ECE 544 Final Project Report

Android & Gesture Controlled Car

Ram Bhattarai

Ming Ma

Zhe Lu

06/11/2019

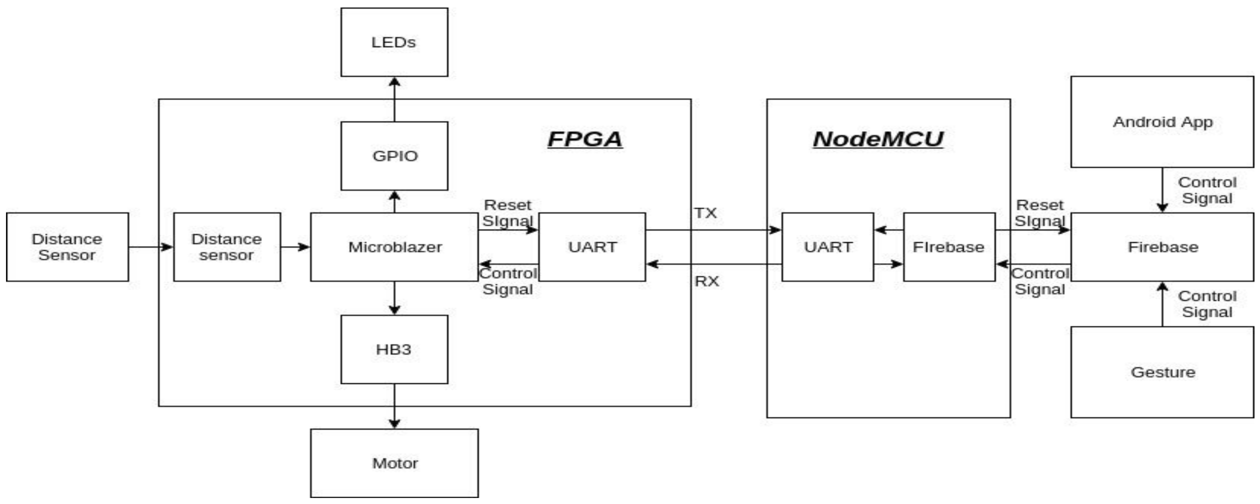
Project Description:

For this project, we have designed an android controlled car. This car has 2 motors which are controlled by two PmodHB3 connected to Nexys A7 board. The control signals are given from two sources: Android App or Gestures. These control signals will be transferred to Nexys A7 board by using Firebase and NodeMCU32. The communications between NodeMCU32 and Nexys A7 are achieved using UART. Also, we have two distance sensors connected to Nexys A7 board. The car can't move forward or backward if there is an obstacle within a range of 15 cm. Four LEDs are also connected to the FPGA. So, our car will turn on different LEDs accordingly based on different motions, just like the real car.

List of Hardware:

1) Nexsys A7
2) Two Wheels
3) Two Motors
4) Two PmodHB3
5) NodeMCU Esp32
6) 4 LEds
7) Headphone
8) 2 Distance Sensor
9) Android Phone

Hardware Description:

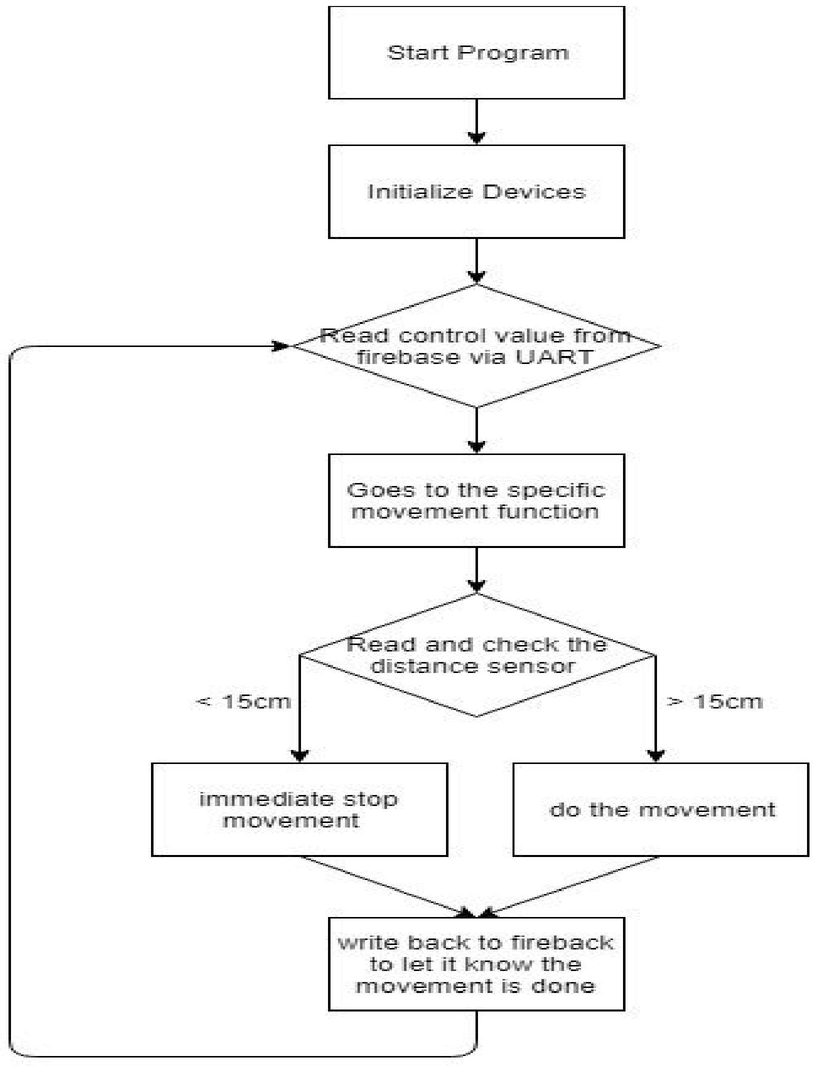


As shown in the figure above, this is the flowchart of our hardware part. The control signals will be given from Android App or Gestures. The control signals will be sent to Firebase and received by NodeMCU32 Wifi module. Then, these control signals will be sent to FPGA by using UART. The Microblaze will execute these control signals and control the LEDs and motors according to the following table.

Signal	MotorL	MotorR	LEDFL	LEDFR	LEDBL	LEDBR
Forward	counter-clockwise	counter-clockwise	off	off	off	off
Back	clockwise	clockwise	off	off	on	on
Left	off	counter-clockwise	on	off	on	off
Right	counter-clockwise	off	off	on	off	on

Also, the microblaze will check the distance between our car and possible obstacle before executing forward and back operation. If the distance is smaller than 15 cm, ignore forward and back signals and keep stop. Otherwise, our car will go forward or backward according to the control signals.

Software Description:


Start Program
Initialize Devices
Read control value from firebase via UART
Goes to the specific movement function
Read and check the distance sensor
< 15cm
> 15cm
immediate stop movement
do the movement
write back to fireback to let it know the movement is done

This program is the car control system based on the standalone. First it initializes each device that we use, such as UART, GPIO drivers.Then reads the control signal via UART communication with NODEMCU which is a WIFI module for reading data from firebase. Then based on these control signals, it will control the movement of the car by given the motor different rotation direction.

Also, we have a distance function that we can read the distance value. we check it every movement function, if the distance is less than 15 centimeters, stop the car immediately, otherwise just do the movement based on the control signal.

Hardware Implementation & Testing:

Power:

For this project, we use two power sources. One is power bank and the other is battery bank. The FPGA and NodeMCU32 are powered by using power bank. Others are powered by using battery bank.

Motor Part:

For this project, we use two cheap motors which are included in the car model package. Two motors are controlled by PmodHB3, like we did in project 2.However, we don't need SA for this project and we only want to control the rotation direction of each motor. One PmodHB3 is connected to JA[0]-JA[2] and the other is connected to JD[0]-JD[2]. We use the hardware of project 2 to test these two motors and we find it works perfectly. Then, we add these motors to our final project.

Distance Sensor:

We use two ultrasonic distance sensors in this project. One is in front of the car and the other is in the back of the car. In order to make this kind of distance sensor work, we need to give trig port high signal for at least 10us. Then, this sensor will continuously send 8 burst of signals. These signals will be reflected when they touch some object and the reflected signals will be received by echo port(echo port becomes 1). Then, we count how long echo port is high and pass this count values to SDK by reading base address of each distance sensor. The front distance sensor is connected to JC[2] and JC[3] and back distance sensor is connected to JC[4] and JC[5]. The basic formula is:
Distance(cm) = (count value * 340 * 100) / ((10^7) * 2). 10Mhz = 10^7.

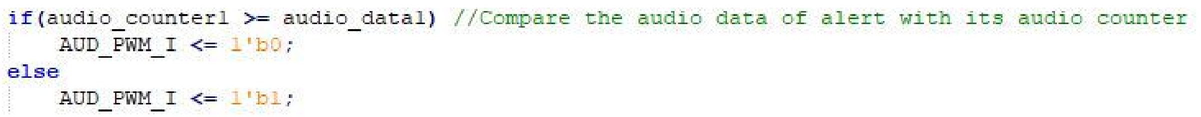
We create a new project to test our distance sensor IP. The count time is exported to SDK. Then, we print calculated distance in SDK terminal to see if this is correct or not.

LED:

We have four LEDs in this design. The LEDs will be turned on according to different control signal. More details are included in the table of Hardware Description section. Each LED is controlled by one of the GPIO ports. There are four GPIO ports and each is declared as 8 bit output. Four LEDs are connected to JB[0]-JB[3]. This is pretty simple. So, we just add this part to our design and it works perfectly.

Sound:

Our car will play alert if there is an obstacle in range of 15 cm(front and back). We use the amplifier in Nexus A7 board to play this sound. We find an alert online which is in the format of wav file and then we convert this file to ceo file by using Matlab. This coe file will be stored in BRAM and read out to play alert. In order to play sound, we need to configure two ports: AUD_PWM and AUD_SD. AUD_PWM is used to decide to play sound or not. The basic idea is shown below:

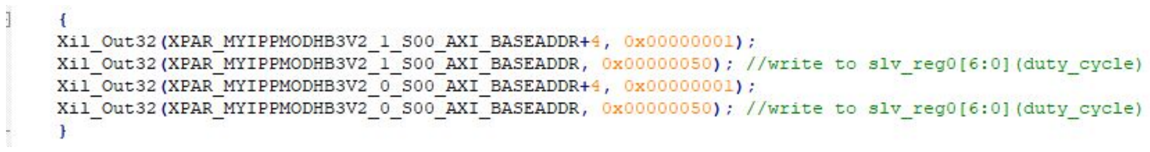


```
if(audio_counter1 >= audio_data1) //Compare the audio data of alert with its audio counter
    AUD_PWM_I <= 1'b0;
else
    AUD_PWM_I <= 1'b1;
```

There is a counter which is incremented every rising edge of clock. Comparing this counter with audio data read from BRAM. AUD_PWM is set to 0 if the value of counter is greater than or equal to audio data. Otherwise, AUD_PWM is set to 1. The AUD_SD is shutdown mode and we just assign this port to constant value 1. We use 16k clock to play the alert. Also, we don't have the speaker which can connect to the FPGA directly. So, we use earphones to play the alert and the sound is loud enough to hear. Basically, we also have another project to test this sound module. We add this sound to our project after we tested this successfully.

Software Implementation and Testing:

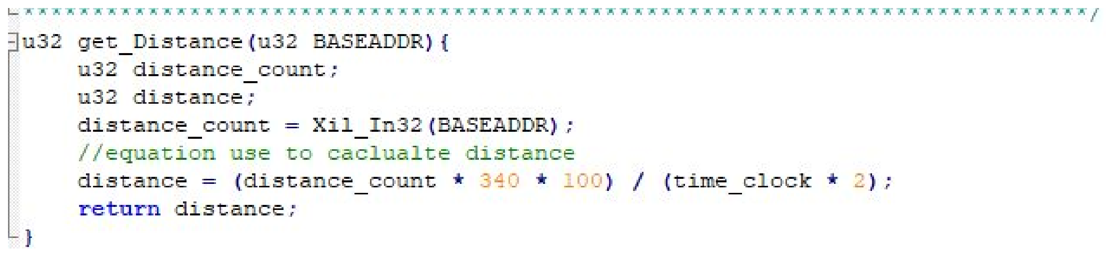
Motor Control:



```
{
Xil_Out32(XPAR_MYIPPMODHB3V2_1_S00_AXI_BASEADDR+4, 0x00000001);
Xil_Out32(XPAR_MYIPPMODHB3V2_1_S00_AXI_BASEADDR, 0x00000050); //write to slv_reg0[6:0](duty_cycle)
Xil_Out32(XPAR_MYIPPMODHB3V2_0_S00_AXI_BASEADDR+4, 0x00000001);
Xil_Out32(XPAR_MYIPPMODHB3V2_0_S00_AXI_BASEADDR, 0x00000050); //write to slv_reg0[6:0](duty_cycle)
}
```

The above code is the forward movement, we just set up the direction for both wheels to forward and give a constant pwm value. Other movements are similar thing that we just give different direction of the motor. We tried each movement function for the car to see the behavior.

Distance Sensor Control:



```
***************************************************************************/
u32 get_Distance(u32 BASEADDR){
    u32 distance_count;
    u32 distance;
    distance_count = Xil_In32(BASEADDR);
    //equation use to caclualte distance
    distance = (distance_count * 340 * 100) / (time_clock * 2);
    return distance;
}
```

Here is the getting distance function for our software part, because we have both front and back distance sensor so that the input parameter is the baseaddress that the distance sensor has, and the output is the distance that gives from that distance sensor. First we can get the distance count which reads from hardware and use the equation to calculate the distance.

LED Control:

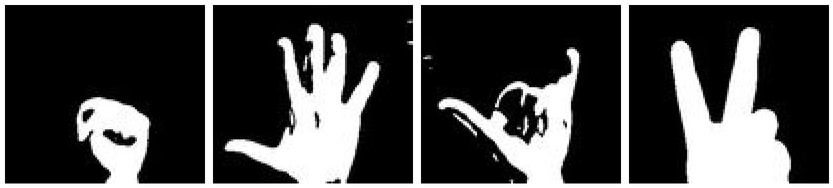
We implemented two set GPIO modules which have a total of 4 channels GPIO port to transmit back the signal for LEDS from software to hardware. We tested it by just flashing all leds.

Sound Control:

We implemented one GPIO to the sound control signal as the enable signal to the hardware module. We created a new embedded system which only have sound module and GPIO module to test it can be enabled or not.

Gesture Recognition:

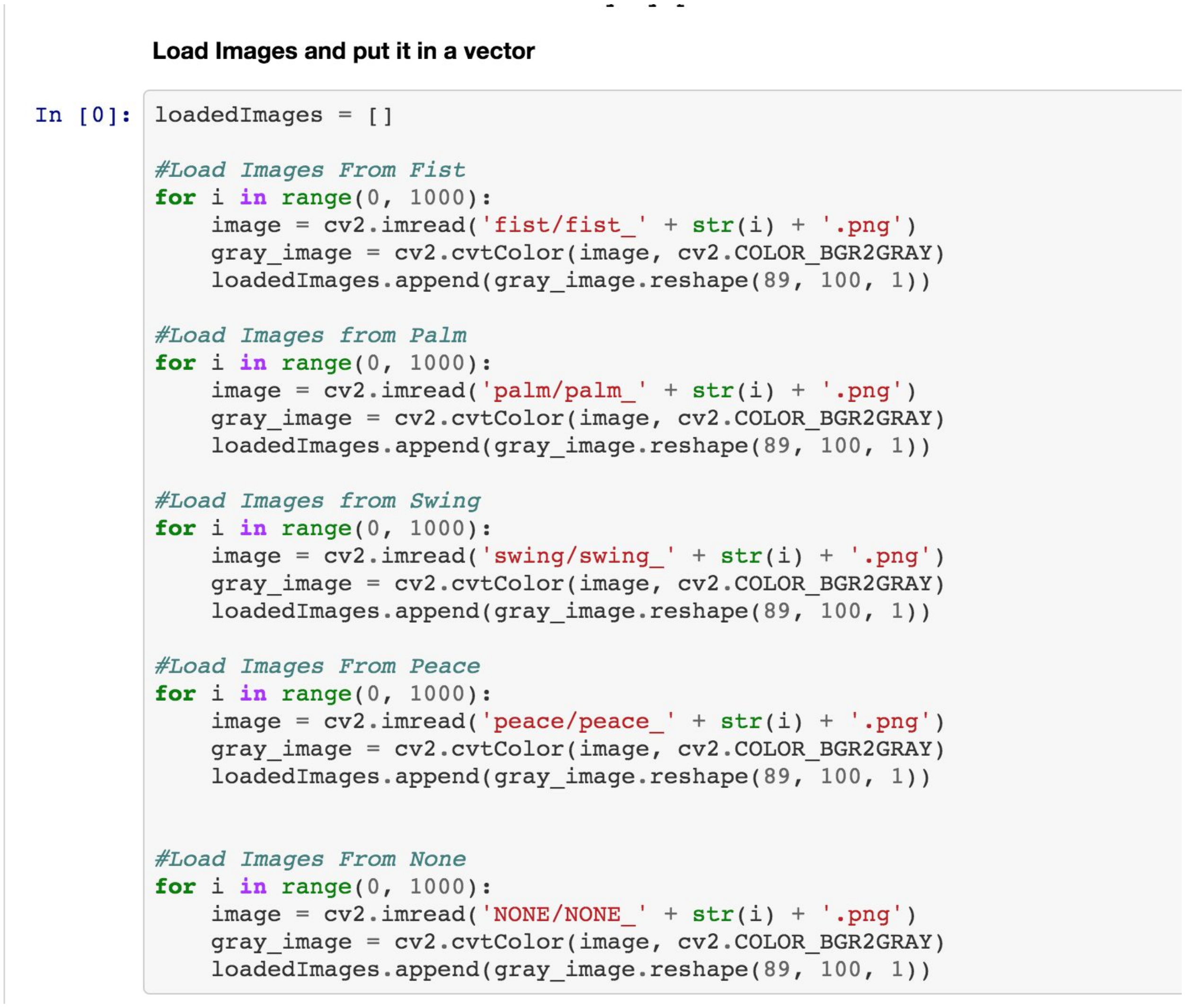
We have 4 gestures(Fist, Palm, Swing, Peace) to control the car movement. Each of the gestures is encoded to 1(FWD), 2(Reverse), 3(Left) ,4(Right) respectively. Four gesture recognition is achieved by training all of them using TensorFlow. A brief list with description of how gesture recognition is achieved.



1) Image Generation

For our training set and testing set, we have generated 1000 images and 100 images respectively. Taking pictures is very tedious so using an automated script is the best way to generate image. Image is generated using a python script tracking.py(included in the github). Image is generated with the same background.

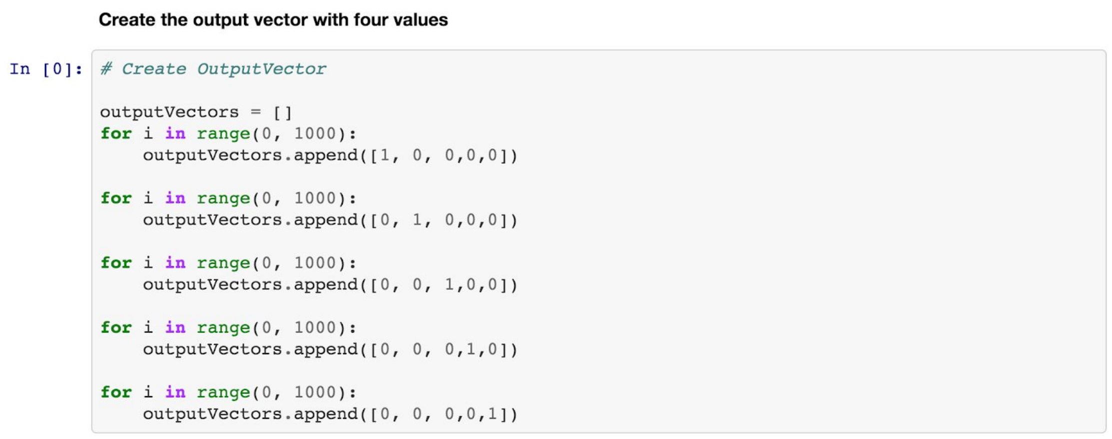
2) Image Resizing
   Training image with larger images will be time consuming so images need to be resized. The bigger the image, the complicated the model will be. Image is resized to 89 x 100 size. Since we have lots of image to resize, python script is used to resize the image.
3) Loading the Training images
   Now, we have our image ready for the training. Next step in the process is storing all the images into a vector. I am attaching a screenshot of how this is achieved.



Load Images and put it in a vector

```
In [0]: loadedImages = []

        #Load Images From Fist
        for i in range(0, 1000):
            image = cv2.imread('fist/fist_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            loadedImages.append(gray_image.reshape(89, 100, 1))

        #Load Images from Palm
        for i in range(0, 1000):
            image = cv2.imread('palm/palm_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            loadedImages.append(gray_image.reshape(89, 100, 1))

        #Load Images from Swing
        for i in range(0, 1000):
            image = cv2.imread('swing/swing_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            loadedImages.append(gray_image.reshape(89, 100, 1))

        #Load Images From Peace
        for i in range(0, 1000):
            image = cv2.imread('peace/peace_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            loadedImages.append(gray_image.reshape(89, 100, 1))


        #Load Images From None
        for i in range(0, 1000):
            image = cv2.imread('NONE/NONE_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            loadedImages.append(gray_image.reshape(89, 100, 1))
```

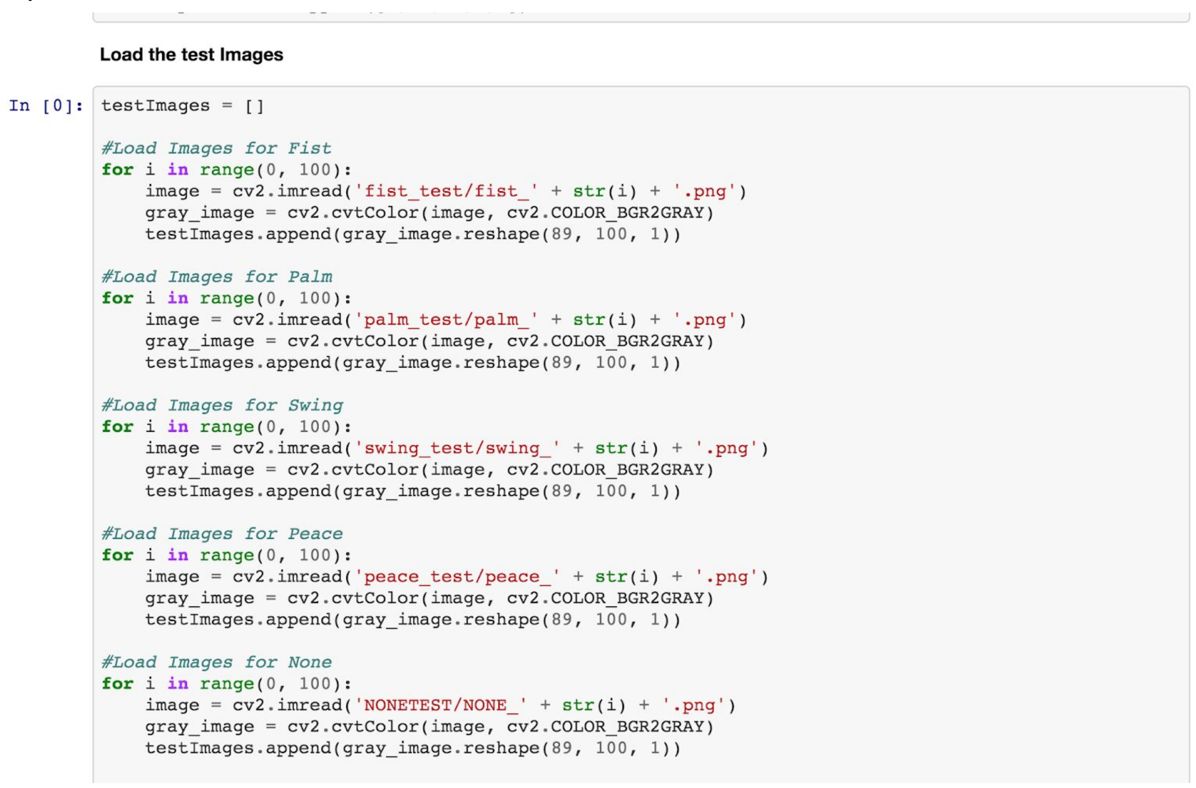
4) Labeling the Training Images
   Labeling is a very important step in the training process. Since we have 4 gestures and 1 gesture with no image on the background. We need to encode them to a vector with either a 0 or 1. Image is already pixelated to a 0 or 1 in the previous step.



Create the output vector with four values

```
In [0]: # Create OutputVector

        outputVectors = []
        for i in range(0, 1000):
            outputVectors.append([1, 0, 0,0,0])

        for i in range(0, 1000):
            outputVectors.append([0, 1, 0,0,0])

        for i in range(0, 1000):
            outputVectors.append([0, 0, 1,0,0])

        for i in range(0, 1000):
            outputVectors.append([0, 0, 0,1,0])

        for i in range(0, 1000):
            outputVectors.append([0, 0, 0,0,1])
```

5) Loading the Testing image
   Our training vector is setup, now we need to setup the testing vector. The model needs to also have a taste of the testing images. It is achieved with following steps:

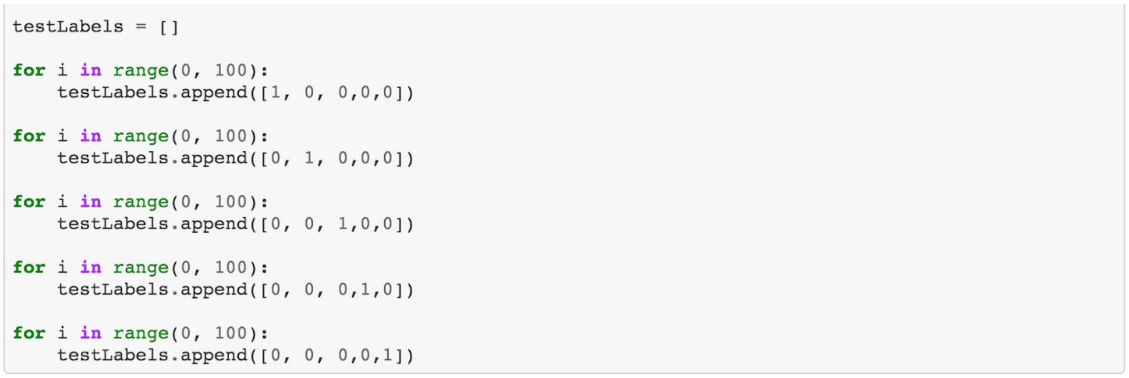


Load the test Images

```
In [0]: testImages = []

        #Load Images for Fist
        for i in range(0, 100):
            image = cv2.imread('fist_test/fist_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            testImages.append(gray_image.reshape(89, 100, 1))

        #Load Images for Palm
        for i in range(0, 100):
            image = cv2.imread('palm_test/palm_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            testImages.append(gray_image.reshape(89, 100, 1))

        #Load Images for Swing
        for i in range(0, 100):
            image = cv2.imread('swing_test/swing_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            testImages.append(gray_image.reshape(89, 100, 1))

        #Load Images for Peace
        for i in range(0, 100):
            image = cv2.imread('peace_test/peace_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            testImages.append(gray_image.reshape(89, 100, 1))

        #Load Images for None
        for i in range(0, 100):
            image = cv2.imread('NONETEST/NONE_' + str(i) + '.png')
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            testImages.append(gray_image.reshape(89, 100, 1))
```

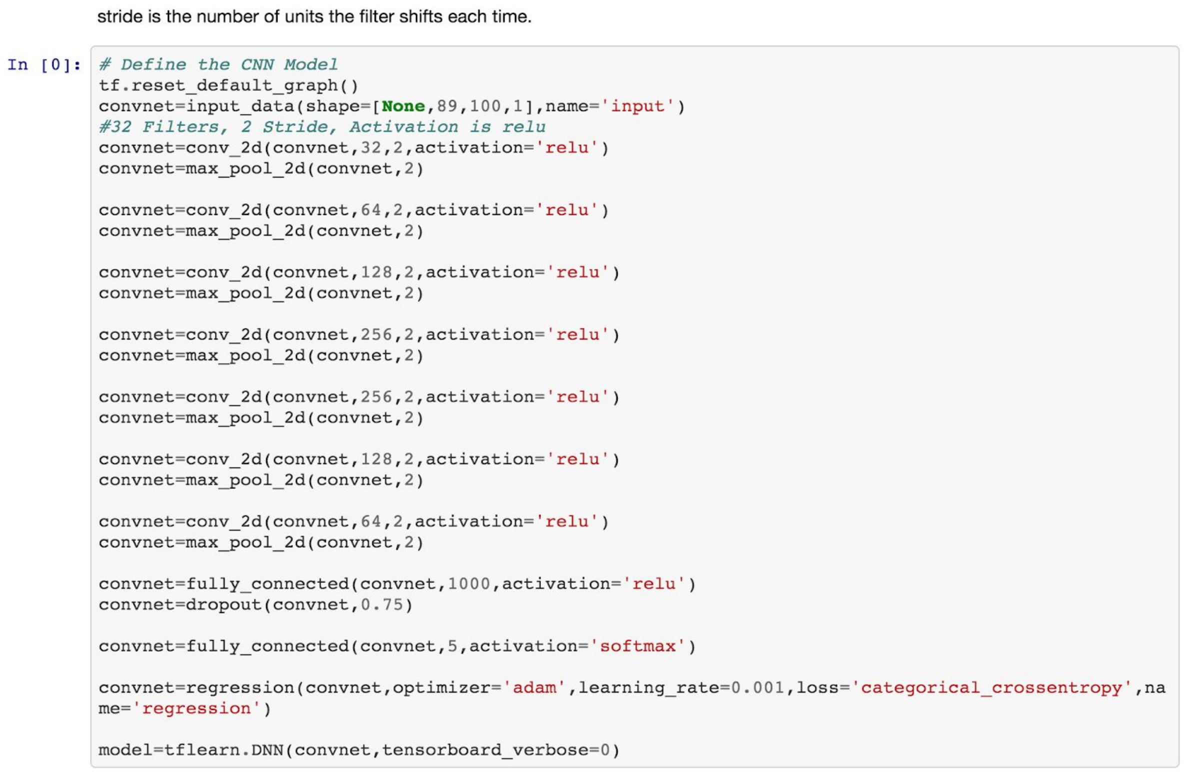
6) Labeling the Testing Image

Testing vector is created to label all the training images. It is achieved in the following manner.



```
testLabels = []

for i in range(0, 100):
    testLabels.append([1, 0, 0,0,0])

for i in range(0, 100):
    testLabels.append([0, 1, 0,0,0])

for i in range(0, 100):
    testLabels.append([0, 0, 1,0,0])

for i in range(0, 100):
    testLabels.append([0, 0, 0,1,0])

for i in range(0, 100):
    testLabels.append([0, 0, 0,0,1])
```

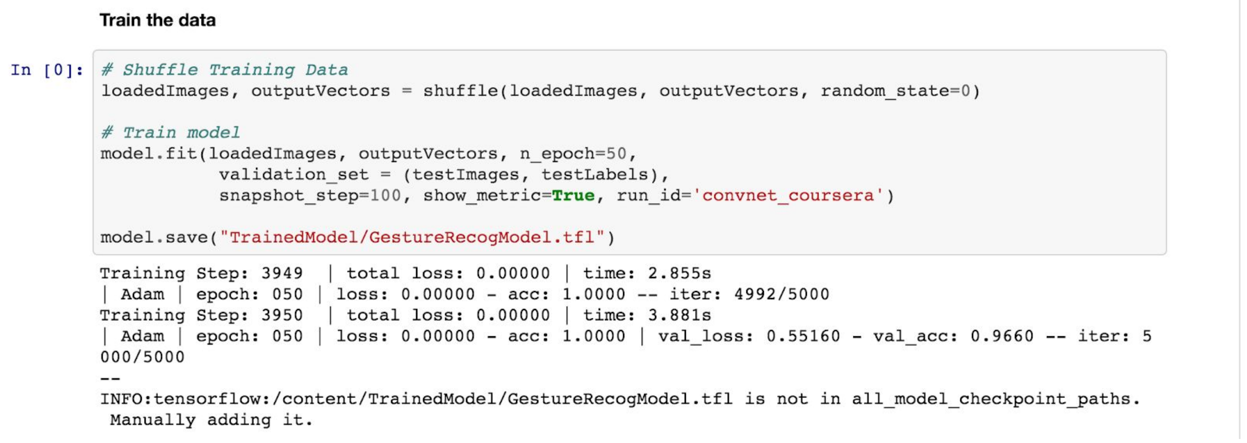
7) Setup the Model for training
Now, this is a critical stage of the training process. We have used deep convolutional neural network for training the model. There are 7 hidden convolutional layers with Relu as the activation layer and 1 fully connected layers which connects all the neurons. The model is trained across 50 iterations with a batch of 64. It is achieved in the following manner. To understand, what convolutional neural network means and how it is actually trained, you can go through these presentation slides that I have prepared.
Tensorflow Slides
The model is setup in following manner:



stride is the number of units the filter shifts each time.

```
In [0]: # Define the CNN Model
        tf.reset_default_graph()
        convnet=input_data(shape=[None,89,100,1],name='input')
        #32 Filters, 2 Stride, Activation is relu
        convnet=conv_2d(convnet,32,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,64,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,128,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,256,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,256,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,128,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=conv_2d(convnet,64,2,activation='relu')
        convnet=max_pool_2d(convnet,2)

        convnet=fully_connected(convnet,1000,activation='relu')
        convnet=dropout(convnet,0.75)

        convnet=fully_connected(convnet,5,activation='softmax')

        convnet=regression(convnet,optimizer='adam',learning_rate=0.001,loss='categorical_crossentropy',na
        me='regression')

        model=tflearn.DNN(convnet,tensorboard_verbose=0)
```

8) Training
   Training is done with all the information available from previous steps. We need training vector, training label vector, testing vector, testing label vector. It goes through each data and trains the model.



Train the data

```
In [0]: # Shuffle Training Data
        loadedImages, outputVectors = shuffle(loadedImages, outputVectors, random_state=0)

        # Train model
        model.fit(loadedImages, outputVectors, n_epoch=50,
                   validation_set = (testImages, testLabels),
                   snapshot_step=100, show_metric=True, run_id='convnet_coursera')

        model.save("TrainedModel/GestureRecogModel.tfl")
```

```
Training Step: 3949  | total loss: 0.00000 | time: 2.855s
| Adam | epoch: 050 | loss: 0.00000 - acc: 1.0000 -- iter: 4992/5000
Training Step: 3950  | total loss: 0.00000 | time: 3.881s
| Adam | epoch: 050 | loss: 0.00000 - acc: 1.0000 | val_loss: 0.55160 - val_acc: 0.9660 -- iter: 5
000/5000
--
INFO:tensorflow:/content/TrainedModel/GestureRecogModel.tfl is not in all_model_checkpoint_paths.
 Manually adding it.
```

9) Testing
   I have tested this model in various places. I tested the model outside the capstone Lab with a proper light and I achieved accuracy of 96 %. In WCC, I achieved accuracy of 90 %.

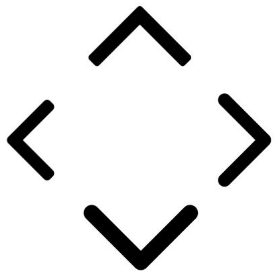
Android APP:

Android App consists of 4 activities which is responsible for controlling the motion of the car.

1) SplashActivity:
   This activity is responsible for producing the splash screen when you open the app. As soon as the app opens, it starts to animate the car image.
2) OptionActivity:
   OptionActivity provides two modes of control for the car. It opens up a car activity or gesture activity based on option selected. If the user selects gesture activity, it triggers gesture recognition signal on firebase. Based on that signal, the laptop starts the prediction of gesture.

3) CarActivity
   Car Activity is responsible for sending a signal to firebase based on which arrow is selected. Arrow(UP) means drive forward, Arrow(down) means drive in reverse direction, Arrow(Left) means turn left, and Arrow(Right) means turn right. Ontouch listener is used to give the control signal continuously instead of

onclicklistener.



4) GestureActivity:
   Gesture Activity continuously listens to updates from the firebase and updates the text field with the new gesture available. This is used as an assurance that the correct signal is given to the car.

Challenges and how to fix:

Ming Ma:

1) The first issue I have is the distance sensor. The measured distance sensor is not correct and not stable. Then, I found I use 100 Mhz clock to measure distance which is too high. Then, I use 10 Mhz clock and it works well. Also, I need to use 5V voltage to make this distance sensor work properly.
2) The second issue I have is our 3D model. Our model is too big to print by using the 3D printers in our school and I didn't realize this early. So, we only have our model ready and we can't print this. We may print this in the future.
3) The last issue I have is the alert. I find the alert is distorted by playing with the amplifier in the FPGA. Then, I find the frequency that I use is too low which is 8 khz. I fix this by using 16 khz clock.

Zhe Lu:

1) Unable to read the control signal via UART. First time when we tried to receive and send data through UART, it didn't work so that we found an example from xilinx that just connect the RX and TX port and run the code to test it, it works really good later.
2) Also for the UART communication, it always receive some garbage values, we tried a lot of ways to avoid these garbage, for example, we tried to give some delay because we think the UART needs some time to reset the buffer inside. However, at the end we found the reason why we receive garbes is because some time NodeMCU send it so that we finally solve this problem.

Ram Bhattarai:

1) Unable to program NodeMCU at the beginning because I did not press the reset button while programming. Also, I later found out that the GPIO0 which is 6th pin from the back of the Nodemcu board should be connected to Ground while programming.
2) Nodemcu was sending some garbage values through send buffer. Every Time before sending new data, the Serial port needed to be flushed.
3) Gesture recognition was a challenge at the beginning. In the beginning, I had an accuracy of 80 %. I was overtraining the model. Using the dropout functionality in tensorflow solved the issue. Overtraining or undertraining a model is really bad for training the model.

Future Work:

1. Use built-in camera from phone to recognize gestures.
2. Use 4-Wheel driven car model.
3. Train the model with images from different backgrounds
4. Try to do the gestures recognition in FPGA.
5. Print our 3D model

References:

1. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
2. https://github.com/gajjanag/6111_Project/tree/master/assets/audio_convert
3. https://github.com/SparshaSaha/Hand-Gesture-Recognition-Using-Background-Elllimination-and-Convolution-Neural-Network