Portland State
UNIVERSITY

ECE 544

PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ESD Design with FPGAs

ANDROBOT REPORT

Atharva Mahindarkar
atharva2@pdx.edu

Prasanna Kulkarni
prasanna@pdx.edu

Date: June 7, 2019

Contents

1   Project Overview

1.1   Introduction

The idea in this project was to create a robotic platform with two modes of operation. The first mode was the accelerometer mode. The second mode was the automated mode. In the accelerometer mode the bot was controlled using the accelerometer on a android phone using an app as an interface to switch between the two modes. In the automated mode the bot follows an obstacle detection algorithm. It will run in a straight line till it encounters an obstacle. When it encounters a obstacle it will turn to avoid it. The turn is a 90° turn. The bot will prioritize the direction without an obstacle for the turn. If the obstacle is detected in front of the bot then it will prioritize a right turn. The path that the bot takes is traced in the android app. The app has the option to save the path as a image. There is also an option to see the previously saved paths.
The underlying goal of the project is to use the concepts learnt in ECE 544 and ECE 558 and integrate them in an application.

1.2   Tools and Specifications

- Digilent Nexys4DDR FPGA board

- Microblaze 32-bit soft-core CPU.

- Node MCU with ESP8266 used with UART

- IR sensors

- Degilent Pmod HB3

- Motors

- Emgreat Robot Chassis Kit

- Arduino Mega (For Debugging)

- Firebase Database

- Xilinx Vivado and SDK.

1.2.1   Project Objectives

- Make an embedded system SoC on FPGA that can use UART to communicate to Firebase using a ESP8266 NodeMCU.

- Integrate the HB3 PMOD drivers into the embedded system and use them to drive the motors.

- Configure the NodeMCU to communicate to the Firebase

- Assemble the robot chassis along with the IR sensor, The HB3 drivers, the driver circuit, power supply, and the Nexys4 board.

- Develop the Android app to control the Firebase elements which will be used to drive the bot.

- Write the code for the obstacle avoidance mode of the robot, calibrate the control of the accelerometer

1.2.2 Stretch Goals

- Add the path mapping functionality to the Android app with the path showing the starting and the ending locations.

1.2.3 Additional Stretch Goals

- Add the ability to save the image in the android app and see the saved images in a gallery feature.
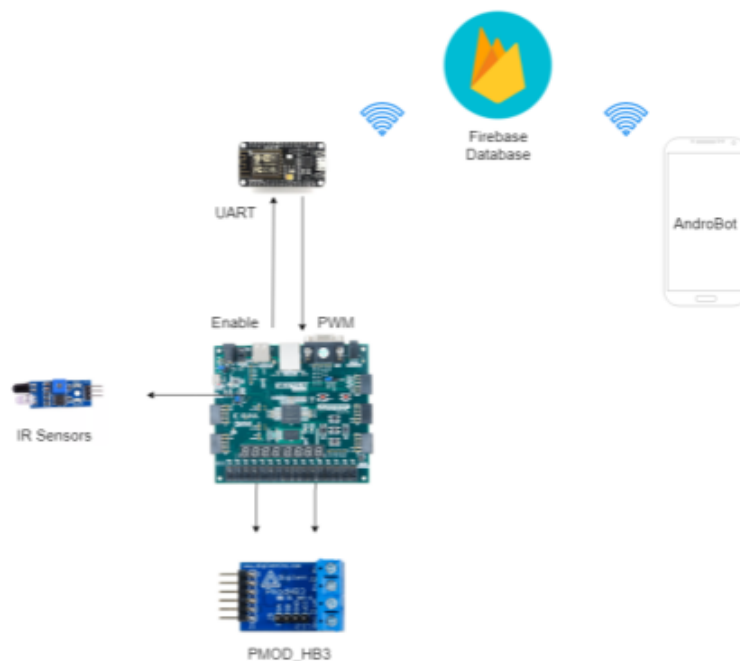
1.2.4 System Block Design

Figure 1: General System Block Diagram

2   Nexys4 Implementation

2.1   Objective

The objective for the Nexys4 implementation was to create a SoC system on the Nexys4 DDR FPGA, that could drive the robot wirelessly from a app on an Android phone. Specifically, the idea was to use the ESP8266 NodeMCU to read the Firebase database and drive the bot according to the elements in the database and the mode of operation.

2.2   Design

2.2.1   Hardware

The hardware portion in the SoC was primarily based upon the Closed Loop Control System project in ECE 544. An additional UART (Universal Asynchronous Receiver Transmitter) was added to the to the system, to communicate with the ESP8266 NodeMCU. There was also a GPIO added to interface the InfraRed (IR) LEDs. The hardware is built to run a FreeRtos. In terms of the block diagram this means that the `AXI Timer 0` has it's interrupt connected to the interrupt handler block `xlc_concat`. In spite of this the software code doesn't use any FreeRtos features. This was done in order to ensure that porting this project to a FreeRtos systems would require no modifications to the hardware.

PMOD Allocation:
On board the Nexys4IO module driver was used to contro the seven segment leds, the switches, and the buttons. The external pins in the hardware were tied to the PMOD connection headers.

Peripheral	Driver	Port
UART Lite	Driver provided in digillent repository	JC
PMOD HB3 Driver 1	Custom driver written by us imported as IP	JA
PMOD HB3 Driver 2	Custom driver written by us imported as IP	JB
IR Sensors	GPIO driver provided in digillent repository	JD

Table 1: PMOD Allocation Table

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
nexys4IO_0	0x44a00000	0x44a0ffff	S00_AXI	REGISTER
mig_7series_0	0x80000000	0x87ffffff	S_AXI	MEMORY
microblaze_0_local_memor...	0x00000000	0x0001ffff	SLMB	MEMORY
microblaze_0_axi_intc	0x41200000	0x4120ffff	s_axi	REGISTER
mdm_1	0x41400000	0x41400fff	S_AXI	REGISTER
axi_uartlite_1	0x40610000	0x4061ffff	S_AXI	REGISTER
axi_uartlite_0	0x40600000	0x4060ffff	S_AXI	REGISTER
axi_timer_1	0x41c10000	0x41c1ffff	S_AXI	REGISTER
axi_timer_0	0x41c00000	0x41c0ffff	S_AXI	REGISTER
axi_gpio_0	0x40000000	0x4000ffff	S_AXI	REGISTER
Pmod_HB3_1	0x44a20000	0x44a2ffff	S00_AXI	REGISTER
Pmod_HB3_0	0x44a10000	0x44a1ffff	S00_AXI	REGISTER

Figure 2: Address Space Allocation

UART Hardware:

We've used UART to communicate with the ESP8266 NodeMCU, the configuration of the UART peripheral is as follows.

Figure 3: UART Configuration

The UART Lite driver is added as an peripheral and acts as a slave on the AXI bus. The clocking signals used are `s_axi_clock` and `s_axi_resetn` respectively. As we can see in the configuration the baud rate is 115200. This UART peripheral is used to transmit and receive data from the ESP8266 NodeMCU. The code that makes the UART communication work is explained in the software implementation. Another UART Lite peripheral is used for debugging purposes.

PMOD HB3 :

The PMOD_HB3 is made up of two major sections. These are the two major files that are used in making the IP. The `pwm_gen.v` is the hdl code that generates the PWM signal that is used to drive that motor. The `rpm_detect.v` is the file that detects the speed of the motor by using a simple edge detection algorithm on the sa_input pin.

`pwm_gen.v` :

```verilog
always@(posedge clk )
begin
    if (!rst) begin
        pwm_counter <= 10'b0;
    end

    else begin
        pwm_counter <= pwm_counter + 1;
    end
end
```

Figure 4: `pwm_gen.v` counter snippet

The basic idea with the `pwm_gen` code is to have a counters. This counter is the `pwm_counter`. In order to improve the resolution of the PID control we have scaled up the PWM to a 11 bit value. This means that it will range from 0-2047. The counter simply increments the count on every clock cycle unless we encounter a reset.

```verilog
always @ (*) begin
  if (!rst) begin
    pwm = 1'b0;
  end
  else begin
    if (pwm_counter < pwm_count) begin
        pwm = 1'b1;
    end

    else begin
        pwm = 1'b0;
    end
  end
```

Figure 5: `pwm_gen.v` pwm snippet

The idea here is quite simple, the signal `pwm` goes high if the count is below a certain value and remains low other wise. A point of interest here is the `pwm_count`. This variable is user given the requested pwm of the system. In order to get this variable into the system we are using a register inside the PMOD HB3 driver. The way we have done this is by editing the hdl code inside the AXI Bus instantiation of the peripheral. Something that was extremely useful in doing this was the RojoBot project from ECE540.

`rpm_detect.v :`

The `rpm_detect` file is basically a tachometer. It is used to measure the speed at which the motor is running. In this project this functionality is not used. This module is mentioned here for the sake of completeness.

IR Sensors

: The IR Sensors are interfaces using a single GPIO of 16- bits. The IR sensors
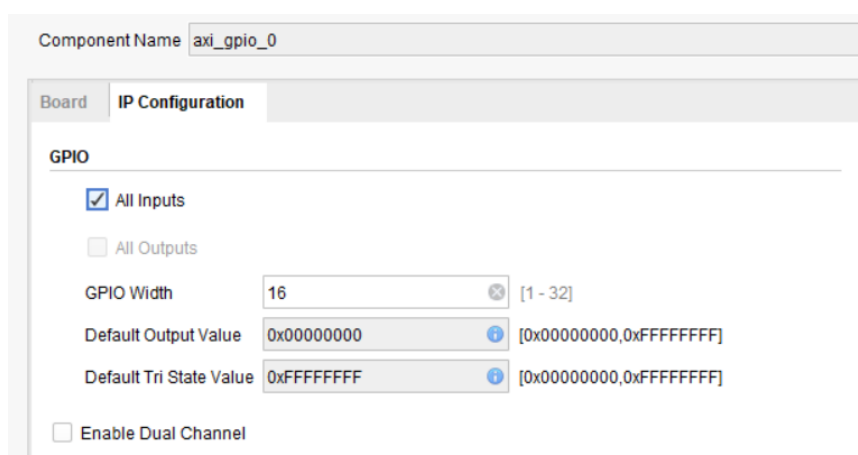

Figure 6: GPIO Configuration

are active low sensor that give a pulse of 0 when there is an object detected and 1 otherwise. We have used PMOD Connetcor Headers on the board to connect the IR Sensors with the FPGA.
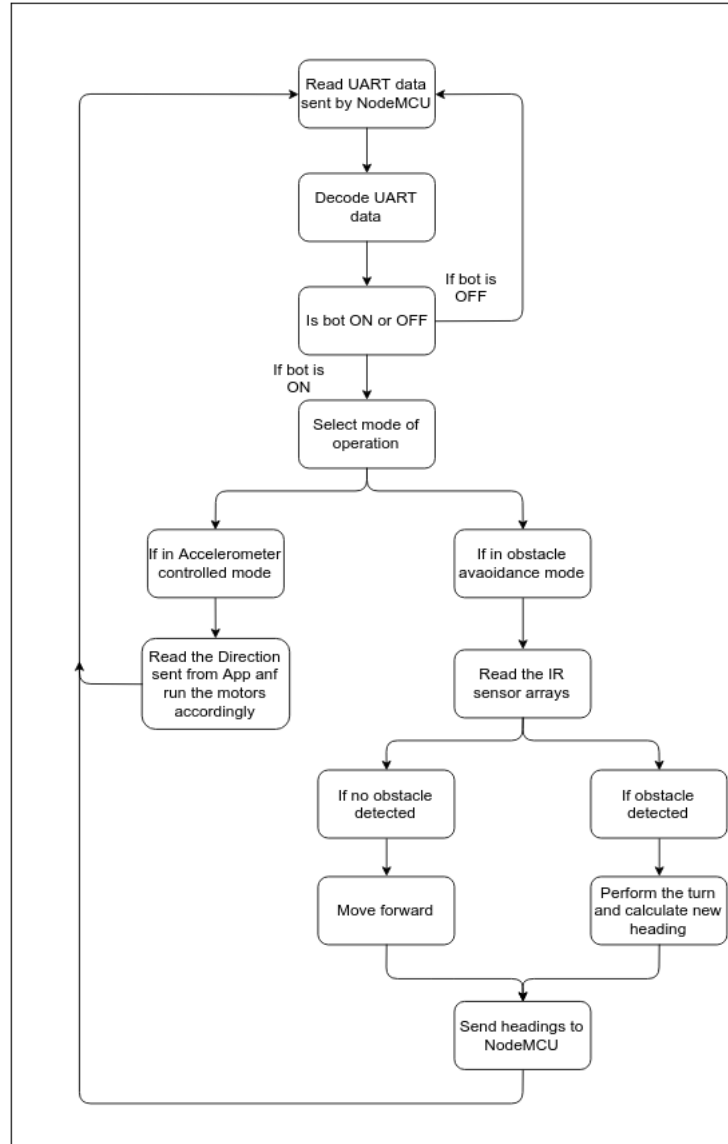
2.2.2 Software

Figure 7: Software Block diagram

The software application for this project is developed on Xilinxs Standalone OS board support package. This software application is responsible for polling the Firebase data base which is done with the help of ESP8266 NodeMCU . An UART channel is established between both to send Firebase data from ESP8266 NodeMCU to Nexys4 board and to get heading data in the opposite direction. Depending upon the mode of operation the bot is either driven by the directions given by the user from mobile app or it runs in automated obstacle detection mode.

```
status = XUartLite_Initialize(&UartLite, UARTLITE_DEVICE_ID);

if (status != XST_SUCCESS)
{
    xil_printf("Failed");
    return XST_FAILURE;
}
```

Figure 8: UART Software Configuration

UART

In this project, we are using two Uartlite blocks, one for debugging purpose and 2nd for UART communication between Nexys4board and NodeMCU. For this, we instantiated an instance of XUartlite. It is then initialised in do_init() function.

In androbot function, we are continuously polling the data sent over the UART. For receiving the data from UART channel, we need to create a buffer of u8 data type. We need to use u8 data type because over the UART channel, we are receiving single byte of data at a time. If buffer is not of u8 data type, then data in that particular buffer gets shifted and hence data gets corrupted.

```
int rcv = XUartLite_Recv(&UartLite,datarcv,1);          // receives uart data into "datarcv buffer"
                                                        // the function return 1 or 0 depending upon sucessful t
uart.uart_in[0] = datarcv[0];                           // set the first byte of data to our bots uart data
```

Figure 9: UART Software Receiver

UART Data Encoding/Decoding

For this application, we need to send various data like, ON/OFF switch, mode of operation, directional data from android app to Nexys4 board. If we try to send this sequentially, it would add latency, hence we decided to encode these 3 parameter in a byte. As byte can have value between 0 to 255, we had to come up with the method where these three parameters would be encoded between this range. Following is the encoding methodology employed in our design

- **ON/OFF switch status** -

    - if 1 then bot is stopped and turned off
    - if 2 then bot is turned on

- **mode of operation** -

    - if 1 then in accelerometer mode
    - if 2 then in obstacle detection and mapping mode

- **direction from app** -

  - if 1 then stop
  - if 2 then go forward
  - if 3 then go reverse
  - if 4 then turn right
  - if 5 then turn left

In NodeMCU, the data is read from Firebase for these parameters and encoded such a way that, ON/OFF switch status is multiplied by 100, mode of operation is multiplied by 10 and these are added with direction from app. Hence it gives us a number in between 111 to 225, which was our aim.

When this byte of data is received via UART as explained above, Decode_UART function is called, which intern again decodes this received byte of data. The following code snippet explains the decoding process -

```
void Decode_uart()
{
    int input = uart.uart_in[0];              // load the byte of data in temperory variable

    uart.on_off_status = input / 100;         // extract the 100th place data
                                              // which if on/off switch data in app
    input = input % 100;                      // remove 100th place digit anf get 2 digit number

    uart.mode_operation = input / 10;         // extract the 10th place data
                                              // which is mode of operation
    input = input % 10;
                                              // remove 10th place digit anf get single digit numbe
                                              // which in turn is a direction data from app
    if(uart.mode_operation == 1)              // if mode of operation is 1, i.e., in accelerometer mode
    {
        uart.direction_from_app = input;      // then only assign unit's place data to direction parameter of the bot
                                              // otherwise discard
    }
}
```

Figure 10: UART Software Decode

After decoding the data, if the bot is switched off, then it again jumps to UART data polling. If it is switched on then depending upon the mode of operation, further action is taken. Modes of Operation -

Our bot works in two modes of operation, accelerometer controlled mode, or in obstacle detection mode. If the mode of operation is accelerometer controlled mode, then only directional data sent from app is read otherwise it is discarded. It can be seen in code snippet above.

Accelerometer Control Mode

This mode is implemented in accelerometer_mode function. In this function, a switch case is implemented. Each case is given a number which are used for encoding directions as explained above. Then in every case statement, the directions and PWM data are assigned to the motors. Also in this mode, if IR sensor detects any obstacle, bot stops immediately. In case the bot has stopped after detecting an obstacle, it has

```
case 2:                                                          // if directiongiven is FWD
    xil_printf("\nforward\n");
    if(ir_input == 255)                                          // if no IR sensor is triggered
    {                                                            // i.e., no obstacle detected
        obstacle[0] = 2;
        PMOD_HB3_mWriteReg(PMODHB3_0_BASEADDR, 4, 1);            // motor direction is set for
        PMOD_HB3_mWriteReg(PMODHB3_0_BASEADDR, 12, fwd_speed);  // FWD movement and PWM is assigned

        PMOD_HB3_mWriteReg(PMODHB3_1_BASEADDR, 4, 1);
        PMOD_HB3_mWriteReg(PMODHB3_1_BASEADDR, 12, fwd_speed);

        /*if(count == 50)
        {
            int a = XUartLite_Send(&UartLite, obstacle, 1);
            count = 0;
        }*/
    }
```

Figure 11: Accelerometer Software Forward Movement Snippet

to be manually moved in the opposite direction. The forward functionality is defined as follows All other modes are implemented in the same way and can be seen in the file.

Obstacle Detection Mode

This mode is implemented in obstacle_detection_mode function. In this function, depending upon the IR sensor's input, movement is controlled. If the obstacle is detected then, depending upon which IR sensors are getting triggered, 90 degree turn is performed and new heading is calculated. Otherwise bot keeps on moving in forward direction keeping the heading same. The functioning of IR sensors is in such a way that, when triggered gives output as 0 and when not triggered gives output as 1. By default, the heading is North and bot moves in forward direction until any obstacle is detected. When obstacle is detected, depending upon which sensor is not triggered, bot moves in that direction. But in the case when only front IR sensor gets triggered, bot turns right as higher priority is given to this turn.
The turn functions, turn_right and turn_left are calibrated to perform approximately 90 degrees of turn. After performing this turn, depending upon current heading, new heading is calculated. This operation is performed in calculate_heading function.
In this software we have also encoded the headings as below -

Heading value	Direction
1	East
2	North
3	West
4	South

Table 2: Heading Encoding

When the new heading is calculated, it is sent to NodeMCU via UART as explained in code. If all the IR sensors gets triggered, the stop bit is raised in the function, and in

this case instead of heading, stop bit is sent via UART to NodeMCU. The NodeMCU further processes these headings or stop bit for calculating X and Y coordinates and this processes is further explained in next section.

2.3   Results

The SoC was implemented on the Board and using an Arduino Mega data strings were sent and received using UART to test the functionality. The SoC was found to produce the desired functionality. The

2.4   Challenges

- Initially we faced problem with Uartlite's receiving buffer, as it was getting overflowed, we used XUartLite_ResetFifos function to clear the buffer.

- As one of our extended streach goals, we decided to implement complete project in FreeRTOS. But when we created and implemented threading model, many a times data packets losses were observed by us. But our program in FreeRTOS when no threading model was used, it worked perfectly. So we decided to revert back to standalone OS.

3   NodeMCU Implementation

3.1   Objective

As we are not having onboard WiFi feature on Nexys4 board, we need to interface an external module like NodeMCU ESP8266 to allow the Nexys4 board to communicate with Firebase. With the help of Apache API 2.0 developed for Arduino, it is convenient to poll the data from Firebase or send the data to Firebase. Due to this module we can use Nexys4 board for IoT applications.

3.2   Design

In NodeMCU, first it is connected to WiFi and then it is connected to Firebase using provided host link and authentication key. During the setup process of NodeMCU, the default X and Y coordinates are set to (500, 500). As the Apache API 2.0 is not having the functionality of addValueEventListener, whereas it is available in android app development, we had to improvise and had to run polling loop at particular delay. In each iteration of polling loop, which is implemented in void loop function of NodeMCUs code, we are first reading the ON/OFF switch state, mode of operation and direction data from Firebase. The data is encoded and sent to Nexys4 board as shown in following code snippet.

```
n = Firebase.getInt("botOn");                    // read the ON/OFF switch data fromfirebase
byte_sent = n * 100;                             // multiply this data by 100

n = Firebase.getInt("autoMode");                 // read the mode of operation
byte_sent = byte_sent + (n * 10);                // multiply by 10 and add this to byte_sent

n = Firebase.getInt("directionfromApp");         // read the direction from app

byte_sent = byte_sent + n;                       // add this data to unit's place of data

//Serial.print(" firebase data is - ");

Serial.write(byte_sent);                         // send the byte of data to nexys4 board
```

Figure 12: NodeMCU Software Serial Data Send

When our bot is in obstacle detection mode, Nexys4 board sends the heading of the bot to NodeMCU. These headings are used to calculate the new coordinate location of the bot. If the bot is heading East, the X coordinate value is incremented and if bot is heading West, the X cordinate value is decremented. But for North and South direction the sequence opposite, because in androidsystem the mapping of the Y axis is inverted when compared to standard cartesian coordinate system. Hence when headding is North, Y cordinate value is decremented and when heading is South, Y cordinate value is incremented.
The following code snippet explains how new coordinate is updated -

```
if(heading == 2)                                 // if heading is north
{
  y = Firebase.getInt("Y");                      // read the previous Y cordinate
  if( y<1000 && y>0)                             // check if bot is crossing the map boundry
    y = y - 20;                                  // decrement the pixelcount by 20
  Firebase.setInt("Y", y);                       // update the new Y cordinate
  Firebase.setInt("pointSet", 1);                // set update bit to 1 so that android app
                                                 // knows new data is being written
}
```

Figure 13: NodeMCU Software Coordinate Update

Here we also needed to take precaution that X and Y coordinate do not cross there boundaries. If the boundaries are crossed, the updating of coordinates stops. Also after every time a coordinate is updated, Point-Set node on the Firebase needs to be set to 1. This lets android app know that new coordinate values are updated and it reads these values and updates the plot on the screen at real time.

3.3 Results

We successfully established UART communication between NodeMCU and Nexys4 board for sending and receiving the data from Firebase. The coordinate update functionality developed on NodeMCU also works flawlessly and results are reflected on android app side at real-time.

3.4 Challenges

The Nexys4 board continuously keeps on sending the heading data, which causes the Rx buffer of the NodeMCU to overflow. This results in delayed responses. The Serial.flush() function only clears the Tx buffers, so that function does not help in clearing Rx buffer of NodeMCU To tackle this problem, we decided to terminate the Serial port connection each time after receiving byte of data from Nexys4 board. The connection is again established just after terminating it. This resulted in system resetting all the buffers of the UART port of NodeMCU and this approach helped us solve the latency issue.

```
Serial.write(byte_sent);                        // send the byte of data to nexys4 board

if(Serial.available())                          // 1st latching of serial port
{

  while(Serial.available())                     // 2nd latching of serial port
  {
    byte_recv = Serial.read();                  // read the data sent from Nexys4 board

  }

  Serial.end();                                 // end the serial connection to clear Rx buffer
  Serial.begin(115200);                         // initialise the serial port and assign baudrate
}
```

Figure 14: NodeMCU Software Latency

4 Android Application

4.1 Objective

The objective of the android application was to give the user control over the bot, and get real- time information from the bot. Specifically, the user should be able to turn on and off the bot, switch between app-controlled mode and obstacle detection mode. In return, the bot sends its current location coordinates to the app. Using the coordinates received, the app displays the current location coordinates and the path the bot followed. The app should support features such as saving the path as pictures to external storage and displaying the saved paths using an inbuilt gallery.

4.2 Design

The application is made up of 3 activities MainActivity, GalleryActivity and PhotoActivity. On opening the application, user is greeted with the MainActivity. This contains the graph where the path is plotted, current location of bot in X, Y coordinates, switches to power the bot and choose between modes, buttons to save path

as picture and to open gallery and an image thats shows the last path saved to storage. GalleryActivity displays all the images found in the apps external storage folder in a MainActivity is designed such that the name of the Project and application is displayed on top along with an image (which is also the launcher icon). Below it is a RelativeLayout which contains a custom view to draw the path, and a TextView where the time is printed. Switches to power the bot and choose between modes, along with direction information and coordinates are visible underneath this RelativeLayout inside a LinearLayout in horizontal orientation. An ImageView to display the last picture saved to the external storage, and buttons to save path as picture and to open gallery are below the LinearLayout. Layout file of the GalleryActivity contains only a RecyclerView. A separate layout file defines how the image and file name are to be arranged to fill the RecyclerView. PhotoActivity contains a single ImageView to display the image clicked on in GalleryActivity. In the OnCreate
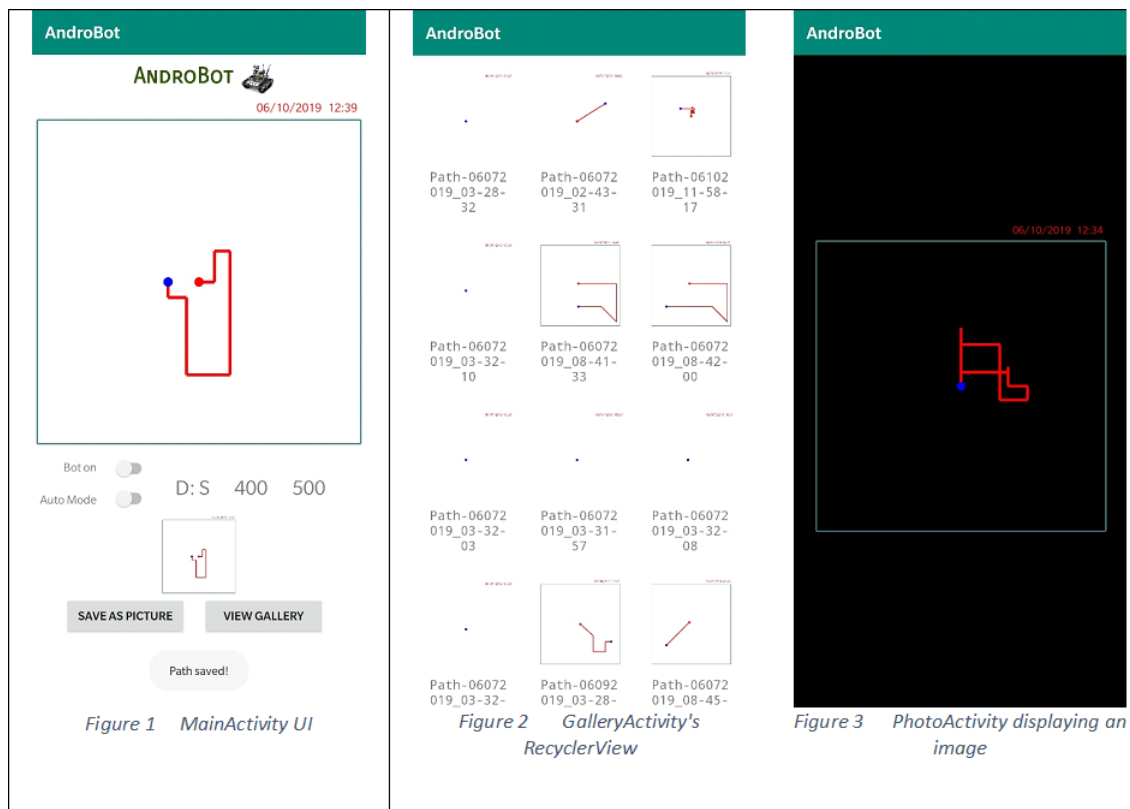

Figure 1   MainActivity UI

Figure 2   GalleryActivity's RecyclerView

Figure 3   PhotoActivity displaying an image
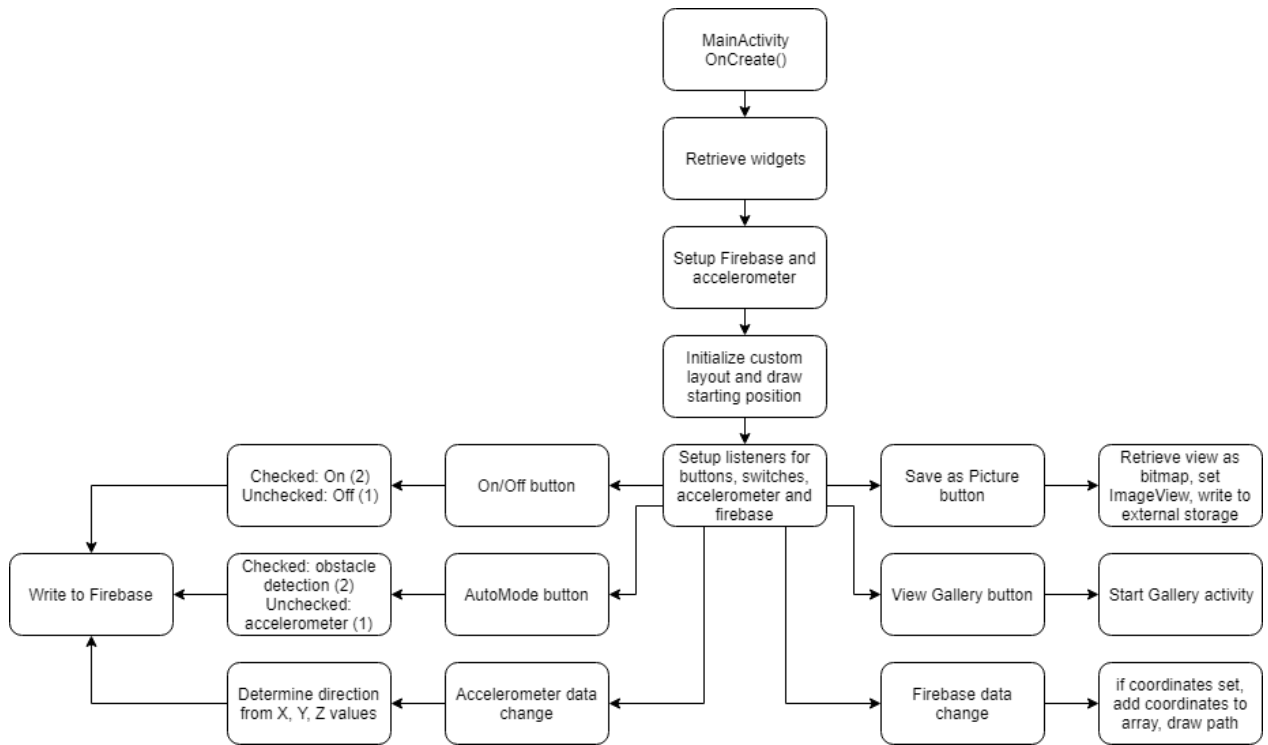
Figure 15: Android App Snapshots

Figure 16: Android App Block Diagram

method of the MainActivity, all widgets found in the layout are retrieved and initialized. Communication with Firebase is also initialized here along with retrieving and initializing accelerometer interface. Once the layout is inflated, width and height of the custom view is retrieved. This is done here since the custom class that extends the View class to draw lines and shapes use pixels as coordinates, and since the pixel density differs with each phone, the dimensions of the screen are to retrieved at run time. Once this is done, a bias value is calculated such that the plot constrained between (0, 0) and (1000, 1000). This means scaling the graph so it is centered and

```
//get width and height of view in pixels dynamically
//height and width vary with phone as it is based on pixel density
lineViewHeight = mLineView.getMeasuredHeight();
lineViewWidth = mLineView.getMeasuredWidth();

//calculate bias so map is centered at 500, 500
//map is constrained between 0,0 and 1000,1000
hBias = (lineViewHeight - 1000)/2;
wBias = (lineViewWidth - 1000)/2;
```

Figure 17: Android Code Snippet 1

fills the space on devices with higher pixel density. Once this is done, a single point is displayed on the center of the graph denoting the bot position. Timestamp is also updated at this point. Methods that read accelerometer values are overridden such that 5 directions can be read from the current X, Y, Z coordinates. This direction is updated on the app and firebase. A data change listener is implemented to listen for data changes on Firebase. The application specifically listens for X, Y coordinates

and updates the path only when the change. The coordinates are added to an array that contains all the coordinates sent by the bot (path history is maintained in this array). This array is then passed to the custom View class to draw the path. Times-

```
paint.setColor(Color.RED);   //set draw color to red

paint.setStrokeWidth(10);    //set draw width

//draw lines between all the coordinates in the array
for(int i = 0; i < pointA.size()-1; i++)
    canvas.drawLine(pointA.get(i).x, pointA.get(i).y, pointA.get(i+1).x,
pointA.get(i+1).y, paint);

//draw circle to show starting point
canvas.drawCircle(pointA.get(0).x, pointA.get(0).y, 15, paint);

//change color to blue
paint.setColor(Color.BLUE);

//draw circle to denote current location of bot in blue
if(pointA.size() > 1)
    canvas.drawCircle(pointA.get(pointA.size()-1).x, pointA.get(pointA.size()-1).y,
15, paint);
```

Figure 18: Android Code Snippet 2

tamp is updated once again when the coordinates change. Clicking on the Save as Picture button retrieves the RelativeLayout (with the custom view and timestamp) as a bitmap and displays it to the ImageView. Before saving it to storage, the app looks for permissions to save to external storage. If permission hasnt been granted, it asks for permission first then proceeds to write it to external storage. Clicking on View

```
//get permissions to store file if already not granted before saving to external
storage
if (checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED) {

    requestPermissions(new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE);
}
```

Figure 19: Android Code Snippet 3

Gallery button starts the GalleryActivity. On creating, the GalleryActivity first reads all the images found in the external storage folder into a bitmap and string array. Bitmap array contains the image while string array contains the file names. It again asks for permission to read first if it has not been granted already. Once all files have been read, the activity uses the adapter and view holder classes to display the images in a scrollable grid layout. This activity also implements an OnImageClickListener interface that listens for clicks and starts the PhotoActivity when an image is clicked on. GalleryActivity passes the bitmap as an extra to this activity. PhotoActivity just displays this bitmap to the ImageView.

4.3   Results

We are able to successfully demonstrate the working of the application. The Application drew the path of the bot in obstacle detection mode. We were also able to control the bot using the phones accelerometer. Application was able to successfully store paths as images to external storage. The images could be viewed in the phones file manager and phones gallery. Gallery activity displayed all files found in storage directory as intended. Photo activity displayed the correct image clicked on.

4.4   Challenges

- We faced multiple challenges creating this app, but were able to make it work at the end. Initially, the view class that plots the path would only draw lines between 2 coordinates. If there was an existing line, and a new line was to be drawn, it would erase the existing line before drawing the new line. In order to solve this issue, we used an array to store the coordinates. So, whenever a new coordinate is added, lines between all the previous coordinates were redrawn.

- When trying to save the plot as a picture, converting the view to a bitmap executed only once. Meaning, after saving a picture once, the updated view did not get converted to bitmap again the picture that was saved first was getting saved again. Looking up on the internet, multiple suggestions were found. However, all of the suggestions resulted in crashes. All of the suggestions told to building a drawing cache, convert it to bitmap and destroy the cache after converting. With a bit of experimenting, rearranging to destroy cache first before rebuilding seemed to work. Surprisingly not one of the stackoverflow answers suggested this order.

- Since the view did not have a background (even though it is white in the app), the saved images background was black. Since the path was drawn in black initially, we thought this was an issue. However, changing the color to red and blue clarified that the background buried the path since they both were same color.

- Adding permissions to read and write from/to external storage did nothing Logcat printed permission denied every time the app tried to read or write. We found that the app had to ask user for permission at runtime. Adding code that asked for user permission at runtime solved this issue.

- Adding permissions to read and write from/to external storage did nothing Logcat printed permission denied every time the app tried to read or write. We found that the app had to ask user for permission at runtime. Adding code that asked for user permission at runtime solved this issue. We decided to plot our map between (0, 0) and (1000, 1000). However, the View class used pixels to determine coordinates, and since the pixel density varied with each brand and make of phone, we had to come up with some math to solve this issue. Height and width of the layout was obtained at runtime, and bias values were

calculated such that (500, 500) would be displayed right in the middle of the layout. This bias also gave the graph some space between the edges of the coordinates and the border of the layout.

5 The Robot

The boards had to be mounted on a Emgreat Robot chassis. One of our biggest challenges was to give enough power to Nexys4 board, NodeMCU and the motors for correct operation. If the power was low it can affect the robots performance in terms of detecting obstacles and making turns. So it was very crucial to supply enough power to the robot. We made a two layer chassis, the bottom part had the motors mounted to it and also it hold the power circuit for the PMOD HB3 drivers. It has all the connections to power the motor. The upper chassis has the Nexys4 board mounted on top of it along with the NodeMCU and the IR sensors which are placed upfront.We used two power banks to power the whole robot. One to power the Nexys4 board and the motors and the other one to power the NodeMCU, The power bank was 20,000 mAh, which gave us sufficient power to run the robot for a long time.
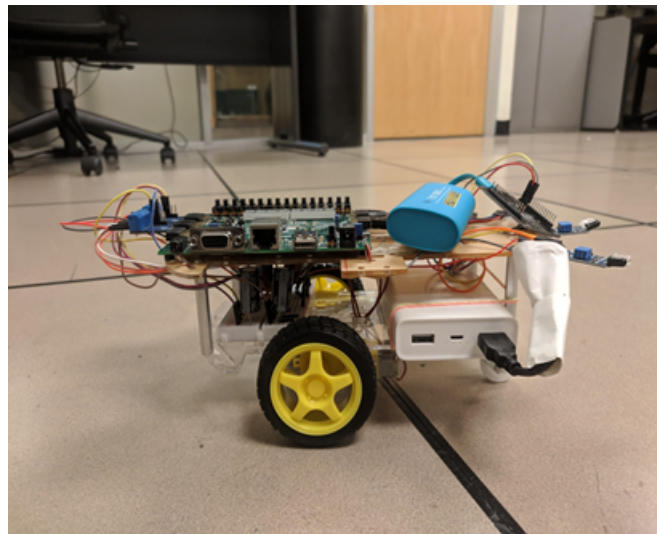
Figure 20: The Androbot

6 Goals Achieved

- Built a robot to work in two modes. Aceelerometer controlled and obstacle avoidance.

- In Accelerometer control the bot can be controlled using the accelerometer on an Android phone.

- In obstacle avoidance mode the bot can detect obstacles in its path and turn to avoid the obstacle.

7 Stretch Goals Achieved

- Can Select the mode in which the bot has to function in the app.

- The app can also save the mapped path as an image to external memory

- Gallery feature in the app to see the saved photos

8 Future Scope

- Improve the IR sensor to get better range

- Can be modified to be used as a Maze solver robot.

- Path Scaling while mapping the bot position.

9 References

- Getting Started Guide for Closed Loop Control provided by Prof Kravitz.

- The Project description in the Closed Loop Control Assignment release document.

- Diligent Documentation about Pmod HB3

- Nexys4 DDR Device Manual

- NeXplorer Report by Suryansh Jain and team, provided by Prof. Kravitz