot moves in the forward direction and when set to false robot moves in reverse direction
- lateralAcc: Stores the converted x axis values sent from the android application to set the speed of the bot in either forward and reverse direction.
- lateralAcc: Stores the y axis values from the android application for the robot to move in either left or right direction
- faceDetection: Boolean flag, when set to true allows only the Face detection feature of the robot to work. The bot then moves in direction of the face.

Note: To avoid hindrance we have set only one feature on at a given instance of time. If the user sets faceDetection on when the application already has robotOn set, robotOn will be disabled and vice versa.

Android Application

Application Layout





It is a single android application developed using Constraint Layout. The application opens as a landscape view, which is set as the default view.

Application Implementation Details

Creating Firebase Database Reference

For sending data across the firebase we have created a firebase database reference

```
database =  FirebaseDatabase.getInstance();
myRef = database.getReference();
```

Getting Sensor data

To get the sensor data, the app needs to create an instance of sensor manager which allows to interface with the sensors.

```
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

To listen for changes in sensor values, the activity also needs to implement the sensor event listener interface. This interface provides a callback – onSensorChanged() that we can implement.

```
public class MainActivity extends AppCompatActivity implements
SensorEventListener {

...

@Override
public void onSensorChanged(SensorEvent event) {
    float x, y;
    x = event.values[0];

    if ((x > 0 && x < 9) && (mOn)) {
        x = 1-(x/9);
    } else {
        x = 0;
    }

    y = event.values[1];
    if (((y < -1 && y > -6 ) || (y > 1 && y < 6 )) && (mOn)) {
        y = y;
        x = 0;
    } else {
        y = 0;
    }

    myRef.child("linearAcc").setValue(x);
    myRef.child("lateralAcc").setValue(y);
}
```

Using WebView for video stream.

We are using android webView for streaming video on the application.

```
mWebview = (WebView) findViewById(R.id.webview);
WebSettings webSettings = mWebview.getSettings();
webSettings.setJavaScriptEnabled(true);
webSettings.setBuiltInZoomControls(true);

mWebview.setWebViewClient(new WebViewClient() {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request) {
        return false;
    }
});
```

Restoring Application state

The application resumes from where it had stopped last time by calling a SingleValueEventListner which reads the values from the firebase last updated.

```
myRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        mWebview.loadUrl("http://" +
dataSnapshot.child("IP").getValue(String.class) + ":8082");
        mReverse.setChecked((Boolean)
dataSnapshot.child("direction").getValue());
        mFaceDetection.setChecked((Boolean)
dataSnapshot.child("faceDetection").getValue());
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

Switching off the Robot on closing the app

Restoring the last application status caused robot to start directly, to prevent this we explicitly override the robotOn to Off every time on closing the application.

```
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
    myRef.child("robotOn").setValue(false);
}
```

Stretch Goals

We were successful in achieving both our stretch goals

☑ Streaming video from Pi Camera mounted on robot to the Android app so that the user can navigate without having a line-of-sight to the robot.

☑ Using the camera to detect faces and follow the movement of the faces.

Challenges Faced

Weight Imbalance of Robot

Earlier, we had attached both the battery packs over the rear wheel of the robot. This meant that there was too much weight over the rear and the front was very light. Due to this the front wheels could not gain enough traction to propel the robot in a straight line and the robot veered off into whichever direction the rear wheel was facing.

Python-Firebase Interface

Initially, we were planning on using the python-firebase python module to connect to firebase. However, we soon realized that this module did not provide methods to register listeners for firebase data values. Therefore, we decided to move on to pyrebase which provides stream listeners for firebase data values.

Open CV installation

Since, there are no pre-compiled packages available for RPi, we have to compile OpenCV on our own. The compilation alone takes ~2 hours to run. We tried compiling 2 times using 4 threads for the compilation. But we were running into resource contention issues on a step of the compilation. So, we had to dial down our compilation to just one thread which took even longer to complete. A whole day was just spent trying to compile OpenCV.

New Learning outcomes

Hardware

Motor drivers and H-bridge
Pi camera

Software

Interfacing with sensor values
OpenCV
WebViews
ConstraintLayout

References

https://github.com/waveform80/pistreaming
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml
https://realpython.com/face-detection-in-python-using-a-webcam/
https://github.com/thisbejim/Pyrebase#database
https://developer.android.com/guide/topics/sensors/sensors_motion#sensors-motion-grav
https://developer.android.com/reference/android/hardware/SensorListener
https://gpiozero.readthedocs.io/en/stable/recipes.html#robot