Course Title: CEE-445 Embedded System Design
Project Title: AMR HMI Project

Student Names: Jacob Hillebrand, Milien McDermott
Instructor Name: Dr. Cheng Liu
University of Wisconsin-Stout
Menomonie, WI 54751

December 18th, 2019

Table of Contents

1.0. PROJECT DESCRIPTION

1.1. Introduction

The idea for this project stems from a need of a related, separate project. For a senior capstone project, the members of this team worked with a larger group of Computer and Electrical Engineering students to design an Automated Mobile Robot (AMR). The AMR will be able to perform real-time navigation in Fryklund Hall, will have a carrying capacity of 300 lbs, and will essentially be an automated retrieval robot. The plan is for the robot to be able to take instructions to travel to a given room, retrieve a package, and deliver the package to a different room using only the onboard navigation. To provide this navigation, the robot will be equipped with omnidirectional wheels, and will use a Raspberry Pi 4 combined with a Lidar scanner and absolute orientation sensor. These tools will allow it to create a map of its environment, chart a course, and perform collision avoidance throughout the route. The project is a challenging undertaking, but the group is confident in achieving their goal.

The project described in this report is supplemental to the goals of this capstone project. The Lidar and orientation sensor have both already been successfully interfaced with the Raspberry Pi and can pass data to it. However, this data is not easily accessible to a user outside of the Pi, save for an ssh connection. To fix this, the goal of this Embedded Systems Project was to provide a means of viewing this data in an easy format on an Android device or PC, thus reducing the need for ssh in the general operation of the robot. This will prove useful in the future, both in designing a Human Machine Interface (HMI) for the AMR and general debugging throughout the design process. As such, this design process is what will be described in the following report.

2.0. BILL OF MATERIALS

Part Name[1]	Cost
Raspberry Pi 4	$55.00
Cooling Fan	$13.95
Heatsinks	$7.66
Micro SD Card (128 GB)	$19.99
BNO055 Absolute Orientation Sensor	$32.00
Google Firebase Realtime Database	$0
RPi LIDAR Scanner	$114.95
Android Device	Provided
Total	$243.57

[1] All products were purchased through Capstone funding, not CEE 445 funding

3.0. DESIGN PROCESS

3.1. HARDWARE DEVELOPMENT

Hardware setup was relatively straightforward for this project. To begin, a Raspberry Pi 4 was chosen as the embedded system, primarily as it was the control device being used in the AMR already. It was also for this reason that the Lidar and orientation sensor were used in this project design. In fact, beyond the Android device, these were the only physical hardware devices used for the project. Wiring for the Lidar was very simple; it utilized the USB interface on the Raspberry Pi for Serial Communication. The BNO055 needed to be wired up to the GPIO on the Raspberry Pi, and a full diagram of this wiring configuration can be found in section 3.2, Figure2. The bulk of the work for this project can be found in section 3.3. SOFTWARE DEVELOPMENT.

3.2. WIRING DIAGRAMS

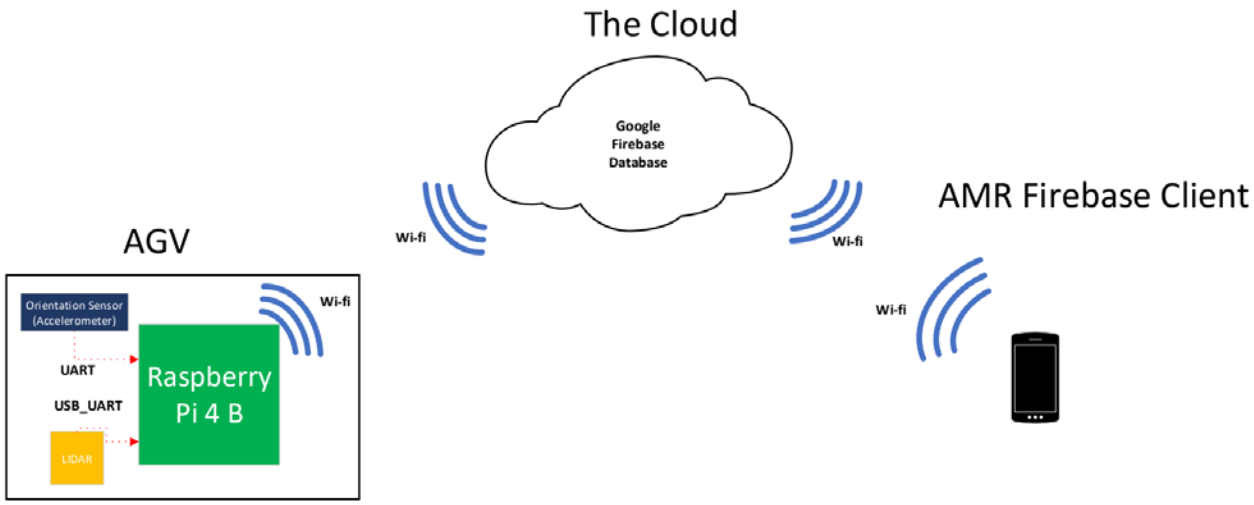

Figure 1 - Network Diagram

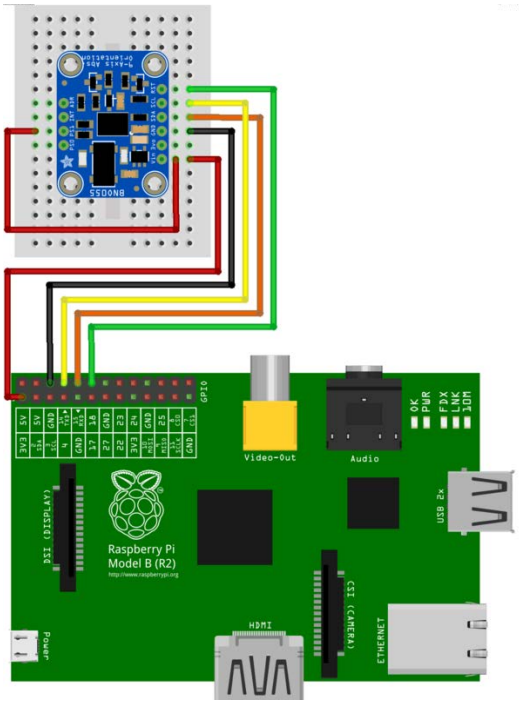

Figure 2 - BNO055 UART Wiring Diagram

3.3. SOFTWARE DEVELOPMENT

The majority of the work done on this project consisted of the software design. To begin, the Pi needed to be able to interface with both the sensors being used. The Lidar sensor was relatively easy to get up and running, as the company that made it, Slamtec, had a software package that could be run on the Pi and provide a graphical interface to view a real-time map that the Pi was producing. Due to the packaged nature of this program, it was decided not to pass this information to the database, and instead simply view it on a PC through X11 forwarding. This method was straightforward and provided a look at the Lidar map on an external device without adding unnecessary complications. All utilities used were built into the Pi and PC, and provided a quick and easy way to view the necessary data.

With the orientation sensor, however, it was decided that a different approach would be taken. To begin with, there were no easy ways to interface with the BNO055 from the Pi. After some research, it was decided that the sensor would be wired to the GPIO on the Pi and communicate over a Serial connection. Interestingly enough, this method required disabling the login capabilities of these pins, as the OS on the Pi would otherwise attempt to provide tty login capabilities to the orientation sensor. Once this configuration had been implemented, data then had to be retrieved from the sensor.

To both retrieve data from the sensor and push that data to a cloud database, a Python script was written on the Pi. This script began by importing a library from Adafruit that provided utilities for accessing the orientation sensor. Through this library, the script was able to initialize an object for the sensor based on its device code in the OS and pull data from the sensor through the object. Then, the script utilized a special library to create an object for the online database used and push the data from the orientation sensor to it as a json object. The script repeated itself every second, providing constant positional updating from the sensor to the cloud.

For the cloud database, Google Firebase was chosen. This service from Google provides a free-to-use database in the cloud, allowing a limited number of requests per month to remain in the free tier. This fit the project needs perfectly and was surprisingly straightforward to get up and running. No authentication was required to use the database, either. While this is a huge security hole that would need to be addressed in the future of the Capstone project, it was very convenient for the progression of this project.

Another benefit of Google Firebase is that it integrates well with Android apps, which was the final piece of the project design. To add the database to an app, one simply needs to add the app's ID to the database's configuration information and download a google-services.json file. This file then is added directly to the Android Studio project and provides all of the configuration information required to access the database. From this file, a database object can be made that references the information provided in the .json file, and data can be pulled from the database using that object in the Java code of the application.

As such, using the database object, the Android application was created in Android Studio. It consisted of one activity, with TextView objects for each datapoint and a Button object to initiate a data request. Java code was then added so that, when the Button was pressed, a listener was placed on the database using the aforementioned database object. Whenever the listener detected that the database had been updated, it updated the database object in the app with the new data. This new data was then written to the TextView objects, and the activity always reflected the most up-to-date information in the database. With this component, the software design portion of this project is complete. For a more in-depth look at the software design, see the 4.0. PROJECT ILLUSTRATION section below.

4.0. PROJECT ILLUSTRATION

4.1 Raspberry Pi Code Snippets (firebase_marvin.py):

```
import logging
import sys
import time

from Adafruit_BNO055 import BNO055

###############FIREBASE SETUP
#import firebase so we can push to the database,
# and datetime to grab the current date and time
from firebase import firebase
import datetime

#abstract the url away into a variable, and set up
# the firebase API object
amr_database = 'https://amr-sensor-data.firebaseio.com/'
firebase = firebase.FirebaseApplication(amr_database)

#grab current date and time,
# splice into string objects for each
d = datetime.datetime.today()
current_date = d.strftime('%d-%m-%y')
current_time = d.strftime('%H:%M:%S')
timestamp = d.strftime('%H:%M:%S %d-%m-%y')


# Create and configure the BNO sensor connection.  Make sure only ONE of the
# below 'bno = ...' lines is uncommented:
# Raspberry Pi configuration with serial UART and RST connected to GPIO 18:
bno = BNO055.BNO055(serial_port='/dev/serial0', rst=18)
# BeagleBone Black configuration with default I2C connection (SCL=P9_19, SDA=P9_20),
# and RST connected to pin P9_12:
#bno = BNO055.BNO055(rst='P9_12')


# Enable verbose debug logging if -v is passed as a parameter.
if len(sys.argv) == 2 and sys.argv[1].lower() == '-v':
    logging.basicConfig(level=logging.DEBUG)

```

Figure 3 - Library Imports, database, time, and sensor object setup

```
# Initialize the BNO055 and stop if something went wrong.
while True:
    try:
        if not bno.begin():
            raise RuntimeError('Failed to initialize BNO055! Is the sensor connected?')
    except:
        print('Failed bno init, retrying');
        time.sleep(1)
        continue
    else:
        break

# Print system status and self test result.
status, self_test, error = bno.get_system_status()
print('System status: {0}'.format(status))
print('Self test result (0x0F is normal): 0x{0:02X}'.format(self_test))
# Print out an error if system status is in error mode.
if status == 0x01:
    print('System error: {0}'.format(error))
    print('See datasheet section 4.3.59 for the meaning.')

# Print BNO055 software revision and other diagnostic data.
sw, bl, accel, mag, gyro = bno.get_revision()
print('Software version:   {0}'.format(sw))
print('Bootloader version: {0}'.format(bl))
print('Accelerometer ID:   0x{0:02X}'.format(accel))
print('Magnetometer ID:    0x{0:02X}'.format(mag))
print('Gyroscope ID:       0x{0:02X}\n'.format(gyro))

print('Reading BNO055 data, press Ctrl-C to quit...')
while True:
    # Read the Euler angles for heading, roll, pitch (all in degrees).
    d = datetime.datetime.today()
    current_date = d.strftime('%d-%m-%y')
    current_time = d.strftime('%H:%M:%S')
    timestamp = d.strftime('%H:%M:%S %d-%m-%y')
    heading, roll, pitch = bno.read_euler()
    # Read the calibration status, 0=uncalibrated and 3=fully calibrated.
    sys, gyro, accel, mag = bno.get_calibration_status()
    # Print everything out.
    print('Heading={0:0.2F} Roll={1:0.2F} Pitch={2:0.2F}\tSys_cal={3} Gyro_cal={4} Accel_cal={5} Mag_cal={6}'.format(
          heading, roll, pitch, sys, gyro, accel, mag))

```

Figure 4 - Pull data from the orientation sensor and assign values to variables, print status for verification

```
111     while True:
112         try:
113             #Send to firebase
114             result = firebase.put(
115                 'orientation sensor',
116                     'data', {
117                         'last_update':str(timestamp),
118                         'Heading':str(heading),
119                         'Roll':str(roll),
120                         'Pitch':str(pitch),
121                         'System Calibration':str(sys),
122                         'Gyroscope Calibration':str(gyro),
123                         'Accelerometer Calibration':str(accel),
124                         'Magnetometer Calibration':str(mag)
125                         }
126                     )
127         except:
128             print('Failed firebase contact, retrying')
129             time.sleep(1)
130             continue
131         else:
132             break
133
```

Figure 5 - Push data to database in .json object, format as shown

4.2 Android Studio Code Snippets (MainActivity.java, strings.xml, activity_main.xml)

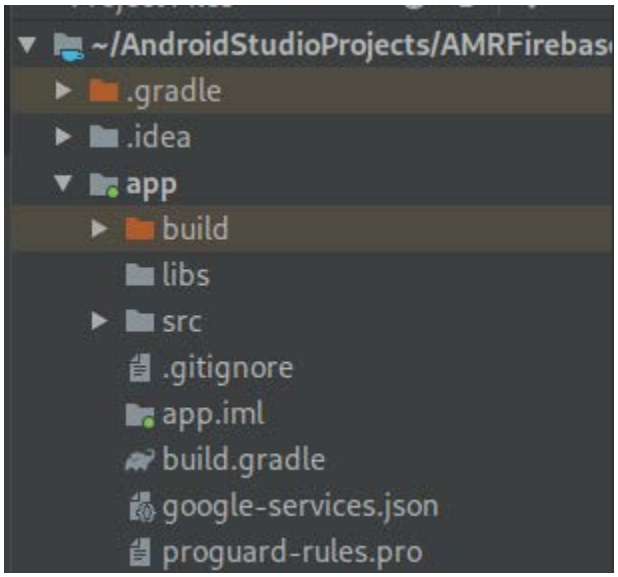

Figure 6 - google-services.json file in project directory

```xml
<resources>
    <string name="app_name">AMRFirebaseClient</string>
    <string name="datetime_label">Last Update:</string>
    <string name="datapoint_label_1">Accel. Calib:</string>
    <string name="datapoint_label_2">Gyro Calib:</string>
    <string name="datapoint_label_3">Heading:</string>
    <string name="datapoint_label_4">MagMom Calib:</string>
    <string name="datapoint_label_5">Pitch:</string>
    <string name="datapoint_label_6">Roll:</string>
    <string name="datapoint_label_7">Sys Calib:</string>
    <string name="no_datetime">No datetime</string>
    <string name="empty_data">Null</string>
    <string name="button_label">Get Data</string>
</resources>
```

Figure 7 - strings.xml, a file setup for explicitly declaring strings

```java
package com.example.amrfirebaseclient;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
```

Figure 8 - Import statements for MainActivity.java, allowing firebase access, UI management, and listener support

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        ////////////////////////////////////////////////////////////////
        // Begin Retrieval Code
        ////////////////////////////////////////////////////////////////

        //Declare TextView Objects
        final TextView datetime_obj = (TextView) findViewById(R.id.datetime_value);
        final TextView data_obj_1 = (TextView) findViewById(R.id.data_value_1);
        final TextView data_obj_2 = (TextView) findViewById(R.id.data_value_2);
        final TextView data_obj_3 = (TextView) findViewById(R.id.data_value_3);
        final TextView data_obj_4 = (TextView) findViewById(R.id.data_value_4);
        final TextView data_obj_5 = (TextView) findViewById(R.id.data_value_5);
        final TextView data_obj_6 = (TextView) findViewById(R.id.data_value_6);
        final TextView data_obj_7 = (TextView) findViewById(R.id.data_value_7);

        //Find the button and prepare it for actions
        Button ubutton_object = findViewById(R.id.update_button);
        ubutton_object.setOnClickListener((v) -> {
                //Add a little pop-up to indicate the user pressed the button
                Toast.makeText(getApplicationContext(),  text: "Updating Data...", Toast.LENGTH_SHORT)
                        .show();

                //Attempt to retrieve data from the firebase servers
                try {
                    FirebaseDatabase database = FirebaseDatabase.getInstance();
                    DatabaseReference database_reference = database.getReference();
                    database_reference.child("orientation sensor").addValueEventListener(new ValueEventListener() {
```

Figure 10 – MainActivity.java; UI setup, Button Click listener, and database object setup and initial retrieval

```
            //Method is invoked in the event that the database values change
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                String dd_1 = dataSnapshot.child("data").child("Accelerometer Calibration").getValue(String.class);
                String dd_2 = dataSnapshot.child("data").child("Gyroscope Calibration").getValue(String.class);
                String dd_3 = dataSnapshot.child("data").child("Heading").getValue(String.class);
                String dd_4 = dataSnapshot.child("data").child("Magnetometer Calibration").getValue(String.class);
                String dd_5 = dataSnapshot.child("data").child("Pitch").getValue(String.class);
                String dd_6 = dataSnapshot.child("data").child("Roll").getValue(String.class);
                String dd_7 = dataSnapshot.child("data").child("System Calibration").getValue(String.class);
                String last_update = dataSnapshot.child("data").child("last_update").getValue(String.class);

                //Now to change TextViews
                datetime_obj.setText(last_update);
                data_obj_1.setText(dd_1);
                data_obj_2.setText(dd_2);
                data_obj_3.setText(dd_3);
                data_obj_4.setText(dd_4);
                data_obj_5.setText(dd_5);
                data_obj_6.setText(dd_6);
                data_obj_7.setText(dd_7);
            }

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

            }
        });

    } catch (Exception e) {
        Toast.makeText(getApplicationContext(),  text: "Error when connecting to Firebase", Toast.LENGTH_LONG);
    }

});
```

Figure 9 – MainActivity.java; Database listener, updates TextView objects with data in real-time

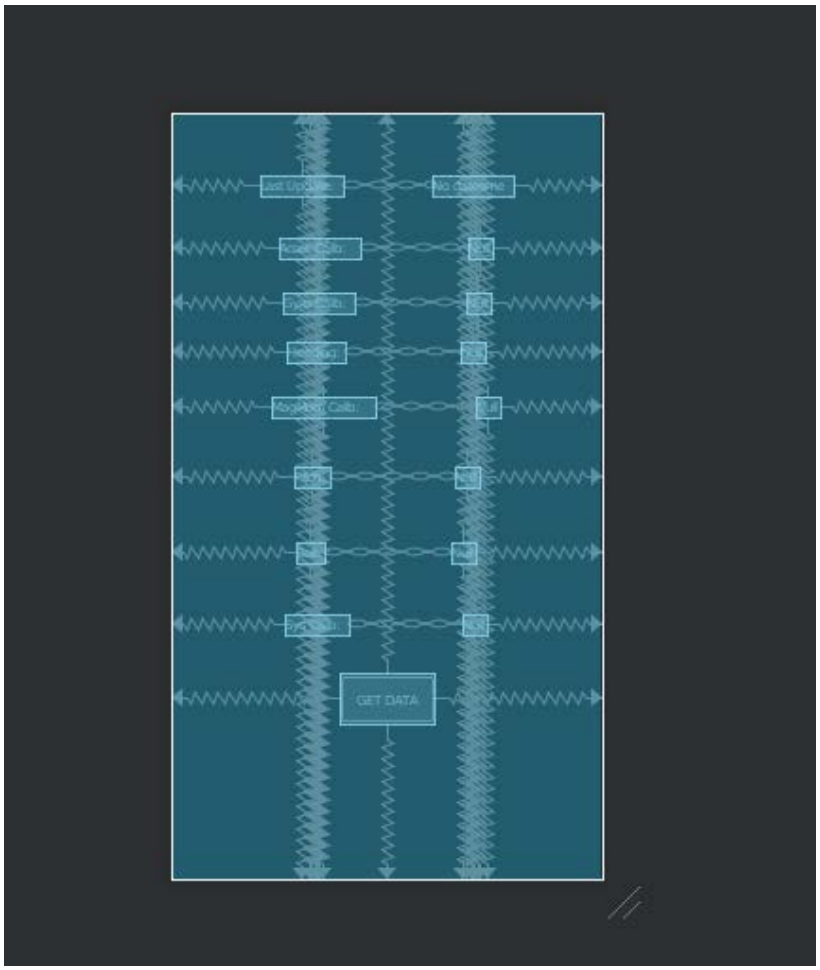

Figure 11 - UI setup in activity_main.xml, shows anchoring and chaining for TextView objects and Button object

4.0. STUBLING BLOCKS AND CONCLUSION

Throughout this project, several stumbling blocks were encountered. To begin with, a large portion of the project scope had to be adjusted. The original plan for the project was to send all of the orientation sensor information to the Android phone utilizing a combination of Robot Operating System (ROS) message passing on the Pi and a Bluetooth Low Energy (BLE) connection between the Pi and the Android device. However, after several hours of research, it became clear that attempting to achieve this, while possible, was going to take significantly longer than the time allotted for the class. Thus, in the final design, working with ROS was exchanged for the Python script and Adafruit libraries, and a Bluetooth server on the Android app was exchanged for the Firebase cloud database and API calls. This allowed the project goals to be accomplished within the given time constraints while maintaining the project objective.

However, the switch to Python was not without its difficulties, either. The python-firebase library used for sending data to the Firebase database was only functional on Python 2, and would not import correctly in Python 3. As such, an additional challenge of the project was ensuring that the Adafruit libraries for accessing the BNO055 sensor were also compatible with Python 2. Fortunately, they were, and the project was completed successfully. If the libraries hadn't been compatible, the project would have involved more research in finding libraries that could successfully run on Python 3.

Additionally, several portions of the Python script had unhandled exceptions. The BNO055 sensor, for example, took a few seconds to initialize, and the Adafruit libraries would throw exceptions if they weren't immediately able to access it. Then, on the other side, if the firebase library couldn't access the database on the first try (which is common with an HTTP connection), it too would throw an exception and return without retrying. To remedy this, both of these operations had to be placed in try/catch loops to provide the exception handling and timing management that was lacking in the default libraries. This is visible in Figure 4 and Figure 5 above.

One final stumbling block in this project was working with the Raspberry Pi on UW-Stout's network. The Pi was being given IP addresses with an unusually short lease time (i.e. 4 minutes) from the DHCP server, and would suddenly and unexpectedly change IP addresses. This would hinder development as the ssh session being used would suddenly drop out, and any unsaved work would be lost. No solution was implemented for this problem, although it is possible to solve it through a clever implementation of a symlink in the systemd service files pointing to the IP notification script and a tmux installation to hang the ssh session upon a disconnect.

In conclusion, despite the challenges faced, this project can be considered a success. As illustrated, data from the orientation sensor was retrieved from the sensor, sent from the Raspberry Pi and to the database, and pulled from the database into the Android app. Additionally, the Lidar sensor information was pulled from the Lidar sensor, displayed on the Pi, and subsequently forwarded to the PC. This achieves the goal of the project of viewing the data from both sensors on devices external to the Pi, and satisfies the requirements of the final project; utilizing two sensors and a wireless interface. This project was an excellent learning experience, and will provide useful background work for the Capstone project in the following semester.