The Following Robot

Senior Project: Electrical/Computer Engineering

January 2017- June 2017

Team Members:

Juan Cerda

Matthew Kwan

Vi Le

Faculty Advisor:

Bryan Mealy

Abstract

The objective of this project is to design, build, and test an autonomous robot with an associated Android application. The robot uses on board inertial measurement sensors (magnetometer, accelerometer, gyroscope) and coordinates itself through Bluetooth communication with the similar built-in measurement sensors on the Android phone to mimic and follow movement.
The Following Robot incorporates the same basic movement functionality as a typical RC car. The robot follows the user's phone through an application on one's phone. This application accesses the phone's accelerometer and gyroscope data and translates into appropriate conversions. Methods of tracking and calculating distance or angular displacement includes numeric integration. Once a certain turning angle or certain distance has reached a predefined threshold, the application sends a command to the robot via Bluetooth indicating which movement to execute: left, right, forward, or backward. The Following Robot has its own sensors to accurately match the threshold of the phone. Combining these two interfaces, the Following Robot can mimic the movements of the user hence the name "following".
Additionally, the application accesses its magnetometer to send the current direction and/or orientation of the phone. The robot uses this information and aligns its direction to match the phone with a click of a button on the application.

I. System Requirements

The Following Robot has four main blocks: Smartphone, Main, Sensors, and Rover. The smartphone should track the user's movements by reading and calculating raw data from built-in accelerometer, gyroscope, and magnetometer sensors. Then it should relay and send movement commands to the rover through the Bluetooth interface (UART communication protocol). Master microcontroller should be able to store multiple commands sent from the smartphone, execute commands in order, and send appropriate commands to sensor and rover microcontrollers through SPI communication protocol. The rover microcontroller should send control signals to two H-bridges to control two DC motors which make the rover to move forward and backward, turn left and right, and stop moving. The sensor microcontroller should be able to determine how much the rover should move, read raw data from accelerometer, gyroscope, and magnetometer sensors, keep calculating distance the rover has moved, and notify the master microcontroller when the rover finishes moving the desired distance. Also, rechargeable and high-capacity batteries should power the rover.

From the requirements described above, the complete system must feature an Android smartphone with built-in Bluetooth module and magnetometer, accelerometer, and gyroscope sensors and its application, three MSP430G2553 microcontrollers, a Bluetooth 4.0 module, two bidirectional DC motors, an H-Bridge, a 3.3-to-5V buffer, magnetometer, accelerometer, and gyroscope sensors, Buck converters, adjustable and fixed-5V linear voltage regulators, and high-capacity and rechargeable batteries.

Figure 1 below shows the high level block diagram of the Following Robot. See the appendices for test plans for each component and the system:

Appendix A: Full list of system requirements

Appendix B: Full list of test plans

Appendix C: Requirements Traceability Matrix

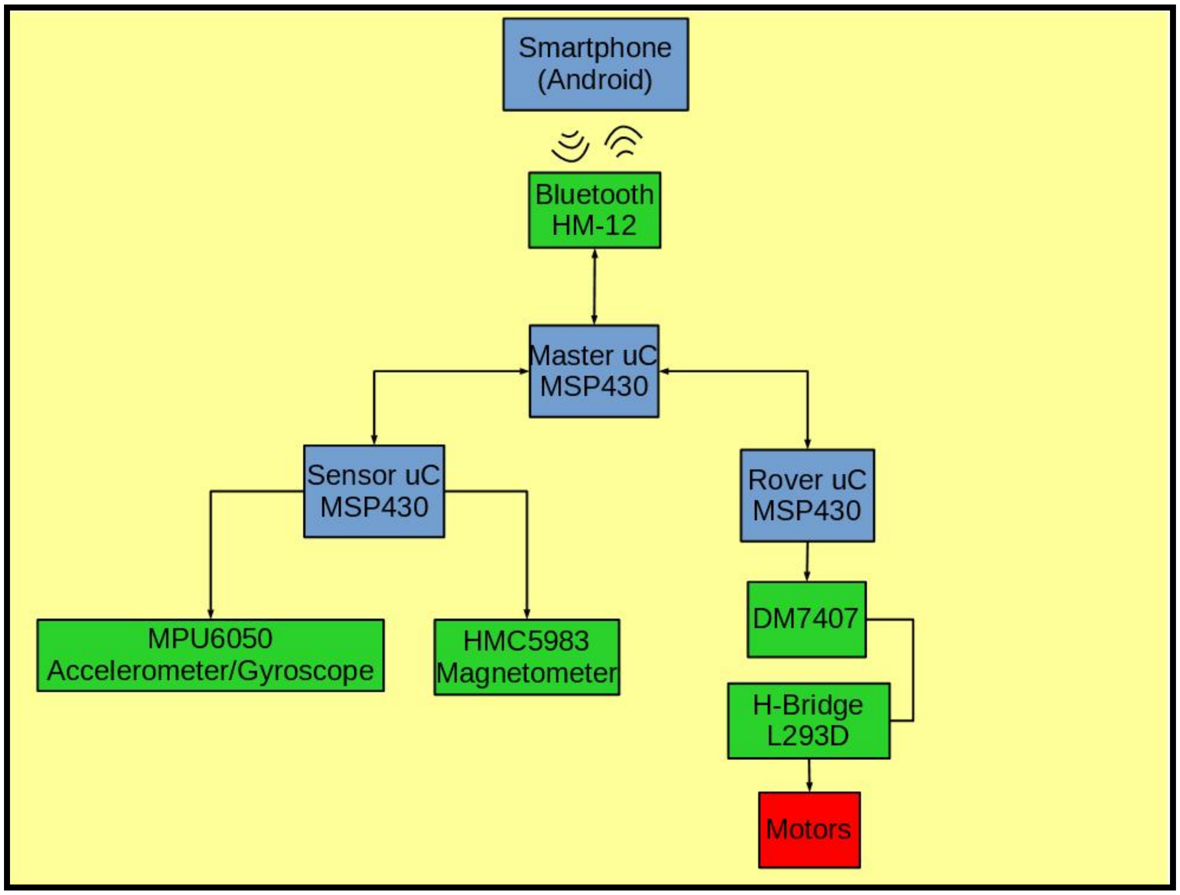


Figure 1. High Level Block Diagram of The Following Robot

II. System Specifications

Table 1. Specifications for each components in the system

MSP430G2553	**Description**	16-bit Microcontrollers - MCU
	Data Bus Width	16 bit
	Maximum Clock Frequency	16 MHz
	Program Memory Size	16 kB
	Data RAM Size	512 B
	Operating Supply Voltage	1.8 V to 3.6 V
	Package / Case	PDIP-20 Through Hole
	Data RAM Type	SRAM
	Interface Type	I2C, SPI, UART
	Number of I/Os	16 I/O pins
	Program Memory Type	Flash
HM-12	**Description**	Bluetooth Specification V4.0 EDR and BLE
	Serial port send/receive byte	90 max
	Working frequency	2.4 GHz ISM Band
	Sensitivity	≤ -84 dBm at 0.1% BER

	Modulation Method	Gaussian Frequency Shift Keying
	Security	Authentication and encryption
	Power	+3.3 V at 50 mA
	Coverage	Up to 60 meters
SN7407N	**Description**	Hex Buffers and Drivers With Open-Collector High-Voltage Outputs
	Low-level Output Voltage	0.7 V max
	High-level Output Current	0.25 mA max
	Low-to-high Propagation Delay	10 ns max
	HIgh-to-low Propagation Delay	30 ns max
L293D	**Description**	Quadruple Half-H Bridge
	Drivers Per Package	4
	Switching Voltage	36 V max
	Peak Output Current	1200 mA
	Delay Time	typical 36 ns
	Output Voltage	36 V max
	Input Compatibility	TTL
	Voltage at lowest Spec Current	typical 1200 mV
	Iout	600 mA max

DC Motors	**Description**	DC motors
	Max Voltage	12 V
	Gear Ratio	1/75
	No-Load Speed	11500 +/- 10% rpm
	No-Load Current	180 mA max
	Stall Current	4500 mA max
	Stall Torque	160 g.cm
	Humidity	30% ~ 95%
HMC5983	**Description**	3-Axis Digital Compass IC
	Supply Voltage Vdd	3.6 V max
	Average Current Draw	typical 100 uA is measurement mode
	Field Range (Full Scale)	-8 ~ +8 gauss
	Sensitivity (Gain)	230 - 1370 LSb/gauss at Vdd = 3.0V
	Digital Resolution	0.73 ~ 4.35 milli-gauss at Vdd = 3.0V, 1-LSb, 12-bit ADC
	Typical Noise Floor (Field Resolution)	2 milli-gauss at Vdd = 3.0V
	Measurement Period	6 ms from receiving command to data ready

	Output Rate	Continuous Meas Mode: 0.75 ~ 220 Hz Single Meas Mode: 160 Hz
	Turn-on Time	Ready for I2C commands: 200 us Analog Circuit Ready for Meas: 50 ms
	I2C Address	8-bit read address: 0x3D 8-bit write address: 0x3C
	I2C Clock Rate (max)	3400 kHz max
MPU-6050	**Description**	3 Axis Analog Gyroscope + 3 Axis Accelerometer Sensor
	Operating Supply Voltage	2.375 V ~ 3.46 V
	Normal Operating Current	typical 3.9 mA
	Start-up Time for Register Read/Write	100 ms max
	I2C Address	8-bit read address: 0xD3 8-bit write address: 0xD0
	I2C Clock Rate	Fast-mode: 400 kHz max Standard-mode: 100 kHz max
	Accel Data Acquisition Delay	19 ms
	Accelerometer Sensitivity	2g = 16384 LSB/(m/s^2)
	Gyro Data Acquisition Delay	0.98 ms

	Gyroscope Sensitivity	131 LSB/(°/seconds)
LM2596	**Description**	Adjustable Power Converter 150-kHz 3-A Step-Down Voltage Regulator
	Efficiency	typical 73%
	Output Leakage Current	50 uA max
	Oscillator Frequency	127 kHz ~ 173 kHz
	Saturation Voltage	1.5 V max
	Current Limit	6.9 A
	Duty Cycle	0% ~ 100%
LM317T	**Description**	Adjustable Linear Voltage Regulator
	Reference Voltage	1.2 V ~ 1.3 V
	Minimum Load Current	10 mA max
	Maximum Load Current	typical 2.2 A
	Output Voltage Range	1.2 to 37 V
	Line Regulation	0.07 %/V max
	Load Regulation	70 mV max for Vout < 5V 1.5% max for Vout ≥ 5V
	Adjustment Pin Current	100 uA max
L7805A	**Description**	5 V Positive Voltage Regulator

	Output Voltage	4.9 V ~ 5.1 V
	Output Current	1.5 A max
	Dropout Voltage	typical 2 V
	Line Regulation	50 mV max
	Load Regulation	100 mV max
Batteries	**Maximum Voltage**	3.7 V
	Maximum Capacity	3000 mAh
	Battery Size	Universal 18650
	Rechargeable?	Yes
	Number of Batteries Required	4

Table 2: Specifications for each communication protocol

Serial Peripheral Interface (SPI)	**Baud Rate**	1 Mb/s
	Clock Phase Select	Data is changed on the first UCLK edge and captured on the following edge.
	Clock Polarity Select	The inactive state is high.
	MSB First Select	MSB first
	Character Length	8-bit data

Inter-Integrated Circuit (I2C)	**Baud Rate**	400kb/s
	Slave Addressing Mode Select	Address slave with 7-bit address
Universal Asynchronous Receiver/Transmitter (UART)	**Baud Rate**	57600 bps
	Parity	Disabled
	MSB First Select	MSB first
	Character Length	8-bit data
	Stop Bit Select	One Stop Bit

III. System Architecture and Design

A. Smartphone Application

1. Overview

The purpose of the smartphone application is to track the user's movement, interpret the movement and relay this information to the robot. The application uses inertial measurement units already available in the phone's hardware and accessible through software using the Android API. The sensing units used include the accelerometer, gyroscope, and magnetometer. In order to transmit information to the robot the application uses bluetooth communications.

2. Development Tools/ Environment

Development IDE: Android Studio 2.3

Development Language: Java, XML

Main Android API’s used:

- SensorManager
    - TYPE_LINEAR_ACCELERATION
    - TYPE_ACCELEROMETER
    - TYPE_GYROPSCOPE
    - TYPE_MAGNETIC_FIELD
- BluetoothAdapter

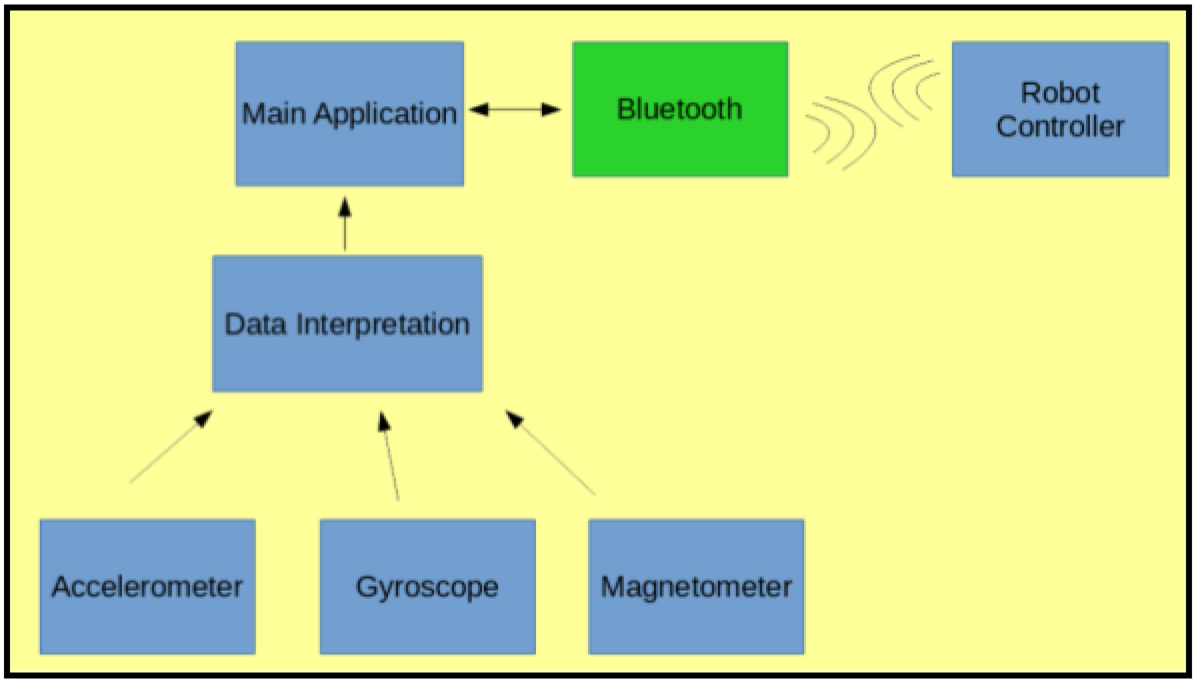


Figure 2. High Level Block Diagram of Android Application

3. Design Description

The main application contains instances of the SensorManager and BluetoothAdapter(fragment). All the sensor data readings and interpretations occurs within the Sensors class. For consistency and ease of implementation the sensor code uses the same trapezoidal integration as the microcontroller code (described in later sections) for displacement measurements (Pseudo Code 1). The calculations for magnetic field are also the same as within the microcontroller (Pseudo Code 3). From the measurements the application then determines the appropriate commands to provide to the robot. The same instance of the Bluetooth fragment as in the main application is used to transmit the movement commands.

Data Flow

Sensor data measurements are read and stored locally in the application. Values are stored and averaged, then parsed and interpreted after a predefined number of values has been reached. These values are used for calculations for displacement measurements. After a predefined displacement threshold is reached the values are then sent the the robot via Bluetooth communication. An instance of a Bluetooth “fragment” was sent to sensor instance and is used for communication. The GUI is then updated to reflect the latest measured values.

B. Master Block

The master microcontroller (MM) is responsible for receiving commands from the smartphone via Bluetooth communication. It then analyzes the commands and sends appropriate control commands to the rover and sensor microcontrollers which are, respectively, responsible for controlling the rover and processing data into displacement. The implementation of a queue - a data structure follows a "First In First Out" policy (as in the case of a normal queue when people stand in line at the counter), where the first element is pushed into the queue, or "enqueued," and the same element when it has to be removed from the queue is "dequeued" - the MM helps store all the commands sent from the smartphone, and the MM executes all commands in order. The MM will send a stop command (“XXX!”) to other microcontrollers when there is no command left in the queue.

Figure 3 below shows the flowchart of the MM. When powered on, the RM starts up by configuring the GPIO pins, UART and SPI master communication protocols and stays in the FLAG_IDLE state waiting for commands from the smartphone (UART communication protocol). When it receives a character, it stores that character in a receive buffer until it receives an exclamation (except for emergency stop command “X”) which notifies the end of the command. If the received message is not the emergency stop, the MM exits the UART receive interrupt subroutine (ISR), and goes to FLAG_USCI_A0_RX_FINISHED state. If the received message is the emergency stop, it sends the stop command to the other microcontrollers when it is still in the UART receive ISR, clears the queue and receive and transfer buffers, reset all the flags then goes back to FLAG_IDLE state.

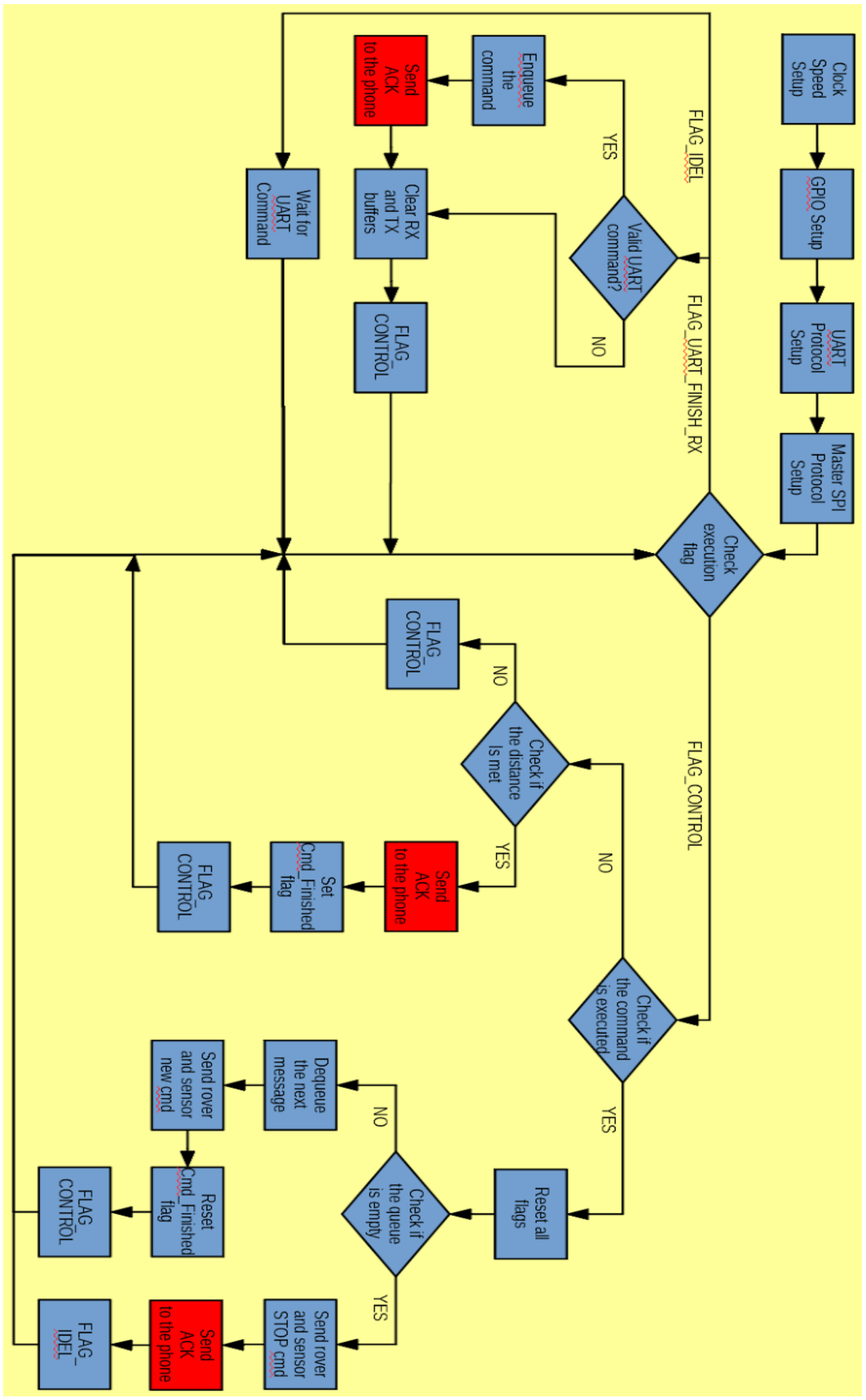


Figure 3: Main Microcontroller Flowchart

In FLAG_USCI_A0_RX_FINISHED state, the MM disables the UART receive interrupt and identifies and distinguishes the command received. While the rover is already moving, new rover commands, the MM adds the new command to the queue. If received commands are for debugging the system, the MM sends queue information back to device which requests for debugging. After that, the MM re-enables the UART interrupt then goes into FLAG_CONTROL state. Table 3 below shows a list of command of the MM receives from the smartphone and how it reacts.

Table 3: List of commands that the MM receives from the smartphone

Command	**Purpose**	**Meaning**	**Control command sent to the RM**	**Control command sent to the SM**
"W!"	Controlling	Command the rover to move forward	"WWW!"	"AAA!"
"A!"	Controlling	Command the rover to turn left	"AAA!"	"GGG!"
"S!"	Controlling	Command the rover to move backward	"SSS!"	"AAA!"
"D!"	Controlling	Command the rover to turn right	"DDD!"	"GGG!"
"E<xxx>!"	Controlling	Re-align the rover so it faces the same angle degree with	"DDD!"	"E<xxx>!"

		the smartphone <xxx>: The angle degree (in number) that the smartphone is currently facing.		
“X”	Controlling	Emergency stop	“XXX!”	“XXX!”
“Q!”	Debugging	Display general information about the implemented queue (number of not-yet-executed commands in queue, front index, rear index, commands at front index and rear index)	N/A	N/A
“P!”	Debugging	Display all not-yet-executed commands in the queue	N/A	N/A

In FLAG_CONTROL state, the MM executes in the following manner:

- If a command finishes executing, the MM resets all status flags and then checks if the queue is empty.
  - ➢ If the queue is not empty, the MM dequeues and analyzes a new command at the front of the queue. Then the MM first sends a control command to the RM and waits for a GPIO pin interrupt sent from the RM. the GPIO pin interrupt means that the RM receives and executes the command successfully. After receiving the interrupt, similarly the MM sends a control command to the SM and waits for a GPIO pin interrupt from the SM.

Then the MM stays in the FLAG_CONTROL state and waits for the new command to finish executing.

- ➢ If the queue is empty, the MM sends a stop command to the RM and SM and waits for acknowledgments from them. Then the MM stays in the FLAG_IDEL state and waits for new command sent from smartphone.

- If a command does not finishes executing, the MM waits for the command to finish by waiting for second GPIO interrupt sent from the SM. The second interrupt means that the rover finished moving the desired distance. The rover finishing its movement is determined by the fact that the SM keeps calculating the distance the rover has moved while the rover is moving.

Table 4 below shows a summary of what actions the MM performs when it gets into a specific state.

Table 4: *Summary of all states in the MM*

State	Actions
FLAG_IDLE	● Waits for new command from UART communication protocol
FLAG_USCI_A0_RX_FINISHED	● Disables the UART receive interrupt. ● Checks if the received message from the UART communication protocol is valid ● Performs appropriate actions depending on the message the MM receives (shown in Table 3). ● Clears receive buffer. ● Re-enables the UART receive interrupt. ● Next state is FLAG_CONTROL.

FLAG_CONTROL	● Checks if the previous command is finished executing. ● In this state, the MM receives the new command from the phone and goes to the FLAG_USCI_A0_RX_FINISHED state anytime. ● If the command is not finished yet, wait until it finishes. ● If the command is finished and the queue is empty, sends the stop command to the RM and the SM, waits for GPIO interrupts, and goes back to FLAG_IDLE state. ● If the command is finished and the queue is not empty, dequeues new command at the front of the queue, send control commands to the RM and SM, waits for GPIO interrupts, and stays in the FLAG_CONTROL state.

C. Sensor Block

The sensor microcontroller(SM) facilitates the data acquisition and processing from the MPU6050 and the HMC5983 or accelerometer/gyroscope and compass respectively via I2C protocol. The SM receives commands from the main controller via SPI communication to determine which sensor to execute. If the SM receives a valid command, it will send its first interrupt to the main microcontroller. Once the commands are executed the SM will continuously sample and process data until a predefined distance is met. Consequently, the SM will execute a second interrupt to the main microcontroller notifying it is finished. The figure below will visually represent the logic.

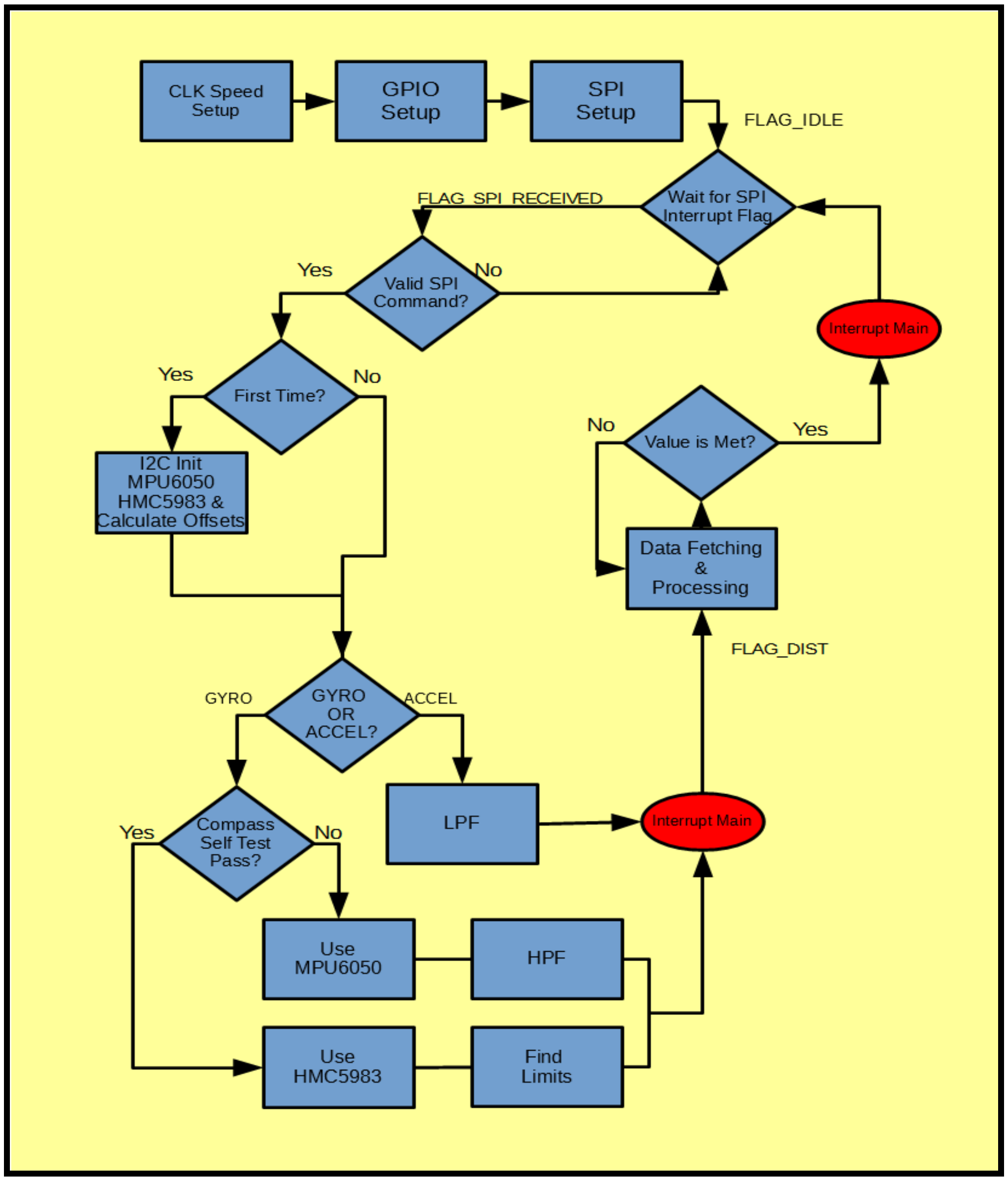


Figure 4. Sensor Block Flow chart(For more in-depth detail regarding this proceed to the Code Sequence and Table)

1. <u>MPU6050 Sensor Calculations</u>

The MSP430 reads MPU6050 data in 2's complement. Depending on the sensitivity/setting of the sensors, the corresponding scaling factor needs to be provided in the equation (See Table 2 above). One 'g' for accelerometer is equivalent to 9.8 m/s^2. So if an axis is pointed directly to the floor, theoretically, it should be giving the value 16384. In reality, this is not the case; this is why it is essential to perform calibration at the very beginning when operating the robot.

Double numeric-discrete data integration method is used to find total distance trave. In the pseudo-code below, the DATA_SIZE is the size of a data buffer with an arbitrary value of 50. The accelerometer data is sampled every 19 ms. This value will be important in distance calculations. The pseudo-code defines the 19 ms as TIME_SAMPLED.
One of the most accurate type of discrete data integration is trapezoidal.

Area of trapezoid = $0.5 \times Base \cdot Height$

In the figures below, refer data point number with a ‘D’ in front of it. (Ex. Point 8 is D8)

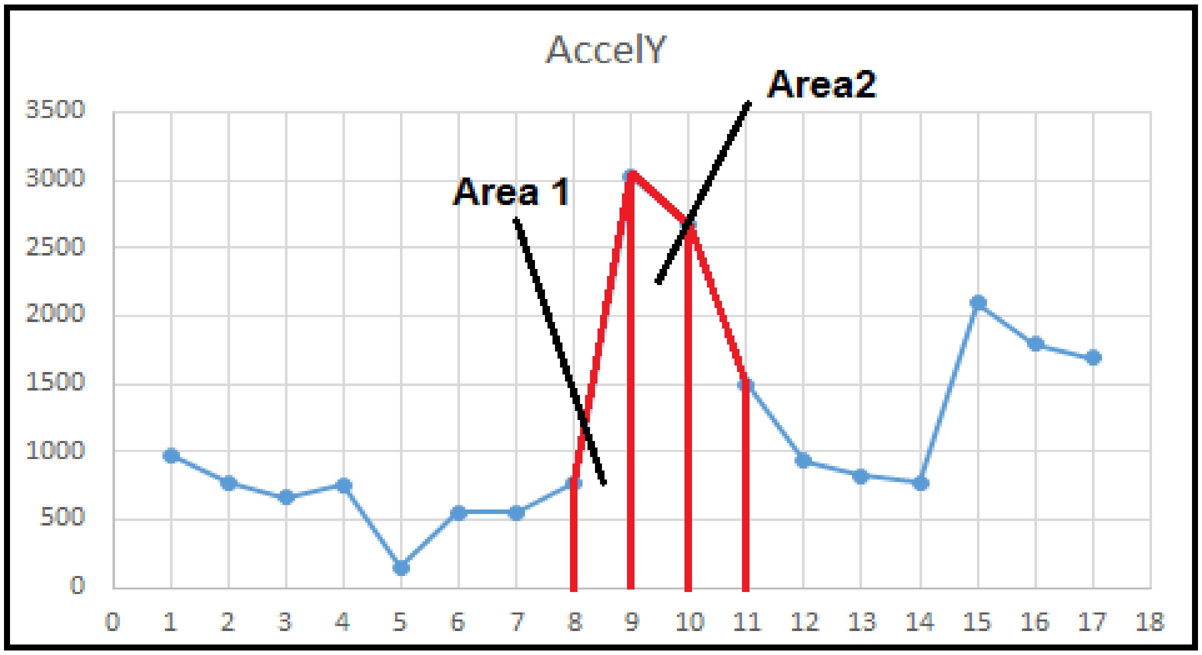


Figure 5(a). Accelerometer Y-axis Raw Data

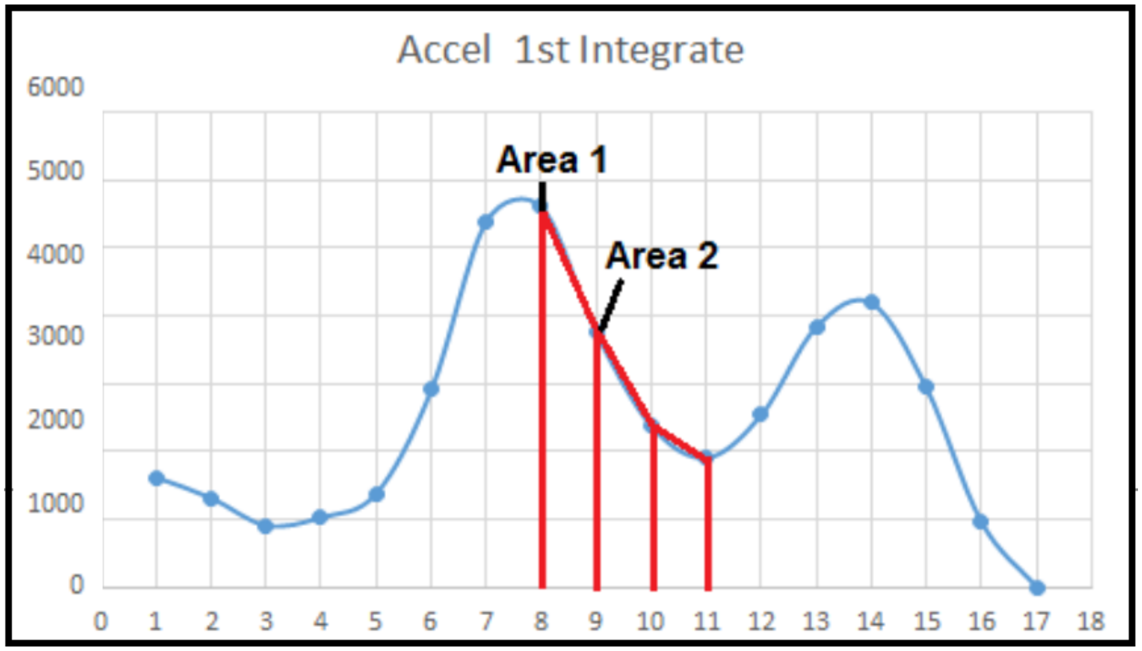


Figure 5(b). Accelerometer Y-axis First Integration

Figure 5(a) shows the raw data of the accelerometer. The red lines dictates the trapezoidal approximation of the values. Assuming the height for the trapezoid is one, Area 1 and Area 2 is represented by:

$$Area1 = 0.5 \times (D8 + D9)$$

$$Area\,2 = 0.5 \times (D9 + D10)$$

Figure 5(b) uses the "Area 1" and "Area 2" as the new data points. Using these points, another integration is necessary. The following new area will be:

$$New\ Area\ 1 = 0.5 \times (Area\ 1 + Area\ 2)$$

Substituting "Area 1" and "Area 2" to the above equations and replace integers with n, the final representation for a single sample double-numeric integration is:

$$\boxed{0.25 \times (D[n] + 2 \times D[n+1] + D[n+2])}$$

Formula 1. Single Sample Double-Numeric Integration Calculation

Note that this is merely a general notation for calculating distance from acceleration values. In order for Formula X to work, the value needs to be summed up (every 50 samples in this case) need to be scaled by the accelerometer sensitivity, the sample time, and the value of gravity. The "offset" pseudo code above is the calibration process which normalizes the raw data values of the accelerometer to zero.

```
for (i = 0; i < DATA_SIZE - 2; i++) {

       DISTANCE += D[i] + 2*D[i + 1]+ D[i + 2] - 4*offset;

}


FINAL_DISTANCE = DISTANCE / (4.0 * 16384) * 9.8 * TIME_SAMPLED_ACCEL;
```

Pseudo-Code 1. Accelerometer Displacement Calculation

`*NOTE: Each data point has absolute values. For the SM only displacement is necessary since direction is not very important.`

The gyroscope calculations are very similar, however only a single numeric integration is needed.

```
for (i = 0; i < DATA_SIZE - 2; i++) {
       ANGLE+= D[i] + D[i + 1] - 2*offset;
}


FINAL_ANGLE = ANGLE/ (2.0 * 131) * TIME_SAMPLED_GYRO;
```

Pseudo-Code 2. Gyro Angle Displacement Calculation

2. Magnetometer Sensor Calculations

Since the magnetometer is highly affected by magnetic interference, the gyroscope can act as a backup if the magnetometer is not reliable. The check to determine whether or not the magnetometer is feasible is not yet implemented. One possible way to implement in the future is to interface an iron detector to measure amounts of iron nearby.

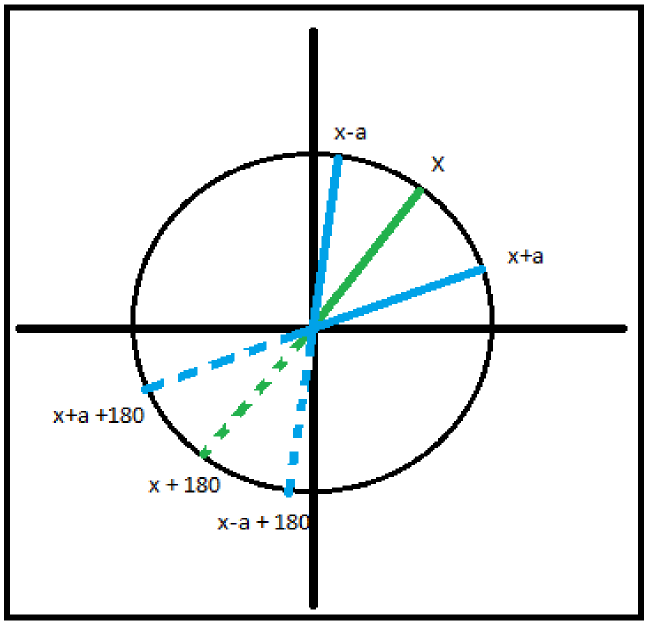
Since the magnetometer returns values in Teslas, a conversion to degrees is necessary. The documentation is found by Honeywell [1].

```
int HMC5983_Direction()

{

    int degree = atan(z_val * 1.0 / z_coor) * 180 / 3.14;

    if (z_val < 0)

    {

        direction = 270 - degree;

    }

    else if (z_val > 0)

    {

        direction = 90 - degree;

    }

    else

    {

        if (x_val < 0)

        {

            direction = 180.0;

        }

        else

        {

            direction = 0.0;

        }

    }

 }
```

Pseudo-Code 3. Compass Translating Teslas to Degrees

In order to correctly track the displacement of the compass there are multiple cases needed to account for. ‘X’ Is the referenced angle. ‘a’ is the pre-defined threshold angle. The 180° offset will ensure that there will be no problems handling boundary cases. Once ‘X’ is returned, the lower and upper bounds needed to be set in order to set boundaries for the robot to move (x+a, x-a). And of course, the values are shifted by 180. ***Note: Cases 1 & 2 occurs only once at the beginning of every sent command from phone to the robot. In other words, Cases 1 & 2 are essentially finding the current reference. Also the top axis in the figures below will be the 0º/360º boundary.***

Case 1: The only arithmetic needed to be done is to add a 180° offset to the sampled angle along with the boundaries.



Pseudo-Code 4. Compass Translating Teslas to Degrees Case 1

Case 2:

a) Referring to the figure in the left below, when the lower bound, “x-a”, is less than 0°, the value needs to be adjusted so that the lower limit is between 0° and 360°. Notice in this case that the lower bound subtracts 180° instead of adding in case 1.

b) Referring to the figure in the right below, when the upper bound, “x+a”, is greater than 360°, the value is corrected to “x+a - 180”.

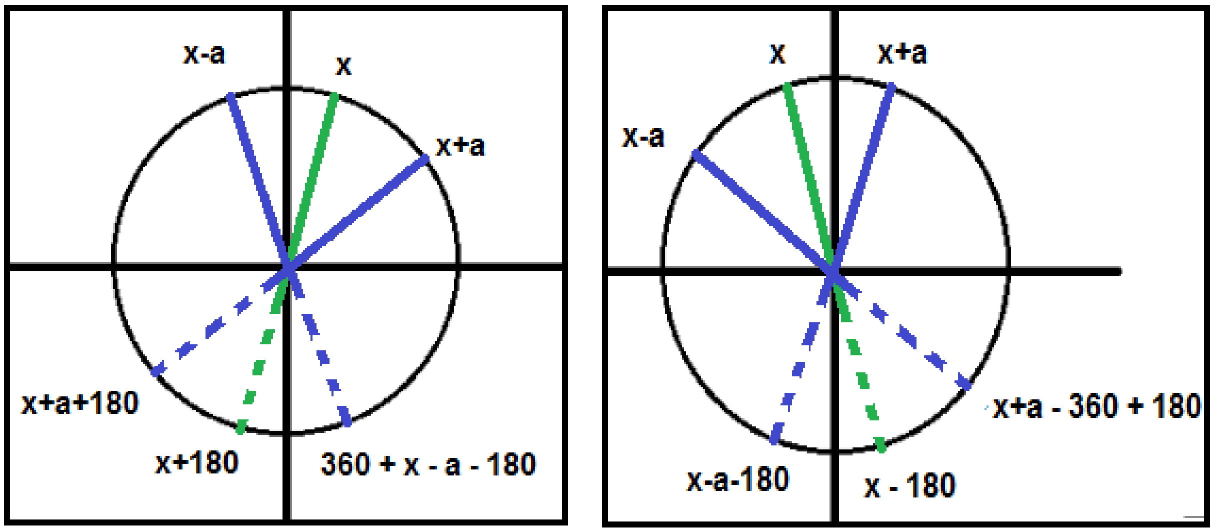


Figure 5. Compass Translating Teslas to Degrees Case 2 [Left (a), Right (b)]

Case 3: This is a special case when the robot finished its reference checking (Case 1 & Case 2). While the robot is turning, the robot will update itself until its newly sampled angle is out of bounds. From the figure above (left), if the new 'x' angle turns counterclockwise and passes the 0°/360° boundary, the "x+180" will not work anymore because the newly sampled angle will be in the top left quadrant range. Adding 180° will result in an angle that is >360°. To accommodate for this case, if the difference between the new angle and reference angle is larger than a predefined threshold(Appendix Code is 324), the new angle will be offsetted by -180° instead of +180°. The similar method will be applied to the right figure above.

```
void compass_reference_find()

{

  int upper_bound = Reference_angle + Angle_Threshold;

  int lower_bound = Reference_angle - Angle_Threshold;


  if(upper_bound  > 360)

  {

      Final_upper_bound = upper_bound - 360 + 180;

      Final_lower_bound = lower_bound - 180;

      Case 2b

  }

  else if(lower_bound  < 0)

  {

      Final_upper_bound = upper_bound  + 180;

      Final_lower_bound = 360 + lower_bound - 180;

      Case 2a

  }

  else

  {

      Final_upper_bound = upper_bound;

      Final_lower_bound = lower_bound;

      Case 1

  }

}
```

Pseudo-Code 6. Magnetometer Finding Reference and Boundaries for Case 1 & 2

3. Sensor Block Code

The following table will go over the three main files regarding the sensor block. Not all functions will be listed.

Table 5. Code Explanation for Sensor Block

File Name	Function/Flags	Description
main.c	`main_Initialize_All`	● Wakes the MPU6050 from sleep mode and initializes the HMC5983 ● Finds offsets of accelerometer and gyro
	`FLAG_IDLE`	● Idle State/Waits for SPI command
	`FLAG_SPI_FINISH_RX`	● Once SM receives something from MM via SPI, this flag will check whether that something is a valid command ● If valid, checks what kind of a command it is and sets necessary filters. If magnetometer, this flag will find reference angle in "`LPF_HPF_MPU6050_setting()`" ● Upon finishing, the MC will receive an interrupt and proceed to `FLAG_DIST` ● If invalid, all flags will be reset and

		return to `FLAG_IDLE`
	`FLAG_DIST`	● Does all the data processing of MPU6050 or HMC5983
	`dist_met`	● Notifying data sampling is done and proceed to send interrupt to MM
MPU6050.h	`LPF_HPF_MPU6050_setting`	● Sets the filters for MPU6050 or HMC5983 ● MPU6050 needs to write to two different registers in order to set the LPF/HPF ● If magnetometer/compass, the boundaries and reference angle will be set here. ● If the compass is within +/- 10 degrees of angle sent by MC, the `dist_met` will be set to true
	`data_request`	● The flag that determines what kind of sensor should be used
	`SPI_flag`	● Will return true if valid command ● Sets all the `data_request` depending on the SPI data ● Sets the predefined thresholds

	`MPU6050_data_loop`	● Loops through data depending on `data_request` ● Checks if is `dist_met` above a threshold (ignoring noise)
	`read_accel_gyro_MPU6050`	● Will constantly write/read from MPU6050 ● Accelerometer needs ~304k delay cycles to sample at 19ms ● Gyroscope samples at 15.68k delay cycles to sample at 0.98 ms ● Not 'X' is not used currently
	`long_averaging/int_averaging`	● Does the numeric integration for accel/gyro respectively
	`Accel_Gyro_data_Collection`	● Scales the data by necessary factors
HMC5983.h	`HMC5983_Direction`	● Converts RAW values int degrees
	`compass_default_settings`	● Sets the upper_bound, lower_bound of compass and returns an integer to notify which case did it execute
	`HMC5983_Total_Angle`	● Accounts for "Case 3" in the compass section

Code Sequence

1. On power on, the SM begins with the standard I2C and SPI setup protocol as well as the GPIO configurations. The SM will then initialize its slave boards--the sensors. The LED will blink quickly for about a second indicating that it is done with initialization which proceeds to the FLAG_IDEL state
2. The SM will stay in FLAG_IDEL until a command is received from the SPI interrupt. Once a command is received, the SM's state is changed to FLAG_SPI_FINISH_RX. Two things happen during this state: a valid command or an invalid command.
3. Upon receiving an invalid command, the SM will reset all its flags and buffers and change its state back to FLAG_IDEL. If a valid command is received, the SM will check which sensor to access. The commands received in the SM will be very similar to the commands in the MM.(Refer to the table in the MM section for more information)
4. When the commands are readily distinguished, the appropriate initialization and calibration are executed. An interrupt will be sent to the MM. After this, the SM's state is changed to FLAG_DIST.
5. FLAG_DIST performs all of the data processing and conditioning. It samples data from the correct sensor and stores in the global variable, "distance_traveled". This global variable will be compared to a predefined threshold depending on the sensor setting and see if it is within the tolerance range. Once it is met, the "dist_met" global variable will be set to TRUE. This will then proceed to interrupt the MM notifying it is finished with its role. The SM's state will be changed back to FLAG_IDEL.

D. Rover Block

The rover microcontroller (RM) is responsible for controlling two motors, so that the rover can either go forward, backward, turn left, turn right, or stop by sending control signals to the L293D H-bridges. Since the RM operates at 3.3V but the H-bridges operates at 5V, after receiving commands from the MM, the RM send control signals to the SN7407N buffer which converts 3.3V to 5V signal level before the control signals go to the H-Bridges. The left side of the H-bridge controls the right motor, and the its right side controls the left motor.

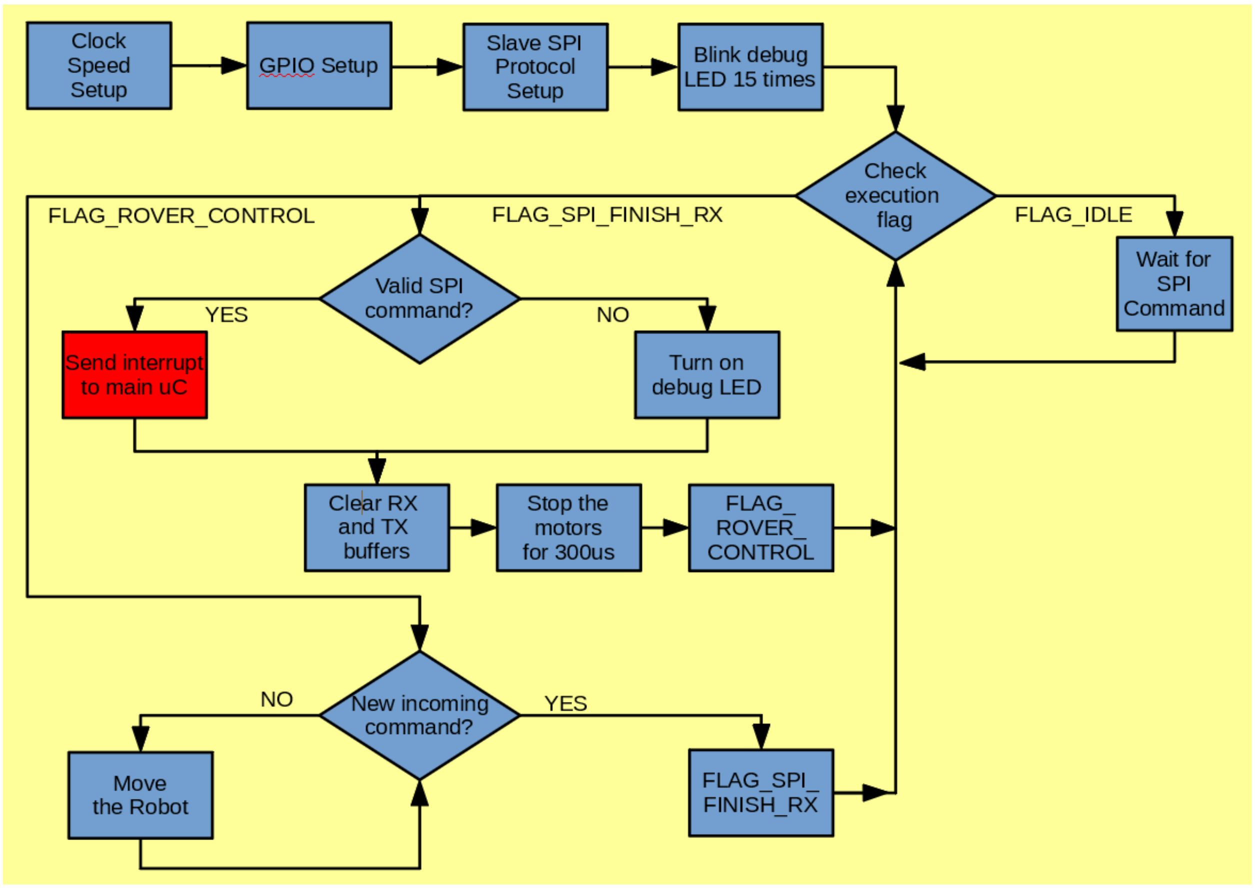


Figure 6. Rover Microcontroller Flowchart

Figure 6 above shows the flowchart of the RM. When powered on, the RM configures the GPIO pins, SPI slave communication protocol which is then followed by blinking the LED multiple times to notify that it is ready to receive commands from the MM.
Initially, the RM is in the FLAG_IDLE state. It waits for new command sent from the MM. SImilarly to the SM, when there is an SPI receive interrupt from the MM, the RM stores the received character in a receive buffer until it receives an exclamation “!” which notifies the end of the message. Then the RM goes into FLAG_SPI_FINISH_RX state.
In the FLAG_SPI_FINISH_RX state, the RM initially disables the SPI receive interrupt then checks whether the received command is valid or not. If the command is invalid, the RM turns on the debug LED. If the command is valid, the RM sends an interrupt signal through one of its GPIO pins to the MM. Next, it clears the receive buffer and stops the motors for 300us before re-enabling the SPI receive interrupt and proceeding to the FLAG_ROVER_CONTROL state.
In FLAG_ROVER_CONTROL state, the rover not only sends control signals to the H-Bridges to drive the motor but also stores future commands received from the MM in a queue. Table 6 below shows required input states of the H-bridge to command the rover to move. When it receives new command from the MM, the RM gets back to the FLAG_SPI_FINISH_RX state to analyze the command.

Table 6. Input states of Quadruple Half-H Driver pins

Right motor				**Left motor**				**Rover's Motion**
1, 2EN	**1A**	**2A**	**Status**	**3, 4EN**	**3A**	**4A**	**Status**	
H	H	L	Rotate clockwise	H	H	L	Rotate counter-clockwise	Move forward
H	L	H	Rotate counter-clockwise	H	L	H	Rotate clockwise	Move backward
H	H	L	Rotate counter-clockwise	L	X	X	Rotate counter-clockwise	Turn left
L	X	X	Rotate clockwise	H	H	L	Rotate clockwise	Turn right
L	X	X	Standstill	L	X	X	Standstill	Stand still
Note: L = low, H = high, X = don't care								

Table 7 below shows a summary of what actions the RM performs when it gets into a specific state.

Table 7: Summary of all states in the RM

State	Actions
FLAG_IDLE	● Initially the RM is in this state. ● Wait for new command from SPI communication protocol. ● Next state is FLAG_SPI_FINISH_RX.
FLAG_SPI_FINISH_RX	● Disable the SPI receive interrupt ● Check if the received command is valid. ● Send a GPIO interrupt to the MM if the command is valid. ● Turn on debug LED if the command is invalid. ● Clear receive buffer. ● Stop the rover for 300 us. ● Re-enable the SPI receive interrupt. ● Next state is FLAG_ROVER_CONTROL.
FLAG_ROVER_CONTROL	● Move the rover according to the command. ● Wait for new command from SPI communication protocol. ● Get back to the FLAG_SPI_FINISH_RX if it receives a new command.

V. Bill of Materials

Table 8. Bill of Materials

Part Description	Distributor	Manuf.	Quantity	Unit Price	Total Price
MSP430G2553 Mixed Signal Microcontrollers	Amazon	Texas Instruments	3	$5	$15
LM2596 DC to DC Buck Converter 3.0-40V to 1.5-35V Power Supply Step Down Module	Amazon	eBoot	2	$5	$10
L7805A Positive Voltage Regulator ICs Output 5v TO-220 Package	Amazon	STMicroelectronics	1	$3	$3
LM317T Adjustable Voltage Regulator IC 1.2 V to 37 V 1.5 A	Amazon	STMicroelectronics	1	$3	$3
L293D Dual H_Bridge Motor Driver	Amazon	Texas Instruments	2	$6	$12
SN7407N Hex Buffer Driver Open Collector DIP-14	Amazon	Texas Instruments	1	$5	$5
HM-12 Serial Bluetooth 4.0 BLE&EDR Dual Mode Module	Amazon	TinySine	1	$15	$15
GY-521 MPU-6050 Module 3 Axis	Amazon	Kootek	1	$7	$7

analog gyro sensors+ 3 Axis Accelerometer Module					
HMC5983 Sensor Temperature Compensation 3-Axial Compass SPI Board I2C Module	Amazon	UCTRONICS	1	$8	$8
Breadboard Jumper Wires Ribbon Cables Kit, 120 pcs	Amazon	Boundto	1	$10	$10
18650 Battery Storage Case Plastic Box Holder Leads With 4 Slots for 6" Wire Leads	Amazon	SACKORANGE	1	$8	$8
18650 Battery Lithium-ion 3000mAh 3.7V Low Self Discharge Rechargeable Batteries, 4 Counts	Amazon	EBL	1	$11	$11
Light-emitting Diode (LED)	N/A	N/A	4	$1	$4
KOOKYE Robot Smart Car Platform Metal Stainless Steel Chassis Speed Encoder Motor 9V with Crawler	Amazon	KOOKYE	1	$100	$100
10 pcs 40 pin 2.54 mm Straight SIngle Row Pin Header Strip	Amazon	OdiySurveil	1	$6	$6
5 pcs 2 Position SPDT Self Locking Toggle Switch, AC 250 V/3 A	Amazon	Uxcell	1	$10	$10

5pcs 6x8 cm Double-Sided Prototype PCB Universal Printed Circuit Board	Amazon	Vktech	1	$7	$7
Resistors and Capacitors	Cal Poly IEEE	N/A	22	$0.5	$11
Miscellaneous (spacers, screws and nuts, solders, wires, etc.)	N/A	N/A	N/A	$20	$20
Total Price					**$262**

VI. Design Verification and Testing

Issues Faced:

Once the majority of implementation and coding was complete, the team proceeded to the testing phase of the project.

An important first step for testing was a determine the appropriate calibration factors for both the phone and the sensors on the robot. The teams needed to determined offset values for the resting state and zero those values out. The next step was to live testing and accommodate for any drift that would accumulate over time. There was some difficulty faced with getting the robot to remain a truely straight path since differences in the current drawn by the motors and slight mechanical inconsistencies in the wheels would cause the robot's path to deviate over time. The team also needed to calibrate for the differences between the phone's sensors and the robot's.

VII. Result and Conclusions

As implementation and testing proceeded the team encountered limitations in both memory and processing power of the microcontrollers (MSP430G2553). The project required 3 microcontrollers communicating in a coordinated manner while processing sensor data. The memory limitation required the group to write short, concise code with very little floating point/ mathematical operations. A faster processor with more memory and more GPIO pins such as a Raspberry Pi could resolve some of these issues.

VIII. References

[1] Allen, Leroy D. "Constant Compass Heading For Great Circle Navigation And The N-1 Compass System*." *Navigation* 3.9 (1953): 325-33. Web.

[2] Texas Instruments, “MSP430G2x53 MSP430G2x13 Mixed Signal Microcontroller,” MSP430G2553 datasheet, Apr. 2011 [Revised Feb. 2013].

[3] Texas Instruments, “MSP430x2xx Family User's Guide,” MSP430G2553 User Guide, Dec. 2004 [Revised Jan. 2012].

[4] Texas Instruments, “L293x Quadruple Half-H Drivers,” L293D datasheet, Sep. 1986 [Revised Jan. 2016].

[5] Texas Instruments, “SNx407 and SNx417 Hex Buffers and Drivers With Open-Collector High-Voltage Outputs,” SN7407N datasheet, Dec. 1983 [Revised Sep. 2016].

[6] Texas Instruments, “LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator,” LM2596D datasheet, Nov. 1999 [Revised May 2016].

[7] TinySine, “Serial Bluetooth 4.0 Smart Ready dual-mode module,” HM-12 User Manual, 2014

[8] Honeywell, “3-Axis Digital Compass IC HMC5983,” HMC5983 datasheet, Jan. 2012.

[9] InvenSense, “MPU-6000 and MPU-6050 Product Specification,” MPU-6050 datasheet, Nov. 2010 [Revised Aug. 2013].

[10] STMicroelectronics, “L78 Positive Voltage Regulator ICs,” L7805T datasheet, Jun. 2004 [Revised Nov. 2016].

[11] STMicroelectronics, “LM217, LM317 1.2 V to 37 V Adjustable Voltage Regulators,” LM317T datasheet, Sep. 2004 [Revised Mar. 2014].

[12] Amber. "KOOKYE Robot Tank Car Chassis with Motor Installation Instructions." *Kookyecom.* KOOKYE, 18 Nov. 2016. Web. 09 Jan. 2017.

[13]Semiconductor, Inc. Freescale. AN3397, Implementing Positioning Algorithms Using Accelerometers - Application Notes (n.d.): n. pag. Web.

IX. <u>Appendices</u>

Appendix A: System Requirements

1. The Bluetooth module shall be capable of transmitting and receiving data and operate at 57600 baud rate
2. The motors shall provide bidirectional rotation.
3. In the MM, the implemented queue should enqueue new command and dequeue command at the front of the queue properly.
4. The MM should be able to send new control commands to the SM and the RM through the SPI communication protocol.
5. The MM must execute in the correct of order
6. The SM and the RM shall be able to receive new control commands from the MM through the SPI communication protocol.
7. The SM shall be able to read raw data from the accelerometer/gyroscope and magnetometer sensors from I2C communication protocol.
8. The SM shall calculate linear displacement the rover is moving forward/backward and angular displacement the rover turn left/right. The result of the calculation should be with 10% of the desired distance/angle.
9. The RM shall be able to send control signals to the H-bridges to control the motors.
10. The smartphone application shall feature a command to connect the PC's Bluetooth stream to the Bluetooth module.
11. The application shall be able to send/receive 8-bit commands to/from the Bluetooth module via the smartphone's Bluetooth stream.

12. The application shall be able to read raw data from the built-in accelerometer, gyroscope, and magnetometer sensors.
13. The smartphone application shall calculate linear displacement the user is moving forward/backward and angular displacement the user turn left/right. The result of the calculation should be with 10% of the desired distance/angle.

Appendix B: Test Plans

<u>TC1: UART Communication Protocol Test (or Bluetooth Interface Test)</u>

To verify that the Bluetooth module can connect to the smartphone, press the “Connect” button in the GUI. Observe that the LED on the Bluetooth module goes from blinking to solid, and that the GUI displays “Connected!” on the phone screen.

To test whether the MM receives the correct message sent from the smartphone, run the MM in debug mode and place a breakpoint at the line of code where it stores the received character from the UART communication protocol in a receive buffer. The smartphone sends each individual character of the whole command, and the debugger mode should display that character. After the smartphone sends an exclamation, check the receive buffer to make sure it stores the entire command.

To test whether smartphone receives the correct message sent from the MM, the smartphone’s GUI should display the entire message.

<u>TC2: SPI Communication Protocol Test</u>

This test requires two computer connecting to two MSP430 development boards: One computer controls the SPI master and the other controls the SPI slave. Set up the SPI master and slave such that they have similar configurations (baud rate, acknowledgements, etc.). Run two microcontrollers in debug mode, and scope clock and master-out-slave-in (MOSI) signal pins. Also on the computer running the SPI slave, place a breakpoint at the line of code where the debugger displays the received character. Transmit any character from the the master and observe the corresponding pulses on the oscilloscope screen. Check the slave to make sure it receives the correct character. After that, remove all breakpoints, send the entire message from the master and check if the slave receives the complete message.

TC3: I2C Communication Protocol Test

After setting up the I2C communication protocol, the microcontroller sends a configuration message to the Configuration Register A (register address: 0x00) of the HMC5983 sensor. Wait for 67 ms then read values stored in the Configuration Register A and compare with the message sent previously.

Next, read the Data Output X Registers A and B, Data Output Y Registers A and B, and Data Output Z Registers A and B located in registers 0x03 through 0x08.

TC4: Communication System Test

This test requires three microcontrollers: One connects to the MM, one connects to the SM, and one connects to the RM. All computers run the microcontrollers in debug mode. On the MM, place a breakpoint at the line of code where the debugger displays entire received message from the smartphone. On the SM and the RM, place a breakpoint at the lines of code where the debugger displays the entire message sent from the MM. The test starts by letting the smartphone send a turning-right message “D!” to the master through UART communication protocol. Check if the receive buffer of the MM stores “D” message. Then resume the MM and check if the SM and RM, respectively, receive “GGG!” and “DDD!” messages through SPI communication protocol. Then resume the SM and check if the HMC5983 sends back raw data to the SM through I2C communication protocol.

TC5: Motor and Motor Control Tests

To test motors, a 50% duty cycle 10V peak-to-peak square wave with a period of 10 seconds was supplied to make sure the motors could rotate in both directions.

Next, the quadruple half H driver was connected to two motors and the MSP430 development board as described above. The control of two motors was tested according to Table 6 above by letting the MM sends commands to the RM.

TC6: Live Movement Test

To verify that the live movement sequence is working properly connect the phone application to the robot via Bluetooth, then press “FORWARD(W!)” and observe that the car moves forward. Press “LEFT(A!)” and observe that the car turns left. Press “BACK(S!)” and observe that the car moves backward. Press “RIGHT(D!)” and observe that the car turns right. Then press the “STOP(X!)” button in the GUI and observe that the rover stops moving.

TC7: Sensor Reading and Calculation Test

Smartphone Application

Linear and Angular Displacement:

To test linear displacement use a reference position displacement measurement in centimeters. First outline reference distance simply by using ruler and two points with a known distance apart. Then move the phone across the two points and measure the distance displayed by the phone and record it, repeat the test, average and compare to the specified tolerances. A similar process is used to measure angular displacement where displacement is in degrees between two points is measured and recorded.

Magnetic Field Readings:

To test the measurements open up the phone application, and verify that the “angle” displayed makes sense, it should be within 0 - 360 degrees. Place the phone on a flat surface and rotate along the xy plane. The values should change in accordance to angular movement, but remain within the set boundaries.

Sensor Microcontroller

MPU6050

The accelerometer of the MPU6050 needs to be tested independently. This is done by breadboarding the microcontroller with the MPU6050 and connected to the computer in Code Compose for debugging. From there, the breadboard is moved along the axis of interest(the robot uses the y-axis) with a ruler by its side to measure the displacement. After the breadboard moves to pre-defined threshold specified in the code, the breakpoint where “dist_met” flag is set to true should be hit(see more in the sensor microcontroller section for more information). In this case, the predefined threshold in the code is 60 cm. Once the breakpoint is hit, the breadboard should travel approximately +/- 10cm from 60cm. The reason for this large error is due to the slow processing within the debugger. In reality 10cm is not a lot when compared to walking pace.

The gyroscope on the MPU6050 is tested nearly the same as the accelerometer. Instead of moving the breadboard along the axis of interest, the breadboard will be placed on a 360° turn-around table. Below the turn-around, important angles are marked. Once the angle moves the certain pre-defined threshold (25° in this case), the breakpoint in Code Composer will be hit at the “dist_met” flag. After testing multiple times, moving at a slow to medium speed gave the most accurate results. This makes sense because moving very fast will cause a lot of inertial error especially since the debugger is slow. Moving at a slow to medium speed results in an error of +/- 8°

HMC5983

The compass on is tested the exact same way as the gyroscope. As expected, the compass is much more accurate than the gyroscope with only a small error of +/- 2°. The compass seems to be the most accurate outdoors.

<u>TC8: MM Execution and Queue Test</u>

Every time it does something, the MM sends a defined message back to the smartphone. Table 9 below shows messages the MM receives from the smartphone, its actions, and messages it sends back to the smartphone.

Table 9. Messages the M receives from and sends to the smartphone

Message received from the smartphone	MM's Action	Message sent back to the smartphone
"Q!"	Display general information about the implemented queue (number of not-yet-executed commands in queue, front index, rear index, commands at front index and rear index)	"<number_of item_in_the_queue> <front_index> <rear_index> <command_at_front_index> <command_at_rear_index>!\n\r"
"P!"	Display all not-yet-executed commands in the queue	"<command_at_front_index> <command_at_second_index> <command_at_three_index> …

		<command_at_rear_index>!\n\r”
N/A	Receiving an interrupt from the SM	“_S”
N/A	Receiving an interrupt from the RM	“_R”
N/A	Enqueuing “W”	“_W”
N/A	Enqueuing “S”	“_S”
N/A	Enqueuing “A”	“_A”
N/A	Enqueuing “D”	“_D”
N/A	Enqueuing “E”	“_E”
N/A	If the MM finishes executing an command	“_SF!\n\r”
N/A	If the queue is empty	“_X!\n\r”

Appendix C: Requirement Traceability Matrix

Table 10. Requirements Traceability Matrix

	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
R1	X			X		X		X
R2					X	X		
R3						X		X
R4		X		X		X		
R5						X		X
R6		X		X		X		
R7			X	X			X	
R8							X	
R9					X	X		
R10	X			X		X		X
R11	X			X		X		X
R12							X	
R13							X	

Appendix D: Android Studio Code

**Note for more in depth explanation of the code, please refer to the individual section regarding the corresponding block*

MainActivity.java

```
/*
* Copyright (C) 2014 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.example.vimin.robot_v12;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;

import android.os.Message;


import com.example.vimin.robot_v12.common.logger.Log;


import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.UUID;


/**

* This class does all the work for setting up and managing Bluetooth

* connections with other devices. It has a thread that listens for

* incoming connections, a thread for connecting with a device, and a

* thread for performing data transmissions when connected.

*/

public class BluetoothChatService {

   // Debugging

   private static final String TAG = "BluetoothChatService";


//    //Name for the SDP record when creating server socket

//    private static final String NAME_SECURE = "oBluetoothChatSecure";

//    private static final String NAME_INSECURE = "BluetoothChatInsecure";

//

//    // Unique UUID for this application

//    private static final UUID MY_UUID_SECURE =

//            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

//    private static final UUID MY_UUID_INSECURE =

//           UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");
```

```
    private static final String NAME_SECURE = "HC-05";
    private static final String NAME_INSECURE = "HC-05";

    private static final UUID MY_UUID_SECURE =
            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    private static final UUID MY_UUID_INSECURE =
            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mSecureAcceptThread;
    private AcceptThread mInsecureAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for incoming
connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing
connection
    public static final int STATE_CONNECTED = 3;  // now connected to a remote
device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     *
     * @param context The UI Activity Context
     * @param handler A Handler to send messages back to the UI Activity
     */
    public BluetoothChatService(Context context, Handler handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mHandler = handler;
    }

    /**
     * Set the current state of the chat connection
     *
     * @param state An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        Log.d(TAG, "setState() " + mState + " -> " + state);
        mState = state;

        // Give the new state to the Handler so the UI Activity can update
        mHandler.obtainMessage(Constants.MESSAGE_STATE_CHANGE, state,
-1).sendToTarget();
    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
        return mState;
```

```
}


/**
 * Start the chat service. Specifically start AcceptThread to begin a
 * session in listening (server) mode. Called by the Activity onResume()
 */
public synchronized void start() {
    Log.d(TAG, "start");

    // Cancel any thread attempting to make a connection
    if (mConnectThread != null) {
        mConnectThread.cancel();
        mConnectThread = null;
    }

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {
        mConnectedThread.cancel();
        mConnectedThread = null;
    }

    setState(STATE_LISTEN);

    // Start the thread to listen on a BluetoothServerSocket
    if (mSecureAcceptThread == null) {
        mSecureAcceptThread = new AcceptThread(true);
        mSecureAcceptThread.start();
    }
    if (mInsecureAcceptThread == null) {
        mInsecureAcceptThread = new AcceptThread(false);
        mInsecureAcceptThread.start();
    }
}

/**
 * Start the ConnectThread to initiate a connection to a remote device.
 *
 * @param device The BluetoothDevice to connect
 * @param secure Socket Security type - Secure (true) , Insecure (false)
 */
public synchronized void connect(BluetoothDevice device, boolean secure) {
    Log.d(TAG, "connect to: " + device);

    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
    }

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {
        mConnectedThread.cancel();
        mConnectedThread = null;
    }

    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device, secure);

    mConnectThread.start();

    setState(STATE_CONNECTING);

}


/**

 * Start the ConnectedThread to begin managing a Bluetooth connection

 *

 * @param socket The BluetoothSocket on which the connection was made

 * @param device The BluetoothDevice that has been connected

 */

public synchronized void connected(BluetoothSocket socket, BluetoothDevice

        device, final String socketType) {

    Log.d(TAG, "connected, Socket Type:" + socketType);


    // Cancel the thread that completed the connection

    if (mConnectThread != null) {

        mConnectThread.cancel();

        mConnectThread = null;

    }


    // Cancel any thread currently running a connection

    if (mConnectedThread != null) {

        mConnectedThread.cancel();

        mConnectedThread = null;

    }


    // Cancel the accept thread because we only want to connect to one device

    if (mSecureAcceptThread != null) {
        mSecureAcceptThread.cancel();

        mSecureAcceptThread = null;

    }

    if (mInsecureAcceptThread != null) {

        mInsecureAcceptThread.cancel();

        mInsecureAcceptThread = null;

    }


    // Start the thread to manage the connection and perform transmissions

    mConnectedThread = new ConnectedThread(socket, socketType);

    mConnectedThread.start();


    // Send the name of the connected device back to the UI Activity

    Message msg = mHandler.obtainMessage(Constants.MESSAGE_DEVICE_NAME);

    Bundle bundle = new Bundle();

    bundle.putString(Constants.DEVICE_NAME, device.getName());

    msg.setData(bundle);

    mHandler.sendMessage(msg);


    setState(STATE_CONNECTED);

}


/**

 * Stop all threads

 */

public synchronized void stop() {

    Log.d(TAG, "stop");


    if (mConnectThread != null) {
        mConnectThread.cancel();

        mConnectThread = null;

    }


    if (mConnectedThread != null) {

        mConnectedThread.cancel();

        mConnectedThread = null;

    }


    if (mSecureAcceptThread != null) {

        mSecureAcceptThread.cancel();

        mSecureAcceptThread = null;

    }


    if (mInsecureAcceptThread != null) {

        mInsecureAcceptThread.cancel();

        mInsecureAcceptThread = null;

    }

    setState(STATE_NONE);

}


/**

 * Write to the ConnectedThread in an unsynchronized manner

 *

 * @param out The bytes to write

 * @see ConnectedThread#write(byte[])

 */

public void write(byte[] out) {

    // Create temporary object
    ConnectedThread r;

    // Synchronize a copy of the ConnectedThread

    synchronized (this) {

        if (mState != STATE_CONNECTED) return;

        r = mConnectedThread;

    }

    // Perform the write unsynchronized

    r.write(out);

}


/**

 * Indicate that the connection attempt failed and notify the UI Activity.

 */

private void connectionFailed() {

    // Send a failure message back to the Activity

    Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);

    Bundle bundle = new Bundle();

    bundle.putString(Constants.TOAST, "Unable to connect device");

    msg.setData(bundle);

    mHandler.sendMessage(msg);


    // Start the service over to restart listening mode

    BluetoothChatService.this.start();

}


/**

 * Indicate that the connection was lost and notify the UI Activity.

 */

private void connectionLost() {
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(Constants.TOAST, "Device connection was lost");
    msg.setData(bundle);
    mHandler.sendMessage(msg);

    // Start the service over to restart listening mode
    BluetoothChatService.this.start();
}

/**
 * This thread runs while listening for incoming connections. It behaves
 * like a server-side client. It runs until a connection is accepted
 * (or until cancelled).
 */
private class AcceptThread extends Thread {
    // The local server socket
    private final BluetoothServerSocket mmServerSocket;
    private String mSocketType;

    public AcceptThread(boolean secure) {
        BluetoothServerSocket tmp = null;
        mSocketType = secure ? "Secure" : "Insecure";

        // Create a new listening server socket
        try {
            if (secure) {
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME_SECURE,
                        MY_UUID_SECURE);
            } else {
                tmp = mAdapter.listenUsingInsecureRfcommWithServiceRecord(
                        NAME_INSECURE, MY_UUID_INSECURE);
            }
        } catch (IOException e) {
            Log.e(TAG, "Socket Type: " + mSocketType + "listen() failed", e);
        }
        mmServerSocket = tmp;
    }

    public void run() {
        Log.d(TAG, "Socket Type: " + mSocketType +
                "BEGIN mAcceptThread" + this);
        setName("AcceptThread" + mSocketType);

        BluetoothSocket socket = null;

        // Listen to the server socket if we're not connected
        while (mState != STATE_CONNECTED) {
            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                socket = mmServerSocket.accept();
            } catch (Exception e) {
                Log.e(TAG, "Socket Type: " + mSocketType + "accept() failed",
e);
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                synchronized (BluetoothChatService.this) {
                    switch (mState) {
                        case STATE_LISTEN:
                        case STATE_CONNECTING:
                            // Situation normal. Start the connected thread.
                            connected(socket, socket.getRemoteDevice(),
                                    mSocketType);
                            break;
                        case STATE_NONE:
                        case STATE_CONNECTED:
                            // Either not ready or already connected. Terminate
new socket.
                            try {
                                socket.close();
                            } catch (IOException e) {
                                Log.e(TAG, "Could not close unwanted socket",
e);
                            }
                            break;
                    }
                }
            }
        }
        Log.i(TAG, "END mAcceptThread, socket Type: " + mSocketType);

    }
    public void cancel() {
        Log.d(TAG, "Socket Type" + mSocketType + "cancel " + this);
        try {
            mmServerSocket.close();
        } catch (Exception e) {
            Log.e(TAG, "Socket Type" + mSocketType + "close() of server failed",
e);
        }
    }
}



/**
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;
    private String mSocketType;

    public ConnectThread(BluetoothDevice device, boolean secure) {
        mmDevice = device;
        BluetoothSocket tmp = null;
        mSocketType = secure ? "Secure" : "Insecure";

        // Get a BluetoothSocket for a connection with the
        // given BluetoothDevice
        try {
            if (secure) {
                tmp = device.createRfcommSocketToServiceRecord(
                        MY_UUID_SECURE);
            } else {
                tmp = device.createInsecureRfcommSocketToServiceRecord(
                        MY_UUID_INSECURE);
            }
        } catch (IOException e) {
            Log.e(TAG, "Socket Type: " + mSocketType + "create() failed", e);
        }
        mmSocket = tmp;
    }


    public void run() {
        Log.i(TAG, "BEGIN mConnectThread SocketType:" + mSocketType);
        setName("ConnectThread" + mSocketType);


        // Always cancel discovery because it will slow down a connection
        mAdapter.cancelDiscovery();


        // Make a connection to the BluetoothSocket
        try {
            // This is a blocking call and will only return on a
            // successful connection or an exception
            mmSocket.connect();
        } catch (IOException e) {
            // Close the socket
            try {

                mmSocket.close();

            } catch (IOException e2) {

                Log.e(TAG, "unable to close() " + mSocketType +

                        " socket during connection failure", e2);

            }

            connectionFailed();

            return;

        }


        // Reset the ConnectThread because we're done

        synchronized (BluetoothChatService.this) {

            mConnectThread = null;

        }


        // Start the connected thread

        connected(mmSocket, mmDevice, mSocketType);

    }


    public void cancel() {

        try {

            mmSocket.close();

        } catch (Exception e) {

            Log.e(TAG, "close() of connect " + mSocketType + " socket failed",
e);

        }

    }

}
/**
 * This thread runs during a connection with a remote device.
 * It handles all incoming and outgoing transmissions.
 */
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket, String socketType) {
        Log.d(TAG, "create ConnectedThread: " + socketType);
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "temp sockets not created", e);
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        Log.i(TAG, "BEGIN mConnectedThread");
        byte[] buffer = new byte[1024];

        int bytes;


        // Keep listening to the InputStream while connected

        while (mState == STATE_CONNECTED) {

            try {

                // Read from the InputStream

                bytes = mmInStream.read(buffer);


                // Send the obtained bytes to the UI Activity

                mHandler.obtainMessage(Constants.MESSAGE_READ, bytes, -1,
buffer)

                        .sendToTarget();

            } catch (IOException e) {

                Log.e(TAG, "disconnected", e);

                connectionLost();

                // Start the service over to restart listening mode

                BluetoothChatService.this.start();

                break;

            }

        }

    }


    /**

     * Write to the connected OutStream.

     *

     * @param buffer The bytes to write

     */

    public void write(byte[] buffer) {
        try {
            mmOutStream.write(buffer);


            // Share the sent message back to the UI Activity
            mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1, buffer)
                    .sendToTarget();
        } catch (IOException e) {
            Log.e(TAG, "Exception during write", e);
        }
    }


    public void cancel() {
        try {
            mmSocket.close();
        } catch (Exception e) {
            Log.e(TAG, "close() of connect socket failed", e);
        }
    }
}
}
```

Sensors.java

```
package com.example.vimin.robot_v12;



/**
* Created by Danny on 1/30/2017.
*/
```

```
import android.app.Activity;

import android.view.View;

import android.hardware.Sensor;

import android.hardware.SensorManager;

import android.hardware.SensorEvent;

import android.hardware.SensorEventListener;

import android.widget.TextView;

import android.widget.Toast;


import com.opencsv.*;


import java.io.*;


public class Sensors extends Activity implements SensorEventListener {


   //region Variable

   private MainActivity MA;


   //arrays sizes

   private static final int avgAccelSize = 15, avgGyroSize = 5, avgMagsize = 30; //
for averaging

   private static final int gyroAngleSize = 30, accelDistSize = 30, magAngSize =
30; //for integration

   private  static final int AccSize= 450;

   //counters

   private int accelCounter = 0, gyroCounter = 0, magCounter = 0;

   private int distCounter = 0, angleCounter = 0, magAngCounter = 0;
```

```
//timestamp variables & constants

private long timestamp0 = -1;

private float timestamp_sampled = 0, gyro_timeStamp = 0;;

private final float timeScale = 1000000000.0f;


//data arrays

private float[][] accel = new float[4][avgAccelSize];

private float[][] gyro = new float[3][avgGyroSize];

private float[][] mag = new float[3][avgMagsize];

private float [][] magtemp = new float[3][magAngSize];

private float[] disttemp = new float[accelDistSize];

private float[] gyrotemp = new float[gyroAngleSize];

private float[] Acc = new float[AccSize];

private float offset_Y = -0.007401f;


//CSV writer

private CSVWriter _csvWrite = null;


int ToggleBit = 0;


//storage variables for distances

private float yDist = 0, distNum = 0, totalYDist = 0;

private float zAngle = 0, angleNum = 0, totalAngle = 0;

private float magAngleNum = 0, magTotalAngle = 0;

private double magAngleChange = 0, magAngleRef = 0, currAngle;

private boolean isRefSet = false;


//Bluetooth

BluetoothChatFragment frag;
```

```
    //endregion


    //region Constructor/ Init

    public Sensors(MainActivity MA, BluetoothChatFragment fragment) {


        frag = fragment;

        // Create our Sensor Manager

        this.MA = MA;

        _csvWrite = CSVwrite();

        config_Toggle();

        config_Align();

        config_Check();

        config_Stop();


        // Register sensor Listener

        MA.SM.registerListener(this, MA.myAccSensor,
SensorManager.SENSOR_DELAY_FASTEST);

        MA.SM.registerListener(this, MA.mySensor,
SensorManager.SENSOR_DELAY_FASTEST);

        MA.SM.registerListener(this, MA.myGyroSensor,
SensorManager.SENSOR_DELAY_FASTEST);

        MA.SM.registerListener(this, MA.myMagnet,
SensorManager.SENSOR_DELAY_FASTEST);

    }


    @Override

    public void onAccuracyChanged(Sensor sensor, int accuracy) {

        // Not in use

    }
    @Override

    public void onSensorChanged(SensorEvent event) {

        Sensor mySensor = event.sensor;


        if(mySensor.getType() == Sensor.TYPE_ACCELEROMETER)

        {

            getdirection(event.values, event.timestamp);

        }


        if (mySensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {

//

            updateAccelVal(event.values, event.timestamp);

        }


        if (mySensor.getType() == Sensor.TYPE_GYROSCOPE) {

            updateGyroVal(event.values, event.timestamp);

        }


        if (mySensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {

            updateMagVal(event.values, event.timestamp);

        }


    }

    //endregion

    int AccCounter =0;

    int lineardirection = 0; //0 is forward, 1 is backwards

    private void getdirection(float values[], long timestamp)
    {
        Acc[AccCounter++] = values[1];

        if(AccCounter == AccSize) {
            float average = mean(Acc);
            if(average < -.1)
            {
                lineardirection = 1;
            }
            else
            {
                lineardirection =0;
            }
            MA.yText.setText("Y: " + values[1] + " dir:"+ lineardirection + " " +
(timestamp / timeScale));
            AccCounter =0;
        }
    }

    //region Accelerometer
    int direction =0;
    float accel_average = 0;
    private void updateAccelVal(float values[], long timestamp) {
        int i = 0;
        float threshold = 0.05f;
        //float threshold = 0.5f; // cm/s/s?
        float  yAvg = 0;
        String[] data = new String[6];// string for excel
        if (timestamp0 == -1) {
            timestamp0 = timestamp;
        }
        timestamp -= timestamp0;

        //accel[1][accelCounter] = (Math.abs(values[1]) > threshold) ? values[1] :
0; //yaxis
        accel[1][accelCounter] = (values[1] > threshold) ? values[1] : 0;
        accel[2][accelCounter] = values[1];
        accelCounter++;
        if (accelCounter == avgAccelSize) {

            direction = ((accel_average = mean(accel[2])) < -.1f) ? -1 : 1;
            yAvg = mean(accel[1]);
            //MA.yText.setText("Y: " + values[1] + " " + (timestamp / timeScale));

            accelCounter = 0;
            yAvg -= offset_Y;

            //region excel strings
            data[0] = String.valueOf(timestamp / timeScale);
            data[1] = String.valueOf(0);
            data[2] = String.valueOf(yAvg);
            data[3] = String.valueOf(0);
            data[4] = String.valueOf(0);
            data[5] = String.valueOf(0);
            //endregion

            disttemp[distCounter] = yAvg;
            distCounter++;

            if (distCounter >= accelDistSize - 1) {

                distCounter = 0;

                updateDist(timestamp);

                data[4] = String.valueOf(this.yDist);

                data[5] = String.valueOf(this.totalYDist);

            }

            _csvWrite.writeNext(data);

////////////////////////////////////////////////////////


        }


    }


    int sentMessages = 0;

    private void updateDist(long timestamp) {

        int i = 0, numMessages = 0, centiPerMessage = 20;

        float calibration = 0.05998f * avgAccelSize; //02.999f with 5 samples

        float temp_vals = 0, threshold = 0.01f; // threshold in cm?

//          float temp_vals = 0, threshold = 0.5f; // threshold cm?

        float t_dif = (timestamp / timeScale) - timestamp_sampled;

        timestamp_sampled = (timestamp / timeScale);


        // integrate for displacement




        for (i = 0; i < accelDistSize - 2; i++) {

            temp_vals += (disttemp[i])

                    + (2 * disttemp[i + 1])
                + (disttemp[i + 2]);
        }
        yDist = temp_vals * (t_dif / 30.0f) * 16;
        yDist -= calibration;

        //check if past threshold, send message
//          if (Math.abs(yDist) > threshold){
        if (yDist > threshold){
            totalYDist += yDist;
        }
        distNum += yDist;

        if (distNum > centiPerMessage && ToggleBit == 1) {
            numMessages = (int) distNum / centiPerMessage;
            for (i = 0; i < numMessages; i++) {
                sentMessages++;
                if(lineardirection ==0) {
                    frag.sendMessage("W!");
                }
                else if(lineardirection ==1  ){
                    frag.sendMessage("S!");
                }
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                    System.out.println("error sleeping");
                }
            }
            distNum = distNum % centiPerMessage;
        }


        MA.yDistText.setText("Dist_y: " + this.yDist + " |total:" + totalYDist + "
|sm:" + sentMessages);

        MA.yDistText.append("\n" + "avg:" + accel_average + " |" + direction );


    }

    //endregion


    //region Gyroscope

    private void updateGyroVal(float values[], long timestamp) {

        float  zAvg = 0;


        if (timestamp0 == -1) {

            timestamp0 = timestamp;

        }

        timestamp -= timestamp0;


        gyro[2][gyroCounter] = values[2]; //zaxis

        gyroCounter++;


        if (gyroCounter == avgGyroSize) {


            zAvg = mean(gyro[2]);

            MA.zGText.setText("Z: " + zAvg);

            gyroCounter = 0;


            gyrotemp[angleCounter] = zAvg;

            angleCounter++;
            if (angleCounter >= gyroAngleSize - 1) {

                angleCounter = 0;

                updateAngle(timestamp);

            }

        }

    }




    float leftMessage = 0, rightMessage = 0;


    private void updateAngle(long timestamp) {

        int i, numMessages , anglePerMessage = 45;

        int complement_angle = anglePerMessage * -1;

        float threshold = 0.01f, temp_vals = 0;

        float t_dif = (timestamp / timeScale) - gyro_timeStamp;

        gyro_timeStamp = (timestamp / timeScale);


        for (i = 0; i < gyroAngleSize - 2; i++) {

            temp_vals += (gyrotemp[i])

                    + (2 * gyrotemp[i + 1]);

        }


        zAngle = temp_vals * (t_dif / 30.0f) * 16 * 1.333333f;

        if (Math.abs(zAngle) > threshold) {

            totalAngle += zAngle;

        }

        angleNum += zAngle;


        if ((angleNum > anglePerMessage || angleNum < complement_angle) && ToggleBit
== 1 && !MA.cbMag.isChecked()) {
            numMessages = Math.abs((int) angleNum / anglePerMessage);
            for (i = 0; i < numMessages; i++) {

                if (angleNum > anglePerMessage) {
                    leftMessage++;
                    frag.sendMessage("A!");
                } else {
                    rightMessage++;
                    frag.sendMessage("D!");
                }
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                    Toast.makeText(this, "Sleep Exception thrown",
Toast.LENGTH_LONG).show();
                }
            }
            angleNum = angleNum % anglePerMessage;
        }

        MA.zAngleText.setText("angle: " + zAngle + " |total:" + totalAngle);
        MA.zAngleText.append("\nLeft: " + leftMessage + " |Right: " + rightMessage);

    }
    //endregion

    //region MagneticField
    private void updateMagVal(float values[], long timestamp)
    {
        float xAvg, yAvg, zAvg;
        double angTemp;

        mag[0][magCounter] = values[0];
        mag[1][magCounter] = values[1];
        mag[2][magCounter] = values[2];

        magCounter++;
        if(magCounter == avgMagsize)
        {
            magtemp[0][magAngCounter] = xAvg = mean(mag[0]);
            magtemp[1][magAngCounter] = yAvg = mean(mag[1]);
            magtemp[2][magAngCounter] = zAvg = mean(mag[2]);
            currAngle = angTemp = HMC5983_Direction(xAvg, yAvg);

            MA.magXText.setText("mag_x:" + xAvg);
            MA.magYText.setText("mag_y:" + yAvg);
//             MA.magZText.setText("mag_z:" + zAvg);
//             MA.magZText.append("\nx:y|  " + angTemp);

            magCounter = 0;
            updateMagAngle(timestamp, angTemp);



        }
    }

    int magLeft = 0, magRight = 0;
    double tempAngleRef = 0;

    private void updateMagAngle(long timestamp, double angle) {

        int i, numMessages , anglePerMessage = 30;

        double angTemp = angle;

        int complement_angle = anglePerMessage * -1;

        double threshold = 1;

//          float t_dif = (timestamp / timeScale) - gyro_timeStamp;

//


        if(isRefSet == false) {

            tempAngleRef = angTemp;

            isRefSet = true;

        }


        angTemp-=tempAngleRef;

        if(angTemp > 180)

        {

            angTemp = angTemp - 360;

        }

        else if(angTemp < -180)

        {

            angTemp = angTemp + 360;

        }


        if (Math.abs(angTemp) > threshold) {

            magTotalAngle += angTemp;

            tempAngleRef = angle;

        }

        magAngleNum += angTemp;
        if ((magAngleNum > anglePerMessage || magAngleNum < complement_angle) &&
ToggleBit == 1 && MA.cbMag.isChecked()) {
            numMessages = Math.abs((int) magAngleNum / anglePerMessage);
            for (i = 0; i < numMessages; i++) {

                if (magAngleNum > anglePerMessage) {
                    magRight++;
                    frag.sendMessage("D!");
                } else {
                    magLeft++;
                    frag.sendMessage("A!");
                }
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                    Toast.makeText(this, "Sleep Exception thrown",
Toast.LENGTH_LONG).show();
                }
            }
            magAngleNum = magAngleNum % anglePerMessage;
        }

        MA.magZText.setText("\nangle: " + angle + " |total:" + magTotalAngle);
        MA.magZText.append("\nLeft: " + magLeft + " |Right: " + magRight);

    }

    private double HMC5983_Direction(double x_coor, double z_coor) {
        double directiontmp;

        int californiaCalibration =15;

        float radian = (float)Math.atan(x_coor*1.0/z_coor);


        if (z_coor < 0) {

            directiontmp = 270 - radian*180/3.14;

        } else if (z_coor > 0) {

            directiontmp = 90 - radian*180/3.14;

        } else {

            if (x_coor < 0) {

                directiontmp = 180.0;

            } else {

                directiontmp = 0.0;

            }

        }


        directiontmp += californiaCalibration;

        if(directiontmp >= 360)

        {

            directiontmp -= 360;

        }

        //handle special cases for calibration

        if(directiontmp > 135 && directiontmp <225 )

        {

            directiontmp -=20;

        }

        return directiontmp;

    }
    //endregion


    void config_Toggle() {
        MA.ToggleButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ToggleBit ^= 1;
                if (ToggleBit == 1) {
                    MA.ToggleButton.setText("Disable Tracking Mode");
                } else {
                    MA.ToggleButton.setText("Enable Tracking Mode");
                }
                angleNum = distNum = magAngleNum = 0;
                totalAngle = totalYDist = magTotalAngle =0;
                magAngleRef = currAngle;
            }
        });
    }


    void config_Align() {
        MA.AlignButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                double calibratedAngle;
//                  calibratedAngle = (currAngle) * 0.8148f + 34.167f;
                calibratedAngle = (currAngle) * 0.8148f + 21.944f;
                String tmpStr = String.format("%03d", (int)calibratedAngle);
                frag.sendMessage("E" + tmpStr + "!");
                magAngleRef = currAngle;
                magAngleNum = magTotalAngle =0;

            }

        });

    }


    void config_Check() {

        MA.cbMag.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View view) {


            }

        });

    }


    void config_Stop(){

        MA.Stop.setOnClickListener(new View.OnClickListener(){

            @Override

            public void onClick(View view) {

                // Send a message using content of the edit text widget

                frag.sendMessage("X!");

                if(ToggleBit == 1)

                    Toast.makeText(MA, "Tracking Mode Disabled",

Toast.LENGTH_SHORT).show();

                MA.ToggleButton.setText("Enable Tracking Mode");

                ToggleBit = 0;

            }

        });


    }
```

```
private CSVWriter CSVwrite() {

    String filePath = MA.path + "/AnalysisData3.csv";

    File f = new File(filePath);

    CSVWriter writer = null;

    FileWriter mFileWriter;

    // File exist

    try {

        if (f.exists() && !f.isDirectory()) {

            mFileWriter = new FileWriter(filePath, true);

            writer = new CSVWriter(mFileWriter);

        } else {

            writer = new CSVWriter(new FileWriter(filePath));

        }

        String[] data = {"Time", "Xaxis", "Yaxis", "Zaxis"};


        writer.writeNext(data);


    } catch (Exception e) {

        Toast.makeText(this, "Exception thrown", Toast.LENGTH_LONG).show();

        System.out.println("Exception from file write." + e);

    }

    return writer;

}


public static float mean(float[] m) {

    float sum = 0;

    for (int i = 0; i < m.length; i++) {

        sum += (float)m[i]/ (float)m.length;

    }
```

```
        return sum;
    }
}
```

GPS.java

```
package com.example.vimin.robot_v12;


/**
* Created by Danny on 2/6/2017.
*/



import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

public class GPS extends AppCompatActivity {

    private MainActivity MA;
    private TextView GPS_Text;
    private LocationManager locationManager;
```

```
   private LocationListener listener;


   public GPS(MainActivity MA) {

       this.MA = MA;

       this.GPS_Text = MA.GPS_Text;

       this.locationManager = MA.locationManager;



       listener = new LocationListener() {

           @Override

           public void onLocationChanged(Location location) {

               GPS_Text.setText("\n " + location.getLongitude() + " " +
location.getLatitude());

           }


           @Override

           public void onStatusChanged(String s, int i, Bundle bundle) {


           }


           @Override

           public void onProviderEnabled(String s) {


           }


           @Override

           public void onProviderDisabled(String s) {


               Intent i = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
```

```
                startActivity(i);
            }
        };

        configure_button();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        switch (requestCode) {
            case 10:
                configure_button();
                break;
            default:
                break;
        }
    }

    void configure_button() {
        if (MA.config()) { //check for permissions

            MA.GPSRequestButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    //noinspection MissingPermission

locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 5000, 0,
listener);
```

```
                   //locationManager.requestLocationUpdates("gps", 5000, 0,
listener);
               }
           });
       }


   }
}
```

BluetoothChatFragment.java

```
/*
* Copyright (C) 2014 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.example.vimin.robot_v12;
```

```
import android.app.ActionBar;

import android.app.Activity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.Intent;

import android.os.Bundle;

import android.os.Handler;

import android.os.Message;

import android.support.annotation.Nullable;

import android.support.v4.app.Fragment;

import android.support.v4.app.FragmentActivity;

import android.view.KeyEvent;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.view.ViewGroup;

import android.view.inputmethod.EditorInfo;

import android.widget.ArrayAdapter;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ListView;

import android.widget.TextView;

import android.widget.Toast;

import java.util.*;


import com.example.vimin.robot_v12.common.logger.Log;
```

```
/**

* This fragment controls Bluetooth to communicate with other devices.

*/

public class BluetoothChatFragment extends Fragment {


   private static final String TAG = "BluetoothChatFragment";


   // Intent request codes

   private static final int REQUEST_CONNECT_DEVICE_SECURE = 1;

   private static final int REQUEST_CONNECT_DEVICE_INSECURE = 2;

   private static final int REQUEST_ENABLE_BT = 3;


   // Layout Views

   private ListView mConversationView;

   private EditText mOutEditText;

   private Button mSendButton;

   private Button Forward,Back,Left, Right;


   /**

    * Name of the connected device

    */

   private String mConnectedDeviceName = null;


   /**

    * Array adapter for the conversation thread

    */

   private ArrayAdapter<String> mConversationArrayAdapter;


   /**
```

```
     * String buffer for outgoing messages
     */
    private StringBuffer mOutStringBuffer;

    /**
     * Local Bluetooth adapter
     */
    private BluetoothAdapter mBluetoothAdapter = null;

    /**
     * Member object for the chat services
     */
    private BluetoothChatService mChatService = null;

    private Queue<String> cmdQueue = new LinkedList<>();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        // Get local Bluetooth adapter
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // If the adapter is null, then Bluetooth is not supported
        if (mBluetoothAdapter == null) {
            FragmentActivity activity = getActivity();
            Toast.makeText(activity, "Bluetooth is not available",
Toast.LENGTH_LONG).show();
            activity.finish();
        }
    }



    @Override
    public void onStart() {
        super.onStart();
        // If BT is not on, request that it be enabled.
        // setupChat() will then be called during onActivityResult
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
            // Otherwise, setup the chat session
        } else if (mChatService == null) {
            setupChat();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mChatService != null) {
            mChatService.stop();
        }
    }

    @Override
    public void onResume() {
        super.onResume();


        // Performing this check in onResume() covers the case in which BT was
        // not enabled during onStart(), so we were paused to enable it...
        // onResume() will be called when ACTION_REQUEST_ENABLE activity returns.
        if (mChatService != null) {
            // Only if the state is STATE_NONE, do we know that we haven't started
already
            if (mChatService.getState() == BluetoothChatService.STATE_NONE) {
                // Start the Bluetooth chat services
                mChatService.start();
            }
        }
    }


    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_bluetooth_chat, container, false);
    }


    @Override
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
        mConversationView = (ListView) view.findViewById(R.id.in);
        mOutEditText = (EditText) view.findViewById(R.id.edit_text_out);
        mSendButton = (Button) view.findViewById(R.id.button_send);
        Forward = (Button) view.findViewById(R.id.Forward_Button);
        Back = (Button) view.findViewById(R.id.Back_Button);
        Left = (Button) view.findViewById(R.id.Left_Button);
        Right = (Button) view.findViewById(R.id.Right_Button);


    }


    /**

     * Set up the UI and background operations for chat.

     */

    private void setupChat() {

        Log.d(TAG, "setupChat()");


        // Initialize the array adapter for the conversation thread

        mConversationArrayAdapter = new ArrayAdapter<String>(getActivity(),

R.layout.message);


        mConversationView.setAdapter(mConversationArrayAdapter);


        // Initialize the compose field with a listener for the return key

        mOutEditText.setOnEditorActionListener(mWriteListener);


        // Initialize the send button with a listener that for click events

        mSendButton.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                // Send a message using content of the edit text widget

                View view = getView();

                if (null != view) {

                    TextView textView = (TextView)

view.findViewById(R.id.edit_text_out);

                    String message = textView.getText().toString();

                    sendMessage(message);
                }

            }

        });


        Forward.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                // Send a message using content of the edit text widget

                View view = getView();

                if (null != view) {

                    sendMessage("W!");

                }

            }

        });

        Left.setOnClickListener(new View.OnClickListener(){

            public void onClick(View v) {

                // Send a message using content of the edit text widget

                View view = getView();

                if (null != view) {

                    sendMessage("A!");

                }

            }

        });

        Right.setOnClickListener(new View.OnClickListener(){

            public void onClick(View v) {

                // Send a message using content of the edit text widget

                View view = getView();

                if (null != view) {

                   sendMessage("D!");

                }
            }

        });

        Back.setOnClickListener(new View.OnClickListener(){

            public void onClick(View v) {

                // Send a message using content of the edit text widget

                View view = getView();

                if (null != view) {

                    sendMessage("S!");

                }

            }

        });



        // Initialize the BluetoothChatService to perform bluetooth connections

        mChatService = new BluetoothChatService(getActivity(), mHandler);


        // Initialize the buffer for outgoing messages

        mOutStringBuffer = new StringBuffer("");

    }


    /**

     * Makes this device discoverable.

     */

    private void ensureDiscoverable() {

        if (mBluetoothAdapter.getScanMode() !=

                BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {

            Intent discoverableIntent = new

Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
```

```
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }

    /**
     * Sends a message.
     *
     * @param message A string of text to send.
     */
    public void sendMessage(String message) {
        // Check that we're actually connected before trying anything
        //if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
        //    Toast.makeText(getActivity(), R.string.not_connected,
Toast.LENGTH_SHORT).show();
        //    return;
        //}

        // Check that there's actually something to send
        if (message.length() > 0) {
            // Get the message bytes and tell the BluetoothChatService to write
            byte[] send = message.getBytes();
            mChatService.write(send);

            // Reset out string buffer to zero and clear the edit text field
            mOutStringBuffer.setLength(0);
            mOutEditText.setText(mOutStringBuffer);
        }
    }
    /**
     * The action listener for the EditText widget, to listen for the return key
     */
    private TextView.OnEditorActionListener mWriteListener
            = new TextView.OnEditorActionListener() {
        public boolean onEditorAction(TextView view, int actionId, KeyEvent event) {
            // If the action is a key-up event on the return key, send the message
            if (actionId == EditorInfo.IME_NULL && event.getAction() ==
KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
                sendMessage(message);
            }
            return true;
        }
    };

    /**
     * Updates the status on the action bar.
     *
     * @param resId a string resource ID
     */
    private void setStatus(int resId) {
        FragmentActivity activity = getActivity();
        if (null == activity) {
            return;
        }
        final ActionBar actionBar = activity.getActionBar();
        if (null == actionBar) {
            return;
        }
        actionBar.setSubtitle(resId);
    }

    /**
     * Updates the status on the action bar.
     *
     * @param subTitle status
     */
    private void setStatus(CharSequence subTitle) {
        FragmentActivity activity = getActivity();
        if (null == activity) {
            return;
        }
        final ActionBar actionBar = activity.getActionBar();
        if (null == actionBar) {
            return;
        }
        actionBar.setSubtitle(subTitle);
    }

    /**
     * The Handler that gets information back from the BluetoothChatService
     */
    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            FragmentActivity activity = getActivity();
            //Toast.makeText(activity, "act: " + msg.toString(),
Toast.LENGTH_SHORT).show();
            msg = msg;
            switch (msg.what) {
                case Constants.MESSAGE_STATE_CHANGE:
                    switch (msg.arg1) {
                        case BluetoothChatService.STATE_CONNECTED:
                            setStatus(getString(R.string.title_connected_to,
mConnectedDeviceName));
                            mConversationArrayAdapter.clear();
                            break;
                        case BluetoothChatService.STATE_CONNECTING:
                            setStatus(R.string.title_connecting);
                            break;
                        case BluetoothChatService.STATE_LISTEN:
                        case BluetoothChatService.STATE_NONE:
                            setStatus(R.string.title_not_connected);
                            break;
                    }
                    break;
                case Constants.MESSAGE_WRITE:
                    byte[] writeBuf = (byte[]) msg.obj;
                    // construct a string from the buffer
                    String writeMessage = new String(writeBuf);
                    mConversationArrayAdapter.add("Me:  " + writeMessage);
                    break;
                case Constants.MESSAGE_READ:
                    byte[] readBuf = (byte[]) msg.obj;
                    // construct a string from the valid bytes in the buffer
                    String readMessage = new String(readBuf, 0, msg.arg1);
                    mConversationArrayAdapter.add(mConnectedDeviceName + ":  " +
readMessage);
                   // Toast.makeText(activity, "Recv Msg: " + readMessage,
Toast.LENGTH_SHORT).show();
                    break;
                case Constants.MESSAGE_DEVICE_NAME:
                    // save the connected device's name
                    mConnectedDeviceName =
msg.getData().getString(Constants.DEVICE_NAME);
                    if (null != activity) {
                        Toast.makeText(activity, "Connected to "
                                + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
                    }
                    break;
                case Constants.MESSAGE_TOAST:
                    if (null != activity) {
                        Toast.makeText(activity,
msg.getData().getString(Constants.TOAST),
                                Toast.LENGTH_SHORT).show();
                    }
                    break;
            }
        }
    };

    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
            case REQUEST_CONNECT_DEVICE_SECURE:
                // When DeviceListActivity returns with a device to connect
                if (resultCode == Activity.RESULT_OK) {
                    connectDevice(data, true);
                }
                break;
            case REQUEST_CONNECT_DEVICE_INSECURE:
                // When DeviceListActivity returns with a device to connect
                if (resultCode == Activity.RESULT_OK) {
                    connectDevice(data, false);
                }
                break;
            case REQUEST_ENABLE_BT:
                // When the request to enable Bluetooth returns
                if (resultCode == Activity.RESULT_OK) {
                    // Bluetooth is now enabled, so set up a chat session
                    setupChat();
                } else {
                    // User did not enable Bluetooth or an error occurred
                    Log.d(TAG, "BT not enabled");
                    Toast.makeText(getActivity(), R.string.bt_not_enabled_leaving,
                            Toast.LENGTH_SHORT).show();
                    getActivity().finish();
                }
        }
    }

    /**
     * Establish connection with other divice
     *
     * @param data   An {@link Intent} with {@link
DeviceListActivity#EXTRA_DEVICE_ADDRESS} extra.
     * @param secure Socket Security type - Secure (true) , Insecure (false)
     */
    private void connectDevice(Intent data, boolean secure) {
        // Get the device MAC address
        String address = data.getExtras()
                .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
        // Get the BluetoothDevice object
        BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
        // Attempt to connect to the device
        mChatService.connect(device, secure);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        inflater.inflate(R.menu.bluetooth_chat, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.secure_connect_scan: {
                // Launch the DeviceListActivity to see devices and do scan
                Intent serverIntent = new Intent(getActivity(),
DeviceListActivity.class);
                startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE_SECURE);
                return true;
            }
            case R.id.insecure_connect_scan: {
                // Launch the DeviceListActivity to see devices and do scan
                Intent serverIntent = new Intent(getActivity(),
DeviceListActivity.class);
                startActivityForResult(serverIntent,
REQUEST_CONNECT_DEVICE_INSECURE);
                return true;
            }
            case R.id.discoverable: {
                // Ensure this device is discoverable by others
                ensureDiscoverable();
                return true;
            }
        }
        return false;
    }
}
```

DeviceListActivity.java

```
/*
* Copyright (C) 2014 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
```

```
* Unless required by applicable law or agreed to in writing, software

* distributed under the License is distributed on an "AS IS" BASIS,

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

* See the License for the specific language governing permissions and

* limitations under the License.

*/


package com.example.vimin.robot_v12;


import android.app.Activity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.content.IntentFilter;

import android.os.Bundle;

import android.view.View;

import android.view.Window;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.Button;

import android.widget.ListView;

import android.widget.TextView;


import com.example.vimin.robot_v12.common.logger.Log;


import java.util.Set;
```

```
/**

* This Activity appears as a dialog. It lists any paired devices and

* devices detected in the area after discovery. When a device is chosen

* by the user, the MAC address of the device is sent back to the parent

* Activity in the result Intent.

*/

public class DeviceListActivity extends Activity {


   /**

    * Tag for Log

    */

   private static final String TAG = "DeviceListActivity";


   /**

    * Return Intent extra

    */

   public static String EXTRA_DEVICE_ADDRESS = "device_address";


   /**

    * Member fields

    */

   private BluetoothAdapter mBtAdapter;


   /**

    * Newly discovered devices

    */

   private ArrayAdapter<String> mNewDevicesArrayAdapter;


   @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Setup the window
        //requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.activity_device_list);

        // Set result CANCELED in case the user backs out
        setResult(Activity.RESULT_CANCELED);

        // Initialize the button to perform device discovery
        Button scanButton = (Button) findViewById(R.id.button_scan);
        scanButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                doDiscovery();
                v.setVisibility(View.GONE);
            }
        });

        // Initialize array adapters. One for already paired devices and
        // one for newly discovered devices
        ArrayAdapter<String> pairedDevicesArrayAdapter =
                new ArrayAdapter<String>(this, R.layout.device_name);
        mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
R.layout.device_name);

        // Find and set up the ListView for paired devices
        ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
        pairedListView.setAdapter(pairedDevicesArrayAdapter);
        pairedListView.setOnItemClickListener(mDeviceClickListener);


        // Find and set up the ListView for newly discovered devices

        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);

        newDevicesListView.setAdapter(mNewDevicesArrayAdapter);

        newDevicesListView.setOnItemClickListener(mDeviceClickListener);


        // Register for broadcasts when a device is discovered

        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);

        this.registerReceiver(mReceiver, filter);


        // Register for broadcasts when discovery has finished

        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);

        this.registerReceiver(mReceiver, filter);


        // Get the local Bluetooth adapter

        mBtAdapter = BluetoothAdapter.getDefaultAdapter();


        // Get a set of currently paired devices

        Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();


        // If there are paired devices, add each one to the ArrayAdapter

        if (pairedDevices.size() > 0) {

            findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);

            for (BluetoothDevice device : pairedDevices) {

                pairedDevicesArrayAdapter.add(device.getName() + "\n" +

device.getAddress());

            }

        } else {
            String noDevices =
getResources().getText(R.string.none_paired).toString();
            pairedDevicesArrayAdapter.add(noDevices);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null) {
            mBtAdapter.cancelDiscovery();
        }

        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
    }

    /**
     * Start device discover with the BluetoothAdapter
     */
    private void doDiscovery() {
        Log.d(TAG, "doDiscovery()");

        // Indicate scanning in the title
        //setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);
        // Turn on sub-title for new devices
        findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering()) {
            mBtAdapter.cancelDiscovery();
        }

        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
    }

    /**
     * The on-click listener for all devices in the ListViews
     */
    private AdapterView.OnItemClickListener mDeviceClickListener
            = new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
            // Cancel discovery because it's costly and we're about to connect
            mBtAdapter.cancelDiscovery();

            // Get the device MAC address, which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Create the result Intent and include the MAC address
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
            // Set result and finish this Activity
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    };

    /**
     * The BroadcastReceiver that listens for discovered devices and changes the
title when
     * discovery is finished
     */
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            // When discovery finds a device
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                // If it's already paired, skip it, because it's been listed already
                if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
                }
                // When discovery is finished, change the Activity title
            } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                //setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);

                if (mNewDevicesArrayAdapter.getCount() == 0) {

                    String noDevices =

getResources().getText(R.string.none_found).toString();

                    mNewDevicesArrayAdapter.add(noDevices);

                }

            }

        }

    };

}
```

BluetoothChatService.java

```
/*

* Copyright (C) 2014 The Android Open Source Project

*

* Licensed under the Apache License, Version 2.0 (the "License");

* you may not use this file except in compliance with the License.

* You may obtain a copy of the License at

*

*      http://www.apache.org/licenses/LICENSE-2.0

*

* Unless required by applicable law or agreed to in writing, software

* distributed under the License is distributed on an "AS IS" BASIS,

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

* See the License for the specific language governing permissions and

* limitations under the License.

*/
```

```
package com.example.vimin.robot_v12;


import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothServerSocket;

import android.bluetooth.BluetoothSocket;

import android.content.Context;

import android.os.Bundle;

import android.os.Handler;

import android.os.Message;


import com.example.vimin.robot_v12.common.logger.Log;


import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.UUID;


/**

* This class does all the work for setting up and managing Bluetooth

* connections with other devices. It has a thread that listens for

* incoming connections, a thread for connecting with a device, and a

* thread for performing data transmissions when connected.

*/

public class BluetoothChatService {

   // Debugging

   private static final String TAG = "BluetoothChatService";
```

```
//    //Name for the SDP record when creating server socket

//    private static final String NAME_SECURE = "BluetoothChatSecure";

//    private static final String NAME_INSECURE = "BluetoothChatInsecure";

//

//    // Unique UUID for this application

//    private static final UUID MY_UUID_SECURE =

//            UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

//    private static final UUID MY_UUID_INSECURE =

//           UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");


   private static final String NAME_SECURE = "HC-05";

   private static final String NAME_INSECURE = "HC-05";


   private static final UUID MY_UUID_SECURE =

           UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

   private static final UUID MY_UUID_INSECURE =

           UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");


   // Member fields

   private final BluetoothAdapter mAdapter;

   private final Handler mHandler;

   private AcceptThread mSecureAcceptThread;

   private AcceptThread mInsecureAcceptThread;

   private ConnectThread mConnectThread;

   private ConnectedThread mConnectedThread;

   private int mState;


   // Constants that indicate the current connection state

   public static final int STATE_NONE = 0;       // we're doing nothing
```

```
    public static final int STATE_LISTEN = 1;     // now listening for incoming
connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing
connection
    public static final int STATE_CONNECTED = 3;  // now connected to a remote
device

    /**
     * Constructor. Prepares a new BluetoothChat session.
     *
     * @param context The UI Activity Context
     * @param handler A Handler to send messages back to the UI Activity
     */
    public BluetoothChatService(Context context, Handler handler) {
        mAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mHandler = handler;
    }

    /**
     * Set the current state of the chat connection
     *
     * @param state An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        Log.d(TAG, "setState() " + mState + " -> " + state);
        mState = state;

        // Give the new state to the Handler so the UI Activity can update
        mHandler.obtainMessage(Constants.MESSAGE_STATE_CHANGE, state,
-1).sendToTarget();
    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
        return mState;
    }

    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity onResume()
     */
    public synchronized void start() {
        Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
        setState(STATE_LISTEN);

        // Start the thread to listen on a BluetoothServerSocket
        if (mSecureAcceptThread == null) {
            mSecureAcceptThread = new AcceptThread(true);
            mSecureAcceptThread.start();
        }
        if (mInsecureAcceptThread == null) {
            mInsecureAcceptThread = new AcceptThread(false);
            mInsecureAcceptThread.start();
        }
    }

    /**
     * Start the ConnectThread to initiate a connection to a remote device.
     *
     * @param device The BluetoothDevice to connect
     * @param secure Socket Security type - Secure (true) , Insecure (false)
     */
    public synchronized void connect(BluetoothDevice device, boolean secure) {
        Log.d(TAG, "connect to: " + device);

        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        // Start the thread to connect with the given device
        mConnectThread = new ConnectThread(device, secure);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection
     *
     * @param socket The BluetoothSocket on which the connection was made
     * @param device The BluetoothDevice that has been connected
     */
    public synchronized void connected(BluetoothSocket socket, BluetoothDevice
            device, final String socketType) {
        Log.d(TAG, "connected, Socket Type:" + socketType);

        // Cancel the thread that completed the connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
```

```
// Cancel any thread currently running a connection
if (mConnectedThread != null) {
    mConnectedThread.cancel();
    mConnectedThread = null;
}

// Cancel the accept thread because we only want to connect to one device
if (mSecureAcceptThread != null) {
    mSecureAcceptThread.cancel();
    mSecureAcceptThread = null;
}
if (mInsecureAcceptThread != null) {
    mInsecureAcceptThread.cancel();
    mInsecureAcceptThread = null;
}

// Start the thread to manage the connection and perform transmissions
mConnectedThread = new ConnectedThread(socket, socketType);
mConnectedThread.start();

// Send the name of the connected device back to the UI Activity
Message msg = mHandler.obtainMessage(Constants.MESSAGE_DEVICE_NAME);
Bundle bundle = new Bundle();
bundle.putString(Constants.DEVICE_NAME, device.getName());
msg.setData(bundle);
mHandler.sendMessage(msg);


setState(STATE_CONNECTED);
```

```
}


/**
 * Stop all threads
 */
public synchronized void stop() {
    Log.d(TAG, "stop");

    if (mConnectThread != null) {
        mConnectThread.cancel();
        mConnectThread = null;
    }

    if (mConnectedThread != null) {
        mConnectedThread.cancel();
        mConnectedThread = null;
    }

    if (mSecureAcceptThread != null) {
        mSecureAcceptThread.cancel();
        mSecureAcceptThread = null;
    }

    if (mInsecureAcceptThread != null) {
        mInsecureAcceptThread.cancel();
        mInsecureAcceptThread = null;
    }
    setState(STATE_NONE);
}
```

```
/**
 * Write to the ConnectedThread in an unsynchronized manner
 *
 * @param out The bytes to write
 * @see ConnectedThread#write(byte[])
 */
public void write(byte[] out) {
    // Create temporary object
    ConnectedThread r;
    // Synchronize a copy of the ConnectedThread
    synchronized (this) {
        if (mState != STATE_CONNECTED) return;
        r = mConnectedThread;
    }
    // Perform the write unsynchronized
    r.write(out);
}

/**
 * Indicate that the connection attempt failed and notify the UI Activity.
 */
private void connectionFailed() {
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(Constants.TOAST, "Unable to connect device");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
```

```
        // Start the service over to restart listening mode
        BluetoothChatService.this.start();
    }

    /**
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.TOAST, "Device connection was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        // Start the service over to restart listening mode
        BluetoothChatService.this.start();
    }

    /**
     * This thread runs while listening for incoming connections. It behaves
     * like a server-side client. It runs until a connection is accepted
     * (or until cancelled).
     */
    private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;
        private String mSocketType;
        public AcceptThread(boolean secure) {
            BluetoothServerSocket tmp = null;
            mSocketType = secure ? "Secure" : "Insecure";

            // Create a new listening server socket
            try {
                if (secure) {
                    tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME_SECURE,
                            MY_UUID_SECURE);
                } else {
                    tmp = mAdapter.listenUsingInsecureRfcommWithServiceRecord(
                            NAME_INSECURE, MY_UUID_INSECURE);
                }
            } catch (IOException e) {
                Log.e(TAG, "Socket Type: " + mSocketType + "listen() failed", e);
            }
            mmServerSocket = tmp;
        }

        public void run() {
            Log.d(TAG, "Socket Type: " + mSocketType +
                    "BEGIN mAcceptThread" + this);
            setName("AcceptThread" + mSocketType);

            BluetoothSocket socket = null;

            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    socket = mmServerSocket.accept();
                } catch (Exception e) {
                    Log.e(TAG, "Socket Type: " + mSocketType + "accept() failed",
e);
                    break;
                }

                // If a connection was accepted
                if (socket != null) {
                    synchronized (BluetoothChatService.this) {
                        switch (mState) {
                            case STATE_LISTEN:
                            case STATE_CONNECTING:
                                // Situation normal. Start the connected thread.
                                connected(socket, socket.getRemoteDevice(),
                                        mSocketType);
                                break;
                            case STATE_NONE:
                            case STATE_CONNECTED:
                                // Either not ready or already connected. Terminate
new socket.
                                try {
                                    socket.close();
                                } catch (IOException e) {
                                    Log.e(TAG, "Could not close unwanted socket",
e);
```

```
                                }
                                break;
                        }
                    }
                }
            }
            Log.i(TAG, "END mAcceptThread, socket Type: " + mSocketType);

        }

        public void cancel() {
            Log.d(TAG, "Socket Type" + mSocketType + "cancel " + this);
            try {
                mmServerSocket.close();
            } catch (Exception e) {
                Log.e(TAG, "Socket Type" + mSocketType + "close() of server failed",
e);
            }
        }
    }



    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        private String mSocketType;



        public ConnectThread(BluetoothDevice device, boolean secure) {

            mmDevice = device;

            BluetoothSocket tmp = null;

            mSocketType = secure ? "Secure" : "Insecure";



            // Get a BluetoothSocket for a connection with the

            // given BluetoothDevice

            try {

                if (secure) {

                    tmp = device.createRfcommSocketToServiceRecord(

                            MY_UUID_SECURE);

                } else {

                    tmp = device.createInsecureRfcommSocketToServiceRecord(

                            MY_UUID_INSECURE);

                }

            } catch (IOException e) {

                Log.e(TAG, "Socket Type: " + mSocketType + "create() failed", e);

            }

            mmSocket = tmp;

        }



        public void run() {

            Log.i(TAG, "BEGIN mConnectThread SocketType:" + mSocketType);

            setName("ConnectThread" + mSocketType);



            // Always cancel discovery because it will slow down a connection
            mAdapter.cancelDiscovery();


            // Make a connection to the BluetoothSocket

            try {

                // This is a blocking call and will only return on a

                // successful connection or an exception

                mmSocket.connect();

            } catch (IOException e) {

                // Close the socket

                try {

                    mmSocket.close();

                } catch (IOException e2) {

                    Log.e(TAG, "unable to close() " + mSocketType +

                            " socket during connection failure", e2);

                }

                connectionFailed();

                return;

            }


            // Reset the ConnectThread because we're done

            synchronized (BluetoothChatService.this) {

                mConnectThread = null;

            }


            // Start the connected thread

            connected(mmSocket, mmDevice, mSocketType);

        }


        public void cancel() {
            try {
                mmSocket.close();
            } catch (Exception e) {
                Log.e(TAG, "close() of connect " + mSocketType + " socket failed",
e);
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device.
     * It handles all incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket, String socketType) {
            Log.d(TAG, "create ConnectedThread: " + socketType);
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte[1024];
            int bytes;

            // Keep listening to the InputStream while connected
            while (mState == STATE_CONNECTED) {
                try {
                    // Read from the InputStream
                    bytes = mmInStream.read(buffer);

                    // Send the obtained bytes to the UI Activity
                    mHandler.obtainMessage(Constants.MESSAGE_READ, bytes, -1,
buffer)
                            .sendToTarget();
                } catch (IOException e) {
                    Log.e(TAG, "disconnected", e);
                    connectionLost();
                    // Start the service over to restart listening mode
                    BluetoothChatService.this.start();
                    break;
                }
            }
        }

        /**
         * Write to the connected OutStream.
         *
         * @param buffer The bytes to write
         */
        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);

                // Share the sent message back to the UI Activity
                mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "Exception during write", e);
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (Exception e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
}
```

Constants.java

```
/*
* Copyright (C) 2014 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.example.vimin.robot_v12;

/**
* Defines several constants used between {@link BluetoothChatService} and the UI.
*/
public interface Constants {

   // Message types sent from the BluetoothChatService Handler
   public static final int MESSAGE_STATE_CHANGE = 1;
   public static final int MESSAGE_READ = 2;
```

```
    public static final int MESSAGE_WRITE = 3;

    public static final int MESSAGE_DEVICE_NAME = 4;

    public static final int MESSAGE_TOAST = 5;


    // Key names received from the BluetoothChatService Handler

    public static final String DEVICE_NAME = "device_name";

    public static final String TOAST = "toast";

}
```

log.java

```
/*

* Copyright (C) 2013 The Android Open Source Project

*

* Licensed under the Apache License, Version 2.0 (the "License");

* you may not use this file except in compliance with the License.

* You may obtain a copy of the License at

*

*      http://www.apache.org/licenses/LICENSE-2.0

*

* Unless required by applicable law or agreed to in writing, software

* distributed under the License is distributed on an "AS IS" BASIS,

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

* See the License for the specific language governing permissions and

* limitations under the License.

*/

package com.example.vimin.robot_v12.common.logger;


/**
```

```
* Helper class for a list (or tree) of LoggerNodes.
*
* <p>When this is set as the head of the list,
* an instance of it can function as a drop-in replacement for {@link
android.util.Log}.
* Most of the methods in this class server only to map a method call in Log to its
equivalent
* in LogNode.</p>
*/
public class Log {
   // Grabbing the native values from Android's native logging facilities,
   // to make for easy migration and interop.
   public static final int NONE = -1;
   public static final int VERBOSE = android.util.Log.VERBOSE;
   public static final int DEBUG = android.util.Log.DEBUG;
   public static final int INFO = android.util.Log.INFO;
   public static final int WARN = android.util.Log.WARN;
   public static final int ERROR = android.util.Log.ERROR;
   public static final int ASSERT = android.util.Log.ASSERT;

   // Stores the beginning of the LogNode topology.
   private static LogNode mLogNode;

   /**
    * Returns the next LogNode in the linked list.
    */
   public static LogNode getLogNode() {
       return mLogNode;
   }
```

```
    /**
     * Sets the LogNode data will be sent to.
     */
    public static void setLogNode(LogNode node) {
        mLogNode = node;
    }

    /**
     * Instructs the LogNode to print the log data provided. Other LogNodes can
     * be chained to the end of the LogNode as desired.
     *
     * @param priority Log level of the data being logged. Verbose, Error, etc.
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
    public static void println(int priority, String tag, String msg, Throwable tr) {
        if (mLogNode != null) {
            mLogNode.println(priority, tag, msg, tr);
        }
    }

    /**
     * Instructs the LogNode to print the log data provided. Other LogNodes can
     * be chained to the end of the LogNode as desired.
     *
     * @param priority Log level of the data being logged. Verbose, Error, etc.
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged. The actual message to be logged.
     */
    public static void println(int priority, String tag, String msg) {
        println(priority, tag, msg, null);
    }

   /**
     * Prints a message at VERBOSE priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
    public static void v(String tag, String msg, Throwable tr) {
        println(VERBOSE, tag, msg, tr);
    }

    /**
     * Prints a message at VERBOSE priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     */
    public static void v(String tag, String msg) {
        v(tag, msg, null);
    }



    /**
     * Prints a message at DEBUG priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *            to extract and print useful information.
     */
    public static void d(String tag, String msg, Throwable tr) {
        println(DEBUG, tag, msg, tr);
    }

    /**
     * Prints a message at DEBUG priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     */
    public static void d(String tag, String msg) {
        d(tag, msg, null);
    }

    /**
     * Prints a message at INFO priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.

     * @param msg The actual message to be logged.

     * @param tr If an exception was thrown, this can be sent along for the logging
facilities

     *            to extract and print useful information.

     */

    public static void i(String tag, String msg, Throwable tr) {

        println(INFO, tag, msg, tr);

    }


    /**

     * Prints a message at INFO priority.

     *

     * @param tag Tag for for the log data. Can be used to organize log statements.

     * @param msg The actual message to be logged.

     */

    public static void i(String tag, String msg) {

        i(tag, msg, null);

    }


    /**

     * Prints a message at WARN priority.

     *

     * @param tag Tag for for the log data. Can be used to organize log statements.

     * @param msg The actual message to be logged.

     * @param tr If an exception was thrown, this can be sent along for the logging
facilities

     *            to extract and print useful information.

     */
    public static void w(String tag, String msg, Throwable tr) {

        println(WARN, tag, msg, tr);

    }


    /**

     * Prints a message at WARN priority.

     *

     * @param tag Tag for for the log data. Can be used to organize log statements.

     * @param msg The actual message to be logged.

     */

    public static void w(String tag, String msg) {

        w(tag, msg, null);

    }


    /**

     * Prints a message at WARN priority.

     *

     * @param tag Tag for for the log data. Can be used to organize log statements.

     * @param tr If an exception was thrown, this can be sent along for the logging
facilities

     *           to extract and print useful information.

     */

    public static void w(String tag, Throwable tr) {

        w(tag, null, tr);

    }


    /**

     * Prints a message at ERROR priority.

     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
    public static void e(String tag, String msg, Throwable tr) {
        println(ERROR, tag, msg, tr);
    }

    /**
     * Prints a message at ERROR priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     */
    public static void e(String tag, String msg) {
        e(tag, msg, null);
    }

    /**
     * Prints a message at ASSERT priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
    public static void wtf(String tag, String msg, Throwable tr) {
        println(ASSERT, tag, msg, tr);
    }

    /**
     * Prints a message at ASSERT priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param msg The actual message to be logged.
     */
    public static void wtf(String tag, String msg) {
        wtf(tag, msg, null);
    }

    /**
     * Prints a message at ASSERT priority.
     *
     * @param tag Tag for for the log data. Can be used to organize log statements.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
    public static void wtf(String tag, Throwable tr) {
        wtf(tag, null, tr);
    }
}
```

LogFragment.java

```
*/

package com.example.vimin.robot_v12.common.logger;

import android.graphics.Typeface;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ScrollView;

/**
* Simple fraggment which contains a LogView and uses is to output log data it
receives
* through the LogNode interface.
*/
public class LogFragment extends Fragment {

   private LogView mLogView;
   private ScrollView mScrollView;

   public LogFragment() {}

   public View inflateViews() {
       mScrollView = new ScrollView(getActivity());
```

```
        ViewGroup.LayoutParams scrollParams = new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT);
        mScrollView.setLayoutParams(scrollParams);

        mLogView = new LogView(getActivity());
        ViewGroup.LayoutParams logParams = new ViewGroup.LayoutParams(scrollParams);
        logParams.height = ViewGroup.LayoutParams.WRAP_CONTENT;
        mLogView.setLayoutParams(logParams);
        mLogView.setClickable(true);
        mLogView.setFocusable(true);
        mLogView.setTypeface(Typeface.MONOSPACE);

        // Want to set padding as 16 dips, setPadding takes pixels.  Hooray math!
        int paddingDips = 16;
        double scale = getResources().getDisplayMetrics().density;
        int paddingPixels = (int) ((paddingDips * (scale)) + .5);
        mLogView.setPadding(paddingPixels, paddingPixels, paddingPixels,
paddingPixels);
        mLogView.setCompoundDrawablePadding(paddingPixels);

        mLogView.setGravity(Gravity.BOTTOM);
        mLogView.setTextAppearance(getActivity(),
android.R.style.TextAppearance_Holo_Medium);

        mScrollView.addView(mLogView);
        return mScrollView;
    }
```

```
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {

       View result = inflateViews();

       mLogView.addTextChangedListener(new TextWatcher() {
           @Override
           public void beforeTextChanged(CharSequence s, int start, int count, int
after) {}

           @Override
           public void onTextChanged(CharSequence s, int start, int before, int
count) {}

           @Override
           public void afterTextChanged(Editable s) {
               mScrollView.fullScroll(ScrollView.FOCUS_DOWN);
           }
       });
       return result;
   }

   public LogView getLogView() {
       return mLogView;
   }
}
```

LogNode.java

```
/*
* Copyright (C) 2012 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.example.vimin.robot_v12.common.logger;


/**
* Basic interface for a logging system that can output to one or more targets.
* Note that in addition to classes that will output these logs in some format,
* one can also implement this interface over a filter and insert that in the chain,
* such that no targets further down see certain data, or see manipulated forms of
the data.
* You could, for instance, write a "ToHtmlLoggerNode" that just converted all the
log data
* it received to HTML and sent it along to the next node in the chain, without
printing it
* anywhere.
```

```
*/
public interface LogNode {


   /**
    * Instructs first LogNode in the list to print the log data provided.
    * @param priority Log level of the data being logged.  Verbose, Error, etc.
    * @param tag Tag for for the log data.  Can be used to organize log statements.
    * @param msg The actual message to be logged. The actual message to be logged.
    * @param tr If an exception was thrown, this can be sent along for the logging
facilities
    *           to extract and print useful information.
    */
   public void println(int priority, String tag, String msg, Throwable tr);
}
```

LogView.java

```
/*
* Copyright (C) 2013 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.example.vimin.robot_v12.common.logger;

import android.app.Activity;
import android.content.Context;
import android.util.AttributeSet;
import android.widget.TextView;

/** Simple TextView which is used to output log data received through the LogNode
interface.
*/
public class LogView extends TextView implements LogNode {

   public LogView(Context context) {
       super(context);
   }

   public LogView(Context context, AttributeSet attrs) {
       super(context, attrs);
   }

   public LogView(Context context, AttributeSet attrs, int defStyle) {
       super(context, attrs, defStyle);
   }

   /**
```

```
     * Formats the log data and prints it out to the LogView.
     * @param priority Log level of the data being logged.  Verbose, Error, etc.
     * @param tag Tag for for the log data.  Can be used to organize log statements.
     * @param msg The actual message to be logged. The actual message to be logged.
     * @param tr If an exception was thrown, this can be sent along for the logging
facilities
     *           to extract and print useful information.
     */
   @Override
   public void println(int priority, String tag, String msg, Throwable tr) {


       String priorityStr = null;

       // For the purposes of this View, we want to print the priority as readable
text.
       switch(priority) {
           case android.util.Log.VERBOSE:
               priorityStr = "VERBOSE";
               break;
           case android.util.Log.DEBUG:
               priorityStr = "DEBUG";
               break;
           case android.util.Log.INFO:
               priorityStr = "INFO";
               break;
           case android.util.Log.WARN:
               priorityStr = "WARN";
               break;
```

```
            case android.util.Log.ERROR:
                priorityStr = "ERROR";
                break;
            case android.util.Log.ASSERT:
                priorityStr = "ASSERT";
                break;
            default:
                break;
        }

        // Handily, the Log class has a facility for converting a stack trace into a
usable string.
        String exceptionStr = null;
        if (tr != null) {
            exceptionStr = android.util.Log.getStackTraceString(tr);
        }

        // Take the priority, tag, message, and exception, and concatenate as
necessary
        // into one usable line of text.
        final StringBuilder outputBuilder = new StringBuilder();

        String delimiter = "\t";
        appendIfNotNull(outputBuilder, priorityStr, delimiter);
        appendIfNotNull(outputBuilder, tag, delimiter);
        appendIfNotNull(outputBuilder, msg, delimiter);
        appendIfNotNull(outputBuilder, exceptionStr, delimiter);

        // In case this was originally called from an AsyncTask or some other off-UI
```

```
thread,
        // make sure the update occurs within the UI thread.
        ((Activity) getContext()).runOnUiThread( (new Thread(new Runnable() {
            @Override
            public void run() {
                // Display the text we just generated within the LogView.
                appendToLog(outputBuilder.toString());
            }
        })));

        if (mNext != null) {
            mNext.println(priority, tag, msg, tr);
        }
    }

    public LogNode getNext() {
        return mNext;
    }

    public void setNext(LogNode node) {
        mNext = node;
    }

    /** Takes a string and adds to it, with a separator, if the bit to be added
isn't null. Since
     * the logger takes so many arguments that might be null, this method helps cut
out some of the
     * agonizing tedium of writing the same 3 lines over and over.
     * @param source StringBuilder containing the text to append to.
     * @param addStr The String to append
     * @param delimiter The String to separate the source and appended strings. A
 tab or comma,
     *                  for instance.
     * @return The fully concatenated String as a StringBuilder
     */
    private StringBuilder appendIfNotNull(StringBuilder source, String addStr,
 String delimiter) {
        if (addStr != null) {
            if (addStr.length() == 0) {
                delimiter = "";
            }

            return source.append(addStr).append(delimiter);
        }
        return source;
    }

    // The next LogNode in the chain.
    LogNode mNext;

    /** Outputs the string as a new line of log data in the LogView. */
    public void appendToLog(String s) {
        append("\n" + s);
    }
 }
```

LogWrapper.java

```
/*
* Copyright (C) 2012 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.example.vimin.robot_v12.common.logger;

import android.util.Log;

/**
* Helper class which wraps Android's native Log utility in the Logger interface.
This way
* normal DDMS output can be one of the many targets receiving and outputting logs
simultaneously.
*/
public class LogWrapper implements LogNode {

   // For piping:  The next node to receive Log data after this one has done its
work.
```

```
   private LogNode mNext;


   /**
    * Returns the next LogNode in the linked list.
    */
   public LogNode getNext() {
       return mNext;
   }


   /**
    * Sets the LogNode data will be sent to..
    */
   public void setNext(LogNode node) {
       mNext = node;
   }


   /**
    * Prints data out to the console using Android's native log mechanism.
    * @param priority Log level of the data being logged.  Verbose, Error, etc.
    * @param tag Tag for for the log data.  Can be used to organize log statements.
    * @param msg The actual message to be logged. The actual message to be logged.
    * @param tr If an exception was thrown, this can be sent along for the logging
facilities
    *           to extract and print useful information.
    */
   @Override
   public void println(int priority, String tag, String msg, Throwable tr) {
       // There actually are log methods that don't take a msg parameter.  For now,
       // if that's the case, just convert null to the empty string and move on.
```

```
        String useMsg = msg;

        if (useMsg == null) {

            useMsg = "";

        }


        // If an exeption was provided, convert that exception to a usable string
and attach

        // it to the end of the msg method.

        if (tr != null) {

            msg += "\n" + Log.getStackTraceString(tr);

        }


        // This is functionally identical to Log.x(tag, useMsg);

        // For instance, if priority were Log.VERBOSE, this would be the same as
Log.v(tag, useMsg)

        Log.println(priority, tag, useMsg);


        // If this isn't the last node in the chain, move things along.

        if (mNext != null) {

            mNext.println(priority, tag, msg, tr);

        }

    }

}
```

MessageOnlyLogFilter.java

```
/*

* Copyright (C) 2013 The Android Open Source Project

*
```

```
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.example.vimin.robot_v12.common.logger;


/**
* Simple {@link LogNode} filter, removes everything except the message.
* Useful for situations like on-screen log output where you don't want a lot of
metadata displayed,
* just easy-to-read message updates as they're happening.
*/
public class MessageOnlyLogFilter implements LogNode {


   LogNode mNext;


   /**
    * Takes the "next" LogNode as a parameter, to simplify chaining.
    *
    * @param next The next LogNode in the pipeline.
    */
   public MessageOnlyLogFilter(LogNode next) {

       mNext = next;

   }


   public MessageOnlyLogFilter() {

   }


   @Override

   public void println(int priority, String tag, String msg, Throwable tr) {

       if (mNext != null) {

           getNext().println(Log.NONE, null, msg, null);

       }

   }


   /**

    * Returns the next LogNode in the chain.

    */

   public LogNode getNext() {

       return mNext;

   }


   /**

    * Sets the LogNode data will be sent to..

    */

   public void setNext(LogNode node) {

       mNext = node;

   }

}
```

SampleActivithyBase.java

```
/*
* Copyright 2013 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.example.vimin.robot_v12.common.activities;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

import com.example.vimin.robot_v12.common.logger.Log;
import com.example.vimin.robot_v12.common.logger.LogWrapper;

/**
* Base launcher activity, to handle most of the common plumbing for samples.
*/
public class SampleActivityBase extends FragmentActivity {
    public static final String TAG = "SampleActivityBase";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    protected  void onStart() {
        super.onStart();
        initializeLogging();
    }

    /** Set up targets to receive log data */
    public void initializeLogging() {
        // Using Log, front-end to the logging chain, emulates android.util.log
method signatures.
        // Wraps Android's native log framework
        LogWrapper logWrapper = new LogWrapper();
        Log.setLogNode(logWrapper);

        Log.i(TAG, "Ready");
    }
}
```

activity_device_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!-- Copyright (C) 2014 The Android Open Source Project

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

         http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   >

   <TextView
       android:id="@+id/title_paired_devices"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:background="#666"
       android:paddingLeft="5dp"
       android:text="@string/title_paired_devices"
       android:textColor="#fff"
       android:visibility="gone"
       />


   <ListView

       android:id="@+id/paired_devices"

       android:layout_width="match_parent"

       android:layout_height="wrap_content"

       android:layout_weight="1"

       android:stackFromBottom="true"

       />


   <TextView

       android:id="@+id/title_new_devices"

       android:layout_width="match_parent"

       android:layout_height="wrap_content"

       android:background="#666"

       android:paddingLeft="5dp"

       android:text="@string/title_other_devices"

       android:textColor="#fff"

       android:visibility="gone"

       />


   <ListView

       android:id="@+id/new_devices"

       android:layout_width="match_parent"

       android:layout_height="wrap_content"

       android:layout_weight="2"

       android:stackFromBottom="true"

       />
   <Button
       android:id="@+id/button_scan"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="@string/button_scan"
       />
</LinearLayout>
```

content_device_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2014 The Android Open Source Project

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

         http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
```

```
>


<TextView

    android:id="@+id/title_paired_devices"

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:background="#666"

    android:paddingLeft="5dp"

    android:text="@string/title_paired_devices"

    android:textColor="#fff"

    android:visibility="gone"

    />


<ListView

    android:id="@+id/paired_devices"

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:layout_weight="1"

    android:stackFromBottom="true"

    />


<TextView

    android:id="@+id/title_new_devices"

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:background="#666"

    android:paddingLeft="5dp"

    android:text="@string/title_other_devices"

    android:textColor="#fff"
```

```
        android:visibility="gone"
        />

    <ListView
        android:id="@+id/new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:stackFromBottom="true"
        />

    <Button
        android:id="@+id/button_scan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_scan"
        />
</LinearLayout>
```

activity_main.xml

```
<!--
 Copyright 2013 The Android Open Source Project

 Licensed under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0
 Unless required by applicable law or agreed to in writing, software

 distributed under the License is distributed on an "AS IS" BASIS,

 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

 See the License for the specific language governing permissions and

 limitations under the License.

 -->

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

   android:id="@+id/sample_main_layout"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   android:orientation="vertical">




   <ViewAnimator

       android:id="@+id/sample_output"

       android:layout_width="match_parent"

       android:layout_height="1px"

       android:layout_weight="1">


   <ScrollView

       android:layout_width="match_parent"

       android:layout_height="match_parent">




   <LinearLayout

               android:orientation="vertical"

               android:layout_width="match_parent"

               android:layout_height="wrap_content"
```

```
        android:baselineAligned="false">


        <Button

            android:id="@+id/Stop_Button"

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:layout_alignRight="@+id/Stop_Button"

            android:layout_centerHorizontal="true"

            android:text="Stop(X!)" />


        <TextView

                    android:id="@+id/accel_text"

                    android:layout_width="wrap_content"

                    android:layout_height="wrap_content"

                    android:layout_marginTop="10dp"

                    android:text="Accelerometer"

                    android:textAppearance="?android:attr/textAppearanceSmall" />


        <TextView

                    android:id="@+id/yText"

                    android:layout_width="wrap_content"

                    android:layout_height="wrap_content"

                    android:layout_alignLeft="@+id/accel_text"

                    android:layout_below="@+id/accel_text"

                    android:layout_marginTop="10dp"

                    android:text="Y"

                    android:textAppearance="?android:attr/textAppearanceSmall" />


                <TextView
                    android:id="@+id/GyroScope_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_below="@+id/yText"
                    android:layout_marginTop="10dp"
                    android:text="GyroScope"
                    android:textAppearance="?android:attr/textAppearanceSmall" />

                <TextView
                    android:id="@+id/zGText"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignRight="@id/zGText"
                    android:layout_below="@+id/GyroScope_text"
                    android:layout_marginTop="10dp"
                    android:text="Z"
                    android:textAppearance="?android:attr/textAppearanceSmall" />

                <TextView
                    android:id="@+id/displacement_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_below="@+id/zGText"
                    android:layout_marginTop="10dp"
                    android:text="Displacement"
                    android:textAppearance="?android:attr/textAppearanceSmall" />

                <TextView
                    android:id="@+id/aDist_Y"
```

```
                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:layout_below="@+id/displacement_text"

                android:layout_marginTop="10dp"

                android:text="Dist_Y: "

                android:textAppearance="?android:attr/textAppearanceSmall" />


            <TextView

                android:id="@+id/angle_Z"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:layout_below="@+id/aDist_Y"

                android:layout_marginTop="10dp"

                android:text="angle_Z: "

                android:textAppearance="?android:attr/textAppearanceSmall" />


        <CheckBox

            android:text="Use Mag"

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"

            android:id="@+id/cbMag"

            android:layout_below="@+id/angle_Z"

            android:layout_alignLeft="@+id/accel_text"/>


        <TextView

                android:id="@+id/mag_x"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:layout_below="@+id/angle_Z"
                android:layout_marginTop="20dp"
                android:text="mag_x: "
                android:textAppearance="?android:attr/textAppearanceSmall" />

            <TextView
                android:id="@+id/mag_y"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/mag_x"
                android:layout_marginTop="0dp"
                android:text="mag_y: "
                android:textAppearance="?android:attr/textAppearanceSmall" />

            <TextView
                android:id="@+id/mag_z"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/mag_y"
                android:layout_marginTop="0dp"
                android:text="mag_z: "
                android:textAppearance="?android:attr/textAppearanceSmall" />
            <Button
                android:id="@+id/alignMag"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:layout_below="@+id/mag_z"
                android:layout_centerHorizontal="true"
                android:text="Align Robot" />
            <Button

                android:id="@+id/Toggle_Mode"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:gravity="center"

                android:layout_below="@+id/alignMag"

                android:layout_centerHorizontal="true"

                android:text="Enable Tracking Mode" />


            <Button

                android:id="@+id/GPSRequestButton"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:gravity="center"

                android:layout_below="@+id/Toggle_Mode"

                android:layout_centerHorizontal="true"

                android:text="Request Location" />


            <TextView

                android:id="@+id/GPS_Text"

                android:layout_width="match_parent"

                android:layout_height="wrap_content"

                android:layout_alignParentLeft="true"

                android:layout_alignParentStart="true"

                android:layout_below="@+id/GPSRequestButton"

                android:gravity="center"

                android:text="Coordinates: "

                android:textSize="30sp" />
```

```
                <TextView
                    android:gravity="center"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:text="default_longitude"
                    android:id="@+id/Longitude"/>
                <TextView
                    android:gravity="center"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:text="default_latitude"
                    android:id="@+id/Latitude"/>
            </LinearLayout>

    </ScrollView>

    <fragment
        android:id="@+id/log_fragment"
        android:name="com.example.vimin.robot_v12.common.logger.LogFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</ViewAnimator>

<View
android:layout_width="match_parent"
android:layout_height="1dp"
android:background="@android:color/darker_gray" />
```

```
<FrameLayout

android:id="@+id/sample_content_fragment"

android:layout_width="match_parent"

android:layout_height="0px"

android:layout_weight="0.31" />


</LinearLayout>
```

content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

   xmlns:app="http://schemas.android.com/apk/res-auto"

   xmlns:tools="http://schemas.android.com/tools"

   android:id="@+id/content_main"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   android:paddingBottom="@dimen/activity_vertical_margin"

   android:paddingLeft="@dimen/activity_horizontal_margin"

   android:paddingRight="@dimen/activity_horizontal_margin"

   android:paddingTop="@dimen/activity_vertical_margin"

   app:layout_behavior="@string/appbar_scrolling_view_behavior"

   tools:context="com.example.vimin.robot_v12.MainActivity"

   tools:showIn="@layout/activity_main">


   <TextView

       android:layout_width="wrap_content"

       android:layout_height="wrap_content"

       android:text="Hello World!" />
```

```
</RelativeLayout>
```

device_name.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2014 The Android Open Source Project

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

         http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
-->
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:padding="5dp"
         android:textSize="18sp"
   />
```

Fragment_bloothchat_chat.xml

```
<?xml version="1.0" encoding="utf-8"?><!--
```

```
Copyright 2014 The Android Open Source Project

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

   <!--<ListView-->
       <!--android:id="@+id/in"-->
       <!--android:layout_width="match_parent"-->
       <!--android:layout_height="match_parent"-->
       <!--android:layout_weight="1"-->
       <!--android:stackFromBottom="true"-->
       <!--android:transcriptMode="alwaysScroll" />-->

   <ScrollView
       android:layout_width="match_parent"
```

```
        android:layout_height="match_parent">


        <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

            xmlns:tools="http://schemas.android.com/tools"

            android:layout_width="match_parent"

            android:layout_height="80dip"

            tools:context=".MainActivity">


            <ListView

                android:id="@+id/in"

                android:layout_width="match_parent"

                android:layout_height="match_parent"

                android:layout_weight="1"

                android:transcriptMode="alwaysScroll"/>


            <Button

                android:id="@+id/Forward_Button"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:layout_centerHorizontal="true"

                android:layout_below="@+id/in"

                android:text="Forward(W!)" />


            <Button

                android:id="@+id/Left_Button"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:layout_below="@+id/Forward_Button"

                android:layout_centerHorizontal="true"
                android:text="Left(A!)" />


            <Button
                android:id="@+id/Right_Button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/Left_Button"
                android:layout_centerHorizontal="true"
                android:text="Right(D!)" />


            <Button
                android:id="@+id/Back_Button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@+id/Right_Button"
                android:layout_centerHorizontal="true"
                android:text="Back(S!)" />



            <EditText
                android:id="@+id/edit_text_out"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_below="@+id/Back_Button"
                android:layout_weight="1" />


            <Button
                android:id="@+id/button_send"
                android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"

                android:layout_below="@+id/Back_Button"

                android:layout_gravity="bottom"

                android:layout_alignRight="@id/edit_text_out"

                android:text="@string/send" />


        </RelativeLayout>

    </ScrollView>

</RelativeLayout>
```

message.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Copyright (C) 2014 The Android Open Source Project


    Licensed under the Apache License, Version 2.0 (the "License");

    you may not use this file except in compliance with the License.

    You may obtain a copy of the License at


         http://www.apache.org/licenses/LICENSE-2.0


    Unless required by applicable law or agreed to in writing, software

    distributed under the License is distributed on an "AS IS" BASIS,

    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

    See the License for the specific language governing permissions and

    limitations under the License.

-->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"

         android:layout_width="match_parent"
```

```
          android:layout_height="wrap_content"

          android:padding="5dp"

          android:textSize="18sp"

    />
```

bluetooth_chat.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Copyright (C) 2014 The Android Open Source Project


    Licensed under the Apache License, Version 2.0 (the "License");

    you may not use this file except in compliance with the License.

    You may obtain a copy of the License at


         http://www.apache.org/licenses/LICENSE-2.0


    Unless required by applicable law or agreed to in writing, software

    distributed under the License is distributed on an "AS IS" BASIS,

    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

    See the License for the specific language governing permissions and

    limitations under the License.

-->

<menu xmlns:android="http://schemas.android.com/apk/res/android">


   <item

       android:id="@+id/secure_connect_scan"

       android:icon="@drawable/ic_action_device_access_bluetooth_searching"

       android:showAsAction="always"

       android:title="@string/secure_connect"/>
   <item
       android:id="@+id/insecure_connect_scan"
       android:showAsAction="never"
       android:title="@string/insecure_connect"/>

   <item
       android:id="@+id/discoverable"
       android:showAsAction="never"
       android:title="@string/discoverable"/>
</menu>
```

main.xml

```
<!--
 Copyright 2013 The Android Open Source Project

 Licensed under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
 -->
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/menu_toggle_log"
   android:showAsAction="always"
   android:title="@string/sample_show_log" />
</menu>
```

fragmentview_strings.xml

```
<!--
 Copyright 2013 The Android Open Source Project

 Licensed under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
-->
<resources>
   <string name="sample_show_log">Show Log</string>
   <string name="sample_hide_log">Hide Log</string>
</resources>
```

strings.xml

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
   <string name="app_name">Robot_v1.2</string>
   <string name="action_settings">Settings</string>
   <string name="title_activity_device_list">DeviceListActivity</string>

   <!--  BluetoothChat -->
   <string name="send">Send</string>
   <string name="not_connected">You are not connected to a device</string>
   <string name="bt_not_enabled_leaving">Bluetooth was not enabled. Leaving
Bluetooth Chat.</string>
   <string name="title_connecting">connecting...</string>
   <string name="title_connected_to">connected to <xliff:g
id="device_name">%1$s</xliff:g></string>
   <string name="title_not_connected">not connected</string>

   <!--  DeviceListActivity -->
   <string name="scanning">scanning for devices...</string>
   <string name="select_device">select a device to connect</string>
   <string name="none_paired">No devices have been paired</string>
   <string name="none_found">No devices found</string>
   <string name="title_paired_devices">Paired Devices</string>
   <string name="title_other_devices">Other Available Devices</string>
   <string name="button_scan">Scan for devices</string>

   <!-- Options Menu -->
   <string name="secure_connect">Connect a device - Secure</string>
```

```
    <string name="insecure_connect">Connect a device - Insecure</string>

    <string name="discoverable">Make discoverable</string>


</resources>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.vimin.robot_v12">


    <!-- Min/target SDK versions (<uses-sdk>) managed by build.gradle -->

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <uses-permission android:name="android.permission.BLUETOOTH" />


    <application

        android:allowBackup="true"

        android:icon="@mipmap/ic_launcher"

        android:label="@string/app_name"

        android:theme="@style/AppTheme">


        <activity

            android:name=".MainActivity"

            android:label="@string/app_name"

            android:configChanges="orientation|keyboardHidden">
            <intent-filter>

                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />

            </intent-filter>

        </activity>


        <activity

            android:name=".DeviceListActivity"

            android:configChanges="orientation|keyboardHidden"

            android:label="@string/select_device"

            android:theme="@android:style/Theme.Holo.Dialog"/>


    </application>


</manifest>

    <!--android:supportsRtl="true"-->
```

Appendix E: MSP430 Code

**Note for more in depth explanation of the code, please refer to the individual section regarding the corresponding block*

1. <u>Main Microcontroller</u>

headers.h

```
#define BLUETOOTH_CONTROL_PORT 0

#define BLUETOOTH_CONTROL_UART_EN 0


#define SENSOR_CONTROL_PORT 0

#define SENSOR_CONTROL_SPI_EN BIT3


#define ROVER_CONTROL_PORT 0

#define ROVER_CONTROL_SPI_EN BIT4


#define GPIO_PORT_1 0

#define GPIO_PORT_2 1


#define MAX_NUM_BYTES 30

#define MAX_QUEUE_ITEM_NUM 250


enum Movements {

    FORWARD,

    BACKWARD,

    LEFT,

    RIGHT,

    STOP

};


enum Logic {

    FALSE,

    TRUE

};


enum Flags {
    FLAG_IDLE,

    FLAG_ROVER_CONTROL,

    FLAG_CONTROL,

    FLAG_USCI_A0_TX,

    FLAG_USCI_A0_RX_FINISHED,

    FLAG_USCI_B0_TX,

    FLAG_USCI_B0_RX_FINISHED

};


char rxData[MAX_NUM_BYTES];

char txData[MAX_NUM_BYTES];

unsigned short index = 0, itemCount = 0, rear_index = 0, front_index = 0,

currentMove;

char currentDirection, char_queue[MAX_QUEUE_ITEM_NUM];

unsigned short isRoverReceived = FALSE, rover_dist_met = FALSE;

unsigned short isSensorReceived = FALSE, sensor_dist_met = FALSE;

unsigned short isCurrentCmdExecuted = TRUE, isEmergencyBrake = FALSE;

enum Flags flag;
```

GPIO_methods.h

```
// Clear all GPIO pins

void GPIO_Reset() {

    // Set P1.x and P2.x to output directions and clear them all

    P1DIR = 0xFF;

    P1OUT = 0x00;

    P2DIR = 0xFF;

    P2OUT = 0x00;

}
```

```
// GPIO Setup
void GPIO_Setup() {
    GPIO_Reset();

    P2DIR &= ~(BIT0 + BIT1); // Input direction to P2.0 and P2.1
    P2REN |= BIT0 + BIT1; // Add int. pullup/pulldown resistor to P2.0 and P2.1
    P2OUT |= BIT0 + BIT1; // Config int. resistors for pullup operation to P2.0 and
P2.1
    P2IES &= ~(BIT0 + BIT1); // Select low to high edge Interrupt on P2.0 and P2.1
    P2IFG &= ~(BIT0 + BIT1); // Clear the interrupt flags to ensure system of P2.0
and P2.1
}
```

queue.h

```
// Check if queue is empty
unsigned short isQueueEmpty() {
    return (itemCount == 0) ? TRUE : FALSE;
}

// Check if queue is full
unsigned short isQueueFull() {
    return (itemCount == MAX_QUEUE_ITEM_NUM) ? TRUE : FALSE;
}

// Clear queue
void Clear_Queue() {
    memset(char_queue, 0, MAX_QUEUE_ITEM_NUM);
    rear_index = 0;

    front_index = 0;

    itemCount = 0;

}


// Enqueue new data

void dataEnqueue(char data) {

    if (isQueueFull() == FALSE) {

        if (rear_index == MAX_QUEUE_ITEM_NUM)

            rear_index = 0;


        char_queue[rear_index++] = data;

        itemCount++;

    }

}


// view data in queue

char viewData(unsigned short index) {

       if (isQueueEmpty() == FALSE)

              return char_queue[index];

       else

              return 'N';

}


// Dequeue data

char dataDequeue() {

    char data;


       data = char_queue[front_index++];
```

```
        if (front_index == MAX_QUEUE_ITEM_NUM)
               front_index = 0;

        itemCount--;

    return data;
}
```

UART.h

```
// Bluetooth - UART Setup on USCI_A0
void UART_Setup() {
    // GPIO Setups
    P1DIR &= ~(BIT1 + BIT2);
    P1SEL = BIT1 + BIT2;     // P1.1 = RXD, P1.2=TXD
    P1SEL2 = BIT1 + BIT2;    // P1.1 = RXD, P1.2=TXD

    // USCI_A0 UART Setup
    UCA0CTL1 |= UCSSEL_2;    // SMCLK
    UCA0CTL1 &= ~UCSWRST;    // Initialize USCI state machine
    UCA0BR0 = 21;          // SMCLK/(21 + 1*256)  = around 57600
    UCA0BR1 = 1;
    UCA0MCTL = UCBRS0;       // Modulation UCBRSx = 1
}

// UART transfer data method
void UART_Transfer_Data(char txData[]) {
    unsigned short i, size;
    unsigned char temp;


    size = strlen(txData);


    for (i = 0; i < size; i++) {

        while ((UCA0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data

        temp = txData[i]; // Keep it for debug

        UCA0TXBUF = temp; // TX next character

        __delay_cycles(500);

    }


    while ((UCB0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data

}
```

SPI_Master_UCB.h

```
// Enable and disable SPI slave

void Enable_Disable_SPI_Slave(unsigned short port_num, unsigned short pin_num,

unsigned short enable) {

    if (enable == TRUE) {

        if (port_num == GPIO_PORT_1) { // Disable TX

            P1OUT |= pin_num;

        } else if (port_num == GPIO_PORT_2) {

            P2OUT |= pin_num;

        }

    } else {

        if (port_num == GPIO_PORT_1) { // Enable TX

            P1OUT &= ~pin_num;

        } else if (port_num == GPIO_PORT_2) {
            P2OUT &= ~pin_num;

        }

    }

    __delay_cycles(5000);

}


// SPI UCB Master setup

void SPI_Setup() {

    P1SEL  |= BIT5 + BIT6 + BIT7; // Dedicate P1.5, P1.6, P1.7

    P1SEL2 |= BIT5 + BIT6 + BIT7; // UCB0CLK, UCB0MISO, and UCB0MOSI respectively


    UCB0CTL0 |= UCMSB + UCMST + UCSYNC + UCMODE_0 + UCCKPL;

    // UCCKPL: Data is captured on the first UCLK edge and changed on the following
edge.

    // UCMSB: MSB first select

    // UCMST: Master mode

    // UCSYNC: Synchronous mode enable

    // UCMODE_0: 3-pin SPI

    UCB0CTL0 &= ~(UC7BIT  + UCCKPH); // 8-bit, Clock polarity select: The inactive
state is low.

    UCB0CTL1 |= UCSSEL_2;    // USCI Clock source select: SMCLK

    UCB0CTL1 &= ~UCSWRST;    // Software reset disabled. USCI reset released for
operation.

    UCB0BR0 = 0x10;      // Bit clock prescaler setting => 2MHz clock

    UCB0BR1 = 0x00;      // The 16-bit value of (UCB0BR0 + UCB0BR1 × 256) forms the
prescaler value.

}


// SPI transfer data method
```

```
void SPI_Transfer_Data(unsigned short port_num, unsigned short pin_num, char
txData[]) {
    unsigned short i, size;

    size = strlen(txData);

    Enable_Disable_SPI_Slave(port_num, pin_num, TRUE);

    for (i = 0; i < size; i++) {
        UCB0TXBUF = txData[i]; // Keep it for debug
        while ((UCB0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data
    }

    Enable_Disable_SPI_Slave(port_num, pin_num, FALSE);
}
```

main.c

```
#include <msp430.h>
#include <string.h>
#include <stdio.h>
#include "headers.h"
#include "GPIO_Methods.h"
#include "queue.h"
#include "UART.h"
#include "SPI_Master_UCB.h"

// reset all status flags
void Reset_Status_Flags() {
    isRoverReceived = FALSE; // reset rover

    rover_dist_met = FALSE;

    isSensorReceived = FALSE; // reset isSensorReceived

    sensor_dist_met = FALSE; // reset distance met

}


// Set status flags

void Set_Status_Flags() {

    isRoverReceived = TRUE; // reset rover

    rover_dist_met = TRUE;

    isSensorReceived = TRUE; // reset isSensorReceived

    sensor_dist_met = TRUE; // reset distance met

}


// Send command

void Send_CMD(char temp) {

    char* roverCMD;

    char* sensorCMD;


    P2IFG &= ~(BIT0 + BIT1); // Clear interrupt flags from P2.0 and P2.1 pins

    P2IE |= BIT0 + BIT1; // Enable interrupts for P2.0 and P2.1


    sensorCMD = (temp == 'W' || temp == 'S') ? "AAA!"

        : (temp == 'A' || temp == 'D') ? "GGG!" : (temp == 'X') ? "XXX!" : "LLL!";


    roverCMD = (temp == 'W') ? "WWW!" : (temp == 'A') ? "AAA!" : (temp == 'S') ?

"SSS!"

        : (temp == 'D' || temp == 'E') ? "DDD!" : (temp == 'X') ? "XXX!" : "LLL!";
    if(temp == 'X'){

        if (isEmergencyBrake == FALSE) {

            __delay_cycles(10000);

            SPI_Transfer_Data(ROVER_CONTROL_PORT, ROVER_CONTROL_SPI_EN, roverCMD);

        }

        while(isRoverReceived == FALSE && isEmergencyBrake == FALSE);

        UART_Transfer_Data("_R"); // For debugging


        if (isEmergencyBrake == FALSE) {

            __delay_cycles(10000);

            SPI_Transfer_Data(SENSOR_CONTROL_PORT, SENSOR_CONTROL_SPI_EN,

sensorCMD);

        }

        while(isSensorReceived  == FALSE && isEmergencyBrake  == FALSE );

        UART_Transfer_Data("_S"); // for debugging*/


    } else {

        if (temp != 'L') {

            __delay_cycles(10000);

            if (temp == 'E') {

                if (isEmergencyBrake == FALSE) {

                    SPI_Transfer_Data(SENSOR_CONTROL_PORT, SENSOR_CONTROL_SPI_EN,

txData);

                    memset(txData, 0, MAX_NUM_BYTES); // Clear TX buffer

                }


            } else {

                if (isEmergencyBrake == FALSE) {

                    SPI_Transfer_Data(SENSOR_CONTROL_PORT, SENSOR_CONTROL_SPI_EN,
sensorCMD);
                }
            }
            while(isSensorReceived  == FALSE  && isEmergencyBrake  == FALSE );
            UART_Transfer_Data("_S");
        }

        if (isEmergencyBrake == FALSE) {
            __delay_cycles(10000);
            SPI_Transfer_Data(ROVER_CONTROL_PORT, ROVER_CONTROL_SPI_EN, roverCMD);
        }
        while(isRoverReceived  == FALSE  && isEmergencyBrake  == FALSE );// Try
sending the command for 10 times before break-out
        UART_Transfer_Data("_R"); // For debugging
    }
}

// Main method
void main(void) {
    unsigned short i;

    WDTCTL = WDTPW | WDTHOLD;   // Stop watch-dog timer

    if (CALBC1_16MHZ != 0xFF) { // If calibration constant erased
        DCOCTL = 0;             // Select lowest DCOx and MODx settings
        BCSCTL1 = CALBC1_16MHZ; // Set range
        DCOCTL = CALDCO_16MHZ;  // Set DCO step + modulation
    }
    GPIO_Setup(); // Setup GPIO pins

    UART_Setup(); // UART Setup

    SPI_Setup(); // SPI Setup


    memset(txData, 0, MAX_NUM_BYTES); // Clear TX buffer

    memset(rxData, 0, MAX_NUM_BYTES); // Clear RX buffer

    IFG2 &= ~(UCA0TXIFG + UCA0RXIFG + UCB0TXIFG + UCB0RXIFG);

    IE2 &= ~(UCA0TXIE + UCB0RXIE); // Disable USCI_A0 TX ISR and USCI_B0 RX
interrupt

    P2IFG &= ~(BIT0 + BIT1);

    P2IE |= BIT0 + BIT1; // Enable interrupts for P1.0 and P1.1

    IE2 |= UCA0RXIE; // Enable USCI_A0 RX interrupt


    _enable_interrupts();

    flag = FLAG_IDLE;


    while (1) {

        if (flag == FLAG_USCI_A0_RX_FINISHED) { // If received data is from UART

            index = 0;


            if (strstr(rxData, "L")) { // Turn on lights

                Send_CMD('L');

                flag = FLAG_CONTROL;


            } else if (strstr(rxData, "Q")) { // Ask for queue information

                sprintf(txData, "%d_%d_%d_%c_%c!\n\r",

                    itemCount, front_index, rear_index, viewData(front_index),
viewData(rear_index - 1));

                UART_Transfer_Data(txData);
                flag = FLAG_CONTROL;

            } else if (strstr(rxData, "P")) { // Print queue
                for (i = front_index; i < rear_index; i++) {
                    sprintf(txData, "%c ", viewData(i));
                    UART_Transfer_Data(txData);
                }
                UART_Transfer_Data("!\n\r");
                flag = FLAG_CONTROL;

            } else {
                if (isQueueFull() == FALSE) {
                    if (strstr(rxData, "W")) { // Forward
                        dataEnqueue('W');
                        UART_Transfer_Data("_W"); // for debugging
                    } else if (strstr(rxData, "S")) { // Backward
                        dataEnqueue('S');
                        UART_Transfer_Data("_S"); // for debugging
                    } else if (strstr(rxData, "A")) { // Left
                        dataEnqueue('A');
                        UART_Transfer_Data("_A"); // for debugging
                    } else if (strstr(rxData, "D")) { // Right
                        dataEnqueue('D');
                        UART_Transfer_Data("_D"); // for debugging
                    } else if (strstr(rxData, "E")) {
                        memcpy(txData, rxData, strlen(rxData));
                        dataEnqueue('E');
                        UART_Transfer_Data("_E"); // for debugging
                    }
                }
                flag = FLAG_CONTROL;
            }
            memset(rxData, 0, MAX_NUM_BYTES); // Clear RX buffer
            //IE2 |= UCA0RXIE; // Re-enable USCI_A0 RX interrupt

        } else if (flag == FLAG_CONTROL) {
            if (isCurrentCmdExecuted == FALSE) { // If the current message is NOT
FINISHED yet,
                // If sensor (or rover) received the message and the distance is
met,
                if ((isSensorReceived && sensor_dist_met)) {
                    if (sensor_dist_met)
                        UART_Transfer_Data("_SF!\n\r"); // For debugging: Queue is
not empty
                    else if (rover_dist_met)
                        UART_Transfer_Data("_RF!\n\r"); // For debugging: Queue is
not empty

                    isCurrentCmdExecuted = TRUE; // go to the state where the
current cmd is FINISHED
                    flag = FLAG_CONTROL;
                }

            } else { // If the current message is FINISHED,
                Reset_Status_Flags();

                if (isQueueEmpty() == FALSE) {
                    Send_CMD(dataDequeue()); // Send new commands to rover and
```

```
sensor
                    isCurrentCmdExecuted = FALSE;
                    flag = FLAG_CONTROL;

                } else { // If the queue is empty, stay in FLAG_IDLE and wait for
new UART cmd
                    UART_Transfer_Data("  X!"); // For debugging
                    Send_CMD('X'); // Stop the rover
                    UART_Transfer_Data("_X!\n\r"); // For debugging
                    flag = FLAG_IDLE; // wait for new cmd from UART
                }
            }
            IFG2 &= ~(UCA0TXIFG + UCA0RXIFG + UCB0TXIFG + UCB0RXIFG);

        } else if (flag == FLAG_IDLE){ // In this flag, wait for new UART cmd
            if (isEmergencyBrake == TRUE) {
                isEmergencyBrake == FALSE;
                Reset_Status_Flags();
                isCurrentCmdExecuted = TRUE;
                index = 0;
                flag = FLAG_IDLE;
            }
        }
    }
}

// USCI_A0 and USCI_B0 Receive ISR
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {
    unsigned char temp;

    //while ((UCB0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data

    if (IFG2 & UCA0RXIFG) { // If it is RX interrupt of USCI_A0
        temp = UCA0RXBUF; // Store value in USCI_A0 RX buffer to temp
        UCA0TXBUF = temp; // TX next character

        while ((UCB0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data
        if (index < MAX_NUM_BYTES) {
            rxData[index++] = temp;
            if (temp == '!') { // if received char isn't carriage return
                flag = FLAG_USCI_A0_RX_FINISHED;

            } else if (temp == 'X') { // Emergency brake
                Clear_Queue(); // Clear queue
                Send_CMD('X'); // Send stop command
                memset(rxData, 0, MAX_NUM_BYTES);
                Set_Status_Flags();
                isEmergencyBrake = TRUE;
            }

        } else {
            flag = FLAG_IDLE;
        }
        IFG2 &= ~UCA0RXIFG; // Clear RX interrupt flag of USCI_A0
    }
}
```

```
// Port 2 ISR

#pragma vector = PORT2_VECTOR // Interrupt vector

__interrupt void Port_2_ISR(void) { // ISR

    if (P2IFG & BIT0) { // If the interrupt flag is from P2.0

        P2IFG &= ~BIT0; // Clear interrupt flag


        if (!isSensorReceived) {

            isSensorReceived = TRUE;


        } else {

            sensor_dist_met = TRUE;

            P2IE &= ~BIT0;

        }


    } else if (P2IFG & BIT1) { // If the interrupt flag is from P2.1

        P2IFG &= ~BIT1; // Clear interrupt flag


        if (!isRoverReceived) {

            isRoverReceived = TRUE;


        } else {

            rover_dist_met = TRUE;

            P2IE &= ~BIT1;

        }

    }

}
```

2. Rover Microcontroller

headers.h

```
#define MAX_NUM_BYTES_RX 20

#define MAX_NUM_BYTES_TX 10

#define MAX_FORWARD_BACKWARD_MOTOR_COUNT 5000

#define MAX_LEFT_RIGHT_MOTOR_COUNT 1000

#define LEFT_MOTOR_ON_TIME 20000

#define LEFT_MOTOR_OFF_TIME 40000


enum Flags {

    FLAG_IDLE,

    FLAG_SPI_FINISH_RX,

    FLAG_ROVER_CONTROL

};


enum Boolean {

    FALSE,

    TRUE

};


enum Movements {

    FORWARD,

    BACKWARD,

    LEFT,

    RIGHT,

    STOP

};


enum Flags flag;

unsigned short index, currentMove;
unsigned int leftCount = 0, rightCount = 0;

unsigned short isLeftMotorOn = FALSE;

char rxData[MAX_NUM_BYTES_RX];

char txData[MAX_NUM_BYTES_TX];


// Clear buffers method

void Clear_Buffers() {

    memset(rxData, 0, MAX_NUM_BYTES_RX);

    memset(txData, 0, MAX_NUM_BYTES_TX);

}
```

GPIO_methods.h

```
// Setup GPIO pins

void GPIO_Setup() {

    P1DIR |= 0xFF; // Set P1.0 to output direction

    P1OUT = 0x00; // Clear all outputs

    P2DIR |= 0xFF; // Set P1.0 to output direction

    P2OUT = 0x00; // Clear all outputs


    P2SEL &= ~(BIT6 + BIT7); // Set P2.6 and P2.7

    P2SEL2 &= ~(BIT6 + BIT7); // to GPIO pins

    P2OUT &= ~(BIT6 + BIT7); // Turn off LEDs

}


// Interrupt to the main uC

void Interrupt_Main_uC() {

    P1OUT &= ~BIT3;

    __delay_cycles(2000);
    P1OUT |= BIT3;

    __delay_cycles(2000);

}
```

SPI_UCB_Slave.h

```
void SPI_UCB_Slave_Setup() {

    P1DIR &= ~(BIT6 + BIT7 + BIT4 + BIT5);

    P1SEL |= BIT6 + BIT7 + BIT4 + BIT5; // Dedicate P1.6, P1.7, P1.4 and P1.5 for

    P1SEL2 |= BIT6 + BIT7 + BIT4 + BIT5; // UCB0MISO, UCB0MOSI, UCB0EN, and UCB0CLK
respectively

    UCB0CTL0 |= UCCKPL + UCMSB + UCSYNC + UCMODE_1;

    // UCCKPL: Data is captured on the first UCLK edge and changed on the following
edge.

    // UCMSB: MSB first select

    // UCSYNC: Synchronous mode enable

    // UCMODE_1: 4-pin SPI with UCxSTE active high: slave enabled when UCxSTE = 1

    UCB0CTL0 &= ~(UCMST + UC7BIT + UCCKPH);

    // Slave, 8-bit, Clock polarity select: The inactive state is low.

    UCB0CTL1 |= UCSSEL_3;    // USCI Clock source select: SMCLK

    UCB0CTL1 &= ~UCSWRST;    // Software reset disabled. USCI reset released for
operation.

    UCB0BR0 = 0x10;      // Bit clock prescaler setting

    UCB0BR1 = 0x00;      // The 16-bit value of (UCB0BR0 + UCB0BR1 × 256) forms the
prescaler value.

}
```

rover_control.h

```
// Initialize Rover Control

void Initialize_Rover() {

    leftCount = 0; // reset counts

    rightCount = 0;

}


// Drive forward method

void Drive_Forward() {

    P2OUT |= BIT3; // Enable channels 1, 2, 3, and 4 of 1st L293D

    P2OUT &= ~(BIT1 + BIT5); // Control 1st L293D

    P2OUT |= BIT2 + BIT4;

    if (isLeftMotorOn == TRUE) {

        P2OUT |= BIT0;

    } else if (isLeftMotorOn == FALSE) {

        P2OUT &= ~BIT0;

    }

}


// Drive backward method

void Drive_Backward() {

    P2OUT |= BIT0 + BIT3; // Enable channels 1, 2, 3, and 4 of 1st L293D

    P2OUT |= BIT1 + BIT5; // Control 1st L293D

    P2OUT &= ~(BIT2 + BIT4);

}


// Turn left method

void Drive_Left() {

    P2OUT |= BIT0 + BIT3;   // Enable channels 1, 2, 3, and 4 of 1st L293D
    P2OUT &= ~(BIT2 + BIT5);  // Enable channels 3 and 4 of 2nd L293D

    P2OUT |= BIT1 + BIT4;

}


// Turn right method

void Drive_Right() {

    P2OUT |= BIT0 + BIT3;   // Enable channels 1, 2, 3, and 4 of 1st L293D

    P2OUT |= BIT2 + BIT5;  // Enable channels 3 and 4 of 2nd L293D

    P2OUT &= ~(BIT1 + BIT4);

}


// Stop rover method

void Drive_Stop() {

    P2OUT &= ~(BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5);   // Disable all channels

of L293D

}


// Check if the received SPI command is valid

int Check_Valid_SPI_Command() {

    if (strstr(rxData, "W")) { // Forward

        currentMove = FORWARD;

    } else if (strstr(rxData, "S")) { // Backward

        currentMove = BACKWARD;

    } else if (strstr(rxData, "A")) { // Left

        currentMove = LEFT;

    } else if (strstr(rxData, "D")) { // Right

        currentMove = RIGHT;

    } else if (strstr(rxData, "X")) { // Stop

        currentMove = STOP;
    } else if (strstr(rxData, "L")) {

        P2OUT ^= (BIT6 + BIT7);

    } else {

        return FALSE;

    }


    return TRUE;

}


// Control the robot method

void Move_Robot() {

    if (currentMove ==  FORWARD) {

        Drive_Forward();

    } else if (currentMove ==  BACKWARD) {

        Drive_Backward();

    } else if (currentMove ==  LEFT) {

        Drive_Left();

    } else if (currentMove ==  RIGHT) {

        Drive_Right();

    } else if (currentMove ==  STOP) {

        Drive_Stop();

    }

}
```

main.c

```
#include <msp430.h>

#include <string.h>

#include <math.h>
```

```
#include "headers.h"

#include "GPIO_Methods.h"

#include "SPI_UCB_Slave.h"

#include "rover_control.h"


// Main method

void main() {

    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer


    if (CALBC1_16MHZ != 0xFF) { // If calibration constant erased

        DCOCTL = 0; // Select lowest DCOx and MODx settings

        BCSCTL1 = CALBC1_16MHZ; // Set range

        DCOCTL = CALDCO_16MHZ; // Set DCO step + modulation

    }


    // Configuration for GPIO pins and SPI slave

    GPIO_Setup();

    SPI_UCB_Slave_Setup(); // SPI slave setup


    // Timer A setup

    TACCR0 = LEFT_MOTOR_OFF_TIME; // PWM Period of 20 ms

    TACTL |= TASSEL_2 + MC_2 + ID_3; // SMCLK, up mode


    // Clear interrupt flags and enable interrupts

    P1IFG &= ~(BIT1); // Clear the interrupt flags to ensure system

    IFG2 &= ~(UCB0RXIFG + UCA0RXIFG + UCA0TXIFG + UCB0TXIFG);


    P1IE |= (BIT1);

    IE2 &= ~UCB0TXIE;
    IE2 |= UCB0RXIE; // Enable USCI0 RX interrupt


    // FLAG_IDLE: wait for SPI cmd

    currentMove = STOP;

    Move_Robot();

    flag = FLAG_IDLE;


    for (index = 0; index < 15; index++) {

        P1OUT |= BIT0; // turn on LED

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        P1OUT &= ~BIT0; // turn on LED

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);

        __delay_cycles(60000);
        __delay_cycles(60000);

    }


    index = 0;

    _enable_interrupts();


    while (1) {

        if (flag == FLAG_SPI_FINISH_RX) {


            index = 0;


            if (Check_Valid_SPI_Command()) { // Check if the received SPI command
is valid

                P1OUT &= ~BIT0; // turn on LED

                Interrupt_Main_uC(); // Send interrupt to the main uC

            } else {

                P1OUT |= BIT0; // turn on LED

            }


            Clear_Buffers(); // Clear RX and TX arrays

            flag = FLAG_ROVER_CONTROL;

            IFG2 &= ~UCB0RXIFG; // clear RX flag of SPI protocol

            IE2 |= UCB0RXIE; // enable RX interrupt of SPIC protocol

            Drive_Stop();

            __delay_cycles(5000);


            TACCTL0 |= CCIE;

        } else if (flag == FLAG_ROVER_CONTROL) {
            P1IFG &= ~(BIT1); // Clear the interrupt flags to ensure system

            P1IE |= (BIT1);

            Initialize_Rover(); // Initialize rover control

            Move_Robot(); // Move the robot


        } else if (flag == FLAG_IDLE) {

            TACCTL0 &= ~CCIE;

        }

    }

}


// UCA and UCB Receive ISR

#pragma vector=USCIAB0RX_VECTOR

__interrupt void USCI0RX_ISR(void) {

    char temp;


    while ((UCB0STAT & UCBUSY)); // Wait if line TX/RX module is busy with data


    if (IFG2 & UCB0RXIFG) { // SPI UCB receive ISR

        IFG2 &= ~UCB0RXIFG; // clear SPI interrupt flag

        temp = UCB0RXBUF;


        if (index < MAX_NUM_BYTES_RX) {

            if (temp == '!') {

                index = 0;

                IE2 &= ~(UCB0RXIE); // disable RX interrupt of SPI USB

                flag = FLAG_SPI_FINISH_RX;      // flag for finish receiving data

            } else {

                rxData[index++] = temp;
            }

        }

    }

}


// Timer A ISR

#pragma vector = TIMER0_A0_VECTOR

__interrupt void Timer_A_ISR (void) {

    TACTL &= ~TAIFG;


    if (isLeftMotorOn == FALSE) {

        TACCR0 = LEFT_MOTOR_ON_TIME;

        isLeftMotorOn = TRUE;

    } else if (isLeftMotorOn == TRUE) {

        TACCR0 = LEFT_MOTOR_OFF_TIME;

        isLeftMotorOn = FALSE;

    }

}
```

3. Sensor Microcontroller

headers.h

```
#define TRANSFER_BYTE_NUM 10

#define RECEIVE_BYTE_NUM 10


//I2C headers

void Init_I2C();

void I2C_Write_Init(char I2C_address);
void I2C_Read_Init(char I2C_address);

void I2C_Write(char I2C_address, char reg_address, char data[],

               unsigned short data_byte_num);

void I2C_Read(char I2C_address, char reg_address, short byte_num);


// HMC5983 headers

void HMC5983_Initialize();

void HMC5983_Direction();

unsigned short HMC5983_Self_Test();

void HMC5983_Read_Measurements();


int offset_check = 0;

int current_compass_dir = 0;

int direction = 0;


struct compass_bounds {

    int upper_bound;

    int lower_bound;

};


struct compass_bounds CB = {0,0};


//FLAGs

enum Flags

{

    FLAG_IDLE,

    FLAG_HMC5983_READY,

    FLAG_RECEIVING,

    FLAG_FINISH_RX,
    FLAG_TRANSFERRING,

    FLAG_FINISH_TX,

    FLAG_ASKED,

    FLAG_DIST,

    FLAG_SPI_FINISH_RX,

    FLAG_DIST_FINISH

};


enum Data_request

{

    IDLE, ACCELLEROMETER, GYROSCOPE, COMPASS, ALIGN_COMPASS

};


enum Coordinates

{

    x_msb, x_lsb, y_msb, y_lsb, z_msb, z_lsb

};


enum Logic

{

    FALSE, TRUE

};


int first_time = TRUE;

int bound_check_compass = FALSE;


enum Data_request data_request;

enum Flags flag;

//enum Data_request data_request;
```

```
char txData[TRANSFER_BYTE_NUM];

char rxData[RECEIVE_BYTE_NUM];

unsigned short index, rxData_len, txData_len;


void Clear_Buffers()

{

    memset(txData, 0, TRANSFER_BYTE_NUM);

    memset(rxData, 0, RECEIVE_BYTE_NUM);

}


void GPIO_Setup()

{

    P2DIR &= ~(BIT2 + BIT1); // Input direction to P1.0

    P2IE |= BIT2 + BIT1; // Enable interrupts for P1.0

    P2REN |= BIT2 + BIT1; // Add int. pullup/pulldown resistor to P1.3 and P1.2

    P2OUT |= BIT2 + BIT1; // Config int. resistors for pullup operation

    P2IES |= (BIT2 + BIT1); // Select low to high edge Interrupt on P1.3 and P1.2

    P2IFG &= ~(BIT2 + BIT1); // Clear the interrupt flags to ensure system

}



float distance_traveled = 0.0;

int dist_spi = 0;

int MPU6050_setting = 1;
```

SPI_UCA_Slave.h

```
// Initialize SPI communication which uses USCI_A (Slave mode)
```

```
void Init_SPI() {
    //P1DIR &= ~(BIT1 + BIT2 + BIT4 + BIT5);
    P1SEL |= BIT1 + BIT2 + BIT4 + BIT5; // Dedicate P1.1, P1.2, P1.4 and P1.5 for
    P1SEL2 |= BIT1 + BIT2 + BIT4 + BIT5; // UCA0MISO, UCA0MOSI, UCA0CLK, and UCA0EN
respectively

    UCA0CTL0 |= UCCKPL + UCMSB + UCSYNC + UCMODE_1;
    // UCCKPL: Data is captured on the first UCLK edge and changed on the following
edge.
    // UCMSB: MSB first select
    // UCSYNC: Synchronous mode enable
    // UCMODE_1: 4-pin SPI with UCxSTE active high: slave enabled when UCxSTE = 1
    UCA0CTL0 &= ~(UCMST + UC7BIT + UCCKPH);
    // Slave, 8-bit, Clock polarity select: The inactive state is low.
    UCA0CTL1 |= UCSSEL_3;   // USCI Clock source select: SMCLK
    UCA0CTL1 &= ~UCSWRST;   // Software reset disabled. USCI reset released for
operation.
    UCA0BR0 = 0x10;     // Bit clock prescaler setting
    UCA0BR1 = 0x00;     // The 16-bit value of (UCB0BR0 + UCB0BR1 × 256) forms the
prescaler value.
```

I2C_UCB.h

```
// Initialize I2C communication which uses USCI_B
void Init_I2C() {
    P1SEL |= BIT6 + BIT7; // Assign I2C pins to USCI_B0
    P1SEL2|= BIT6 + BIT7; // (BIT6 = UCB0SCL, BIT7 = UCB0SDA)

    UCB0CTL1 |= UCSWRST; // Enable SW reset
    UCB0CTL0 |= UCMST + UCMODE_3 + UCSYNC;

    // Master, I2C and Synchronous modes

    UCB0CTL0 &= ~(UCA10 + UCSLA10 + UCMM);

    // Own and slave addresses are 7-bit, single master environment

    UCB0CTL1 |= UCSSEL_2 + UCTR + UCSWRST;

    // Use SMCLK, TX mode

    UCB0CTL1 &= ~(UCTXNACK + UCTXSTP + UCTXSTT);

    // Acknowledge normally, no STOP generated, no START generated,

    UCB0BR0 = 40; //Bit clock prescaler setting.

    UCB0BR1 = 0;

    // The 16-bit value of (UCBxBR0 + UCBxBR1 × 256) forms the prescaler value.

    // SCL = 16MHz / 40 = 400 kHz

    UCB0CTL1 &= ~UCSWRST; // clear software reset to resume operation

}


// Initialization of the I2C Module for Write operation.

void I2C_Write_Init(char I2C_address){

    UCB0I2CSA  = I2C_address; // Slave Address

    UCB0CTL1 |= UCTR; // UCTR = 1 => Transmit Mode (R/W bit = 0)

    UCB0CTL1 |= UCSWRST;

    UCB0CTL1 &= ~UCSWRST;

    IFG2 &= ~UCB0TXIFG; // Disable TXIFG since should be set by UCTXSTT

    IE2 &= ~UCB0RXIE; // Disable Receive ready interrupt

    IE2 |= UCB0TXIE; // Enable Transmit ready interrupt

}


// Initialization of the I2C Module for Read operation.

void I2C_Read_Init(char I2C_address){

    UCB0I2CSA  = I2C_address; // Slave Address
    UCB0CTL1 &= ~UCTR; // UCTR = 0 => Receive Mode (R/W bit = 1)

    UCB0CTL1 |= UCSWRST;

    UCB0CTL1 &= ~UCSWRST;

    IFG2 &= ~UCB0RXIFG; // disable TXIFG since should be set by UCTXSTT

    IE2 &= ~UCB0TXIE; // disable Transmit ready interrupt+

    IE2 |= UCB0RXIE; // enable Receive ready interrupt

}


// I2C Write Method

void I2C_Write(char I2C_address, char reg_address, char data[], unsigned short

data_byte_num) {

    Clear_Buffers();


    while (UCB0STAT & UCBUSY); // Wait until I2C module has finished all

operations.


    // Store register address and data to Write_Buffer

    txData_len = data_byte_num + 1;

    txData[0] = reg_address;

    for (index = 1; index < data_byte_num + 1; index++)

        txData[index] = data[index - 1];


    index = 0;

    I2C_Write_Init(I2C_address); // Set flags/conditions for writing

    UCB0CTL1 |= UCTXSTT; // Generate start condition

    __bis_SR_register(LPM0_bits); // Enter LPM0 with interrupts

    UCB0CTL1 |= UCTXSTP; // I2C stop condition

    while(UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent

    // UCTXSTP automatically cleared after STOP is generated
```

```
}


// I2C Read Method

void I2C_Read(char I2C_address, char reg_address, short byte_num) {

    Clear_Buffers();


    while (UCB0STAT & UCBUSY); // Wait until I2C module has finished all operations


    index = 0;

    txData_len = 2;

    txData[0] = reg_address;

    I2C_Write_Init(I2C_address); // Set flags/conditions for writing

    UCB0CTL1 |= UCTXSTT; // Generate start condition

    __bis_SR_register(LPM0_bits); // Enter LPM0 w/ interrupts


    index = 0;

    rxData_len = byte_num;

    I2C_Read_Init(I2C_address);  // Set flags/conditions for reading

    UCB0CTL1 |= UCTXSTT; // Generate second start condition for reading

    while(UCB0CTL1 & UCTXSTT); // Start condition sent?

    __bis_SR_register(LPM0_bits); // Enter LPM0 w/ interrupts // comment this line
out

}
```

MPU6050.h

```
#ifndef MPU6050_H_

#define MPU6050_H_

#include <math.h>
```

```
#define MPU6050_SLAVE_ADDRESS     0x68 // address pin low (GND), default for
InvenSense evaluation board
#define MPU6050_RA_CONFIG           0x1A
#define MPU6050_RA_GYRO_CONFIG      0x1B
#define MPU6050_RA_ACCEL_CONFIG     0x1C

#define MPU6050_RA_ACCEL_XOUT_H     0x3B
#define MPU6050_RA_ACCEL_XOUT_L     0x3C
#define MPU6050_RA_ACCEL_YOUT_H     0x3D
#define MPU6050_RA_ACCEL_YOUT_L     0x3E
#define MPU6050_RA_ACCEL_ZOUT_H     0x3F
#define MPU6050_RA_ACCEL_ZOUT_L     0x40
#define MPU6050_RA_TEMP_OUT_H       0x41
#define MPU6050_RA_TEMP_OUT_L       0x42
#define MPU6050_RA_GYRO_XOUT_H      0x43
#define MPU6050_RA_GYRO_XOUT_L      0x44
#define MPU6050_RA_GYRO_YOUT_H      0x45
#define MPU6050_RA_GYRO_YOUT_L      0x46
#define MPU6050_RA_GYRO_ZOUT_H      0x47
#define MPU6050_RA_GYRO_ZOUT_L      0x48

//#define MPU6050_RA_MOT_DETECT_STATUS    0x61

#define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
#define MPU6050_RA_SIGNAL_PATH_RESET    0x68
#define MPU6050_RA_MOT_DETECT_CTRL      0x69
#define MPU6050_RA_USER_CTRL        0x6A
#define MPU6050_RA_PWR_MGMT_1       0x6B
```

```
#define MPU6050_RA_PWR_MGMT_2       0x6C

#define MPU6050_RA_BANK_SEL         0x6D

#define MPU6050_RA_MEM_START_ADDR   0x6E

#define MPU6050_RA_MEM_R_W          0x6F

//#define MPU6050_RA_DMP_CFG_1        0x70

//#define MPU6050_RA_DMP_CFG_2        0x71

#define MPU6050_RA_FIFO_COUNTH      0x72

#define MPU6050_RA_FIFO_COUNTL      0x73

#define MPU6050_RA_FIFO_R_W         0x74


#define SMP_RATE_DIV 0x19


#define TWOG 16384

#define DEGREE250 131

#define TIME_SAMPLE 0.019

#define DATA_SIZE 50

#define ANGLE_OFFSET 90

//

//


int accel_offset_arr[3] = { 0, 0, 0 };


char buffer[8];

//int debug;


char ADDRESS_DEFINED_ACCEL = 0x00;

char ADDRESS_DEFINED = 0x00;

unsigned short data_index = 0;

unsigned short dist_met = 0;
```

```
//AVERAGE and FINAL CALC values for ACCEL and GYRO

struct ACCEL_GYRO_FINAL_AVG

{

    float x;

    float y;

    float z;

};


void read_accel_gyro_MPU6050();

void write_MPU6050(char data[], unsigned short num_bytes, char address_write);

void initialize_MPU6050();

void ADDRESS_Assign(char register_val);

void write_accel_config();

void write_gyro_config();

void Accel_Gyro_data_Collection(int data[]);

long long_averaging(int data_temp[]);

long int_averaging(int data_temp[]);


struct ACCEL_GYRO_STORE

{

    int y[DATA_SIZE];

    int z[DATA_SIZE];

};


struct ACCEL_GYRO_STORE ACCEL_GYRO_RAW_DATA;

struct ACCEL_GYRO_FINAL_AVG ACCEL_GYRO_FINAL_AVG;


void write_accel_config()
```

```
{
    buffer[0] = 0x00;   // 0x00 = no self test and 2g sensitivity most sensitive
    write_MPU6050(buffer, 1, MPU6050_RA_ACCEL_CONFIG);
    __delay_cycles(60000);
}

void write_gyro_config()
{
    buffer[0] = 0x00;                   // 0x00 = not self test and 250 degree/s
    write_MPU6050(buffer, 1, MPU6050_RA_GYRO_CONFIG);
    __delay_cycles(60000);
}

void ADDRESS_Assign(char register_val)
{
    ADDRESS_DEFINED_ACCEL = register_val; //Initializing the Accel address with X
first
    ADDRESS_DEFINED = 0x00;
}

void initialize_MPU6050()
{
    char data[5];

    data[0] = 0x00; // Initialization
    I2C_Write(MPU6050_SLAVE_ADDRESS, MPU6050_RA_PWR_MGMT_1, data, 1); // Important
initialization : awake from sleep
}
```

```
void LPF_HPF_MPU6050_setting()

{

    unsigned short LPF = 0;

    unsigned short HPF = 0;

    char data[5];


    while (!HPF)

    {

        if (data_request == ACCELLEROMETER)

        {

            if (!LPF)

            {

                //write_accel_config();            //Sends Accel config address

                data[0] = 0x00;    //Sending First byte of LP filter for Accell :
CUT-OFF 5 hz

                LPF = 1;

            }

            else

            {

                data[0] = 0x06;    //Sends Second byte of LP filter for Accell : 19
ms time sampling

                HPF = 1;

            }

        }

        else if (data_request == GYROSCOPE)

        {

            if (!LPF)

            {

                write_gyro_config();                //Sends Gyro config address
                data[0] = 0x01;                                  //Sends first byte of HP
filter 5Hz
                LPF = 1;
            }
            else
            {
                data[0] = 0x00;                    //Sends second byte for GYRO HP
filter
                HPF = 1;
            }
        }
        else
        {
            HPF = 1;
            LPF = 1;
            if(data_request == COMPASS){
                HMC5983_Direction();
                current_compass_dir = direction;
                bound_check_compass = compass_default_settings();
            }else{
                HMC5983_Direction();
                if(abs(direction-dist_spi) < 10){
                    dist_met = TRUE;
                }
                bound_check_compass = 0;
            }


        }
        if (LPF && !HPF)
        {
            write_MPU6050(data, 1, MPU6050_RA_ACCEL_CONFIG); //Setting HPF for GYRO
0x01 sets 5 Hz cut off
        }
        else
        {
            write_MPU6050(data, 1, MPU6050_RA_CONFIG); //LPF setting for Accelerom
0x06
        }                                //GYRO 0x00 for 0.98 ms 0x06 for 18.6ms
        __delay_cycles(60000);
        __delay_cycles(60000);
    }
}

void write_MPU6050(char data[], unsigned short num_bytes, char address_write)
{
    I2C_Write(MPU6050_SLAVE_ADDRESS, address_write, data, num_bytes); //Could
probably take out this function
}

//.................................................................................
..........
//      READING ACCEL AND GYRO
FUNCTIONS......................................................
//.................................................................................
..........

void read_accel_gyro_MPU6050()    //Reads Accel and Gyro Data
```

```
{

    while (data_index < DATA_SIZE)
    {
        if (data_request == ACCELLEROMETER)
        {
            __delay_cycles(60000); //
            __delay_cycles(60000);
            __delay_cycles(60000);
            __delay_cycles(60000);
            __delay_cycles(60000);
            __delay_cycles(4000); //Configured at 19ms data sampled : roughly 304k
cycles needed for Accel
        }
        else
        {
            __delay_cycles(15680); //Gyro configured at 0.98 ms : roughly 15.68k
cycles
        }
        I2C_Read(MPU6050_SLAVE_ADDRESS, ADDRESS_DEFINED_ACCEL, 6); //Start With X
Address and increment
        //__delay_cycles(60000);

        //ACCEL_GYRO_RAW_DATA.y[data_index] = ((rxData[0] << 8) + rxData[1]);
//X
        ACCEL_GYRO_RAW_DATA.y[data_index] = ((rxData[2] << 8) + rxData[3]);  //Y
        ACCEL_GYRO_RAW_DATA.z[data_index++] = ((rxData[4] << 8) + rxData[5]); //Z
    }
    data_index = 0;
```

```
}


void Accel_Gyro_Offsets(int data[],int data2[])

{

    int  i;

    long temp = 0;

    for (i = 0; i < DATA_SIZE; i++)

    {

        temp += (data[i]);

    }


    temp /= DATA_SIZE;




    if(data_request == ACCELLEROMETER){


           accel_offset_arr[0] = temp;

    }

    else

    {

           accel_offset_arr[1] = temp;

    }

}


void Accel_Gyro_data_Collection(int data[]) //Conditioning Data for real values

{

    long temp_x;
    if (data_request == ACCELLEROMETER)

    {

        temp_x = long_averaging(data);

        ACCEL_GYRO_FINAL_AVG.y = (float) temp_x / (4.0f * TWOG); //Trapezoidal

double integration hence divide by 4

        ACCEL_GYRO_FINAL_AVG.y = ACCEL_GYRO_FINAL_AVG.y * 9.8f * TIME_SAMPLE* 10;

//Multiplied by Accel data sampling time

    }

    else

    {

        temp_x = int_averaging(data);

        ACCEL_GYRO_FINAL_AVG.z = (float) temp_x / 131.0f * 10 * 0.00098;


//(TIME_SAMPLE - 0.0004f);


    }

}


long long_averaging(int data_temp[])

{

    long temp_vals = 0;


    unsigned short i = 0;

    for (i = 0; i < DATA_SIZE - 2; i++)

    {

        temp_vals += abs(abs(data_temp[i]) + abs(2 * data_temp[i + 1]) +

abs(data_temp[i + 2]) - 4 * abs(accel_offset_arr[0])) / 10;

    }                    //Total Formula : 0.25 * (D1 + 2*D2 + D3) * height
    return temp_vals;

}


long int_averaging(int data_temp[])

{

    long long temp_vals = 0;

    unsigned short i = 0;


    for (i = 0; i < DATA_SIZE - 1; i++)

    {

        temp_vals += abs(abs(data_temp[i]) + abs(data_temp[i + 1])- 2 *

abs(accel_offset_arr[1])) / 10;

        //Single trapezoidal integration : 0.5 * (D1 + D2) * height

    }


    return temp_vals;

}


/*/////////////////////////////////////////////////////////////////////////////////

////////////////////

 *

 * Data Collection loop for Gyro/Accel Functionality

 *


*./////////////////////////////////////////////////////////////////////////////////

////////////////////

 */


unsigned short SPI_flag(char data_buf[])
```

```
{
    unsigned short checking = TRUE;
    if (strstr(data_buf, "A"))
    {
        if(strstr(data_buf,"Z")) //Check for last command
        {
            dist_spi = 200;
        }
        else
        {
            dist_spi = 750;
        }
        data_request = ACCELLEROMETER;
        checking = TRUE;
    }
    else if (strstr(data_buf, "G"))
    {
        dist_spi = 25;
        data_request = COMPASS; //forces to use COMPASS
        checking = TRUE;
    }
    else if(strstr(data_buf, "E"))
    {

       dist_spi = atoi(data_buf + 1) + ANGLE_OFFSET; //Phone lags by 90 degrees
        //dist_spi = 200 + ANGLE_OFFSET;
       if(dist_spi > 360)
       {
           dist_spi = dist_spi - 360;
       }


       data_request = ALIGN_COMPASS;

       checking = TRUE;

    }

    else

    {

        //data_request = IDLE;

        checking = FALSE;

        //main_Finalize_All();

    }


    return checking;


}


void MPU6050_data_loop()

{

    if (data_request != COMPASS)

    {


        switch (data_request)

        {


        case IDLE:

            break;

        case ACCELLEROMETER:

            ADDRESS_Assign(MPU6050_RA_ACCEL_XOUT_H);

            read_accel_gyro_MPU6050();
            break;


        case GYROSCOPE:

            ADDRESS_Assign(MPU6050_RA_GYRO_XOUT_H);

            read_accel_gyro_MPU6050();

            break;

        }


    }


    if(offset_check == 1){

        if (data_request == ACCELLEROMETER)

        {

            Accel_Gyro_data_Collection(ACCEL_GYRO_RAW_DATA.y);

            if (ACCEL_GYRO_FINAL_AVG.y * 1000 > 75)

            {

                distance_traveled += (ACCEL_GYRO_FINAL_AVG.y * 1000);

            }

        }

        else

        {

           Accel_Gyro_data_Collection(ACCEL_GYRO_RAW_DATA.z);

           if(ACCEL_GYRO_FINAL_AVG.z > 0.03){

           distance_traveled += ACCEL_GYRO_FINAL_AVG.z;

           }

        }


        if (distance_traveled >= dist_spi && distance_traveled != 0)

        {
            //flag = FLAG_IDLE;

            //distance_traveled = 0;

            dist_met = 1;

            //IE2 |= UCA0RXIE;

        }

    }

}
```

HMC5983.h

```
/*Magnetometer Interfacing

*  Last Updated : 6/8/2017

**********************************/

// HMC5983 I2C Addresses

#define HMC5983_I2C_ADDRESS 0x1E


// HMC5983 SPI Read Multiple Registers

#define HMC5983_I2C_0_INCREMENT_ADDRESS 0x00

#define HMC5983_I2C_1_INCREMENT_ADDRESS 0x40


// Registers and their addresses

#define CONFIGURATION_A_REGISTER 0x00

#define CONFIGURATION_B_REGISTER 0x01

#define MODE_REGISTER 0x02

#define DATA_OUTPUT_X_MSB_REGISTER 0x03

#define DATA_OUTPUT_X_LSB_REGISTER 0x04

#define DATA_OUTPUT_Z_MSB_REGISTER 0x05

#define DATA_OUTPUT_Z_LSB_REGISTER 0x06

#define DATA_OUTPUT_Y_MSB_REGISTER 0x07
```

```
#define DATA_OUTPUT_Y_LSB_REGISTER 0x08

#define STATUS_REGISTER 0x09

/*#define IDENTIFICATION_A_REGISTER 0x10

#define IDENTIFICATION_B_REGISTER 0x11

#define IDENTIFICATION_C_REGISTER 0x12*/

#define TEMPERATURE_OUTPUT_MSB_REGISTER 0x31

#define TEMPERATURE_OUTPUT_LSB_REGISTER 0x32


// Enable/Disable Temperature sensor

#define TEMPERATURE_SENSOR_ENABLE 0x80

#define TEMPERATURE_SENSOR_DISABLE 0x00


// Number of samples averaged per measurement output configurations

#define SAMPLES_AVERAGED_1 0x00

#define SAMPLES_AVERAGED_2 0x20

#define SAMPLES_AVERAGED_4 0x40

#define SAMPLES_AVERAGED_8 0x60


// Data Output Rate Configurations

#define OUTPUT_RATE_0_75 0x00

#define OUTPUT_RATE_1_5 0x04

#define OUTPUT_RATE_3 0x08

#define OUTPUT_RATE_7_5 0x0C

#define OUTPUT_RATE_15 0x10

#define OUTPUT_RATE_30 0x14

#define OUTPUT_RATE_75 0x18

#define OUTPUT_RATE_220 0x1C


// Measurement Configurations
```

```
#define MEASUREMENT_MODE_NORMAL 0x00

#define MEASUREMENT_MODE_POSITIVE_BIAS 0x01

#define MEASUREMENT_MODE_NEGATIVE_BIAS 0x02

#define MEASUREMENT_MODE_TEMPERATURE_ONLY 0X03


// Gain Configurations

#define GAIN_1370 0x00

#define GAIN_1090 0x20

#define GAIN_820 0x40

#define GAIN_660 0x60

#define GAIN_440 0x80

#define GAIN_390 0xA0

#define GAIN_330 0xC0

#define GAIN_230 0xE0


// Operating Mode Configurations

#define OPERATING_MODE_CONTINUOUS 0x00

#define OPERATING_MODE_SINGLE 0x01

#define OPERATING_MODE_IDLE 0x02


#define DECLINATION_ANGLE 15 //SLO is 15 degree offset

//unsigned short isHMC5983Passed = 0;

int x_coor, y_coor, z_coor;




// Initialize HMC5983

void HMC5983_Initialize()

{
    char temp[5];


    temp[0] = TEMPERATURE_SENSOR_ENABLE | SAMPLES_AVERAGED_8 | OUTPUT_RATE_220

            | MEASUREMENT_MODE_NORMAL;

    I2C_Write(HMC5983_I2C_ADDRESS, CONFIGURATION_A_REGISTER, temp, 1);


    temp[0] = GAIN_390;

    I2C_Write(HMC5983_I2C_ADDRESS, CONFIGURATION_B_REGISTER, temp, 1);


    temp[0] = OPERATING_MODE_CONTINUOUS;

    I2C_Write(HMC5983_I2C_ADDRESS, MODE_REGISTER, temp, 1);


    __delay_cycles(50000);

    __delay_cycles(50000);


    //compass_default_settings();

}


float xa, ya, za;

// Read measurements

void HMC5983_Read_Measurements()

{

    __delay_cycles(60000);

    __delay_cycles(60000);

    I2C_Read(HMC5983_I2C_ADDRESS, DATA_OUTPUT_X_MSB_REGISTER, 6);

    x_coor = rxData[x_msb] << 8 | rxData[x_lsb];

    //y_coor = rxData[y_msb] << 8 | rxData[y_lsb];

    z_coor = rxData[z_msb] << 8 | rxData[z_lsb];
```

```
}


// HMC5983 Calculate direction
void HMC5983_Direction()
{
    HMC5983_Read_Measurements();

    int radian = atan(x_coor * 1.0 / z_coor) * 180 / 3.14;
    if (z_coor < 0)
    {
        direction = 270 - radian;
    }
    else if (z_coor > 0)
    {
        direction = 90 - radian;
    }
    else
    {
        if (x_coor < 0)
        {
            direction = 180.0;
        }
        else
        {
            direction = 0.0;
        }
    }
    direction += DECLINATION_ANGLE;

        if(direction < 0)

        {

            direction = 360 + direction;

        }

        else if(direction > 360)

        {

            direction = direction - 360;

        }

}


//Accumulate distance

void HMC5983_Total_Angle(int upper_bound_param, int lower_bound_param)

{


    int val = 0;

   val = (direction - current_compass_dir);

  switch(bound_check_compass){

      case 0:                //Default

          if(data_request == ALIGN_COMPASS)

          {

              if(abs(direction - dist_spi) < 5)//dist_spi) < 10)

              {

                  dist_met = TRUE;

              }

          }

          break;

      case 1:                //> 360 upperbound

          if( val < -324)
          {
              direction += 180;
          }
          else
          {
              direction -= 180;
          }
          break;
      case 2:                // < 0 lowerbound
          if(val > 324)
          {
             direction -= 180;
          }
          else
          {
             direction += 180;
          }
          break;
  }

  if(data_request == COMPASS){
      if(direction > upper_bound_param || direction < lower_bound_param)
      {
         dist_met = TRUE;
      }
  }
}

int compass_default_settings(){
  int high_temp = current_compass_dir + dist_spi;
  int low_temp = current_compass_dir - dist_spi;
  int boundary_cond = 0;

  if(high_temp > 360)
  {
      CB.upper_bound = high_temp - 360 + 180;
      CB.lower_bound = current_compass_dir - dist_spi - 180;
      boundary_cond = 1;
  }
  else if(low_temp < 0)
  {
      CB.upper_bound = high_temp + 180;
      CB.lower_bound = 360 + low_temp - 180;
      boundary_cond = 2;
  }
  else
  {
      CB.upper_bound = high_temp;
      CB.lower_bound = low_temp;
      boundary_cond = 0;
  }

  return boundary_cond;
}
```

main.c

```c
#include <msp430g2553.h>

#include <string.h>

#include <math.h>

#include "headers.h"

#include "SPI_UCA_Slave.h"

#include "I2C_UCB.h"

#include <stdlib.h>

#include "MPU6050.h"

#include "HMC5983.h"


int count = 0;

char data_request_buff[RECEIVE_BYTE_NUM];

unsigned short isExecuting = FALSE;


// Interrupt main uC: Telling master that received correct data

void Interrupt_Main_uC() {

    P2OUT &= ~BIT0;

    __delay_cycles(2000);

    P2OUT |= BIT0;

}


void main_Initialize_All() {

    initialize_MPU6050();

    HMC5983_Initialize();


    data_request = ACCELLEROMETER;

    LPF_HPF_MPU6050_setting();

    MPU6050_data_loop();

    Accel_Gyro_Offsets(ACCEL_GYRO_RAW_DATA.y, ACCEL_GYRO_RAW_DATA.y);
    Clear_Buffers();

    data_request = GYROSCOPE;

    LPF_HPF_MPU6050_setting();

    MPU6050_data_loop();

    Accel_Gyro_Offsets(ACCEL_GYRO_RAW_DATA.z, ACCEL_GYRO_RAW_DATA.z);

    offset_check = 1;


    first_time = FALSE;

}


void main_Finalize_All(){

    data_request = IDLE; //Send back to IDLE flag aka getting ready for SPI data
check

    dist_met = FALSE;                      //

    flag = FLAG_IDLE;                      //

    distance_traveled = 0;

    index = 0;

    Clear_Buffers();

    P1OUT |= BIT0;

    __delay_cycles(2000);

    P1OUT &= ~BIT0;                        //Just for debugging

    UCA0RXBUF = 0;

    IFG2 &= ~UCB0RXIFG;

    IE2 |= UCA0RXIE;

}


// Main method

void main()
```

```
{


    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer


    if (CALBC1_16MHZ != 0xFF)
    { // If calibration constant erased
        DCOCTL = 0;               // Select lowest DCOx and MODx settings
        BCSCTL1 = CALBC1_16MHZ; // Set range
        DCOCTL = CALDCO_16MHZ;   // Set DCO step + modulation
    }


    __delay_cycles(60000);
    P1DIR |= 0xFF;
    P1OUT = 0x00;
    P2DIR |= BIT0;
    P2OUT = 0x00;
    Init_SPI();
    Init_I2C();            //Initiating I2C
    //Interrupt_Main_uC(); //Telling master that received correct data
    _enable_interrupts();



    IFG2 &= (UCA0RXIFG + UCA0TXIFG + UCB0RXIFG + UCB0TXIFG);
    IE2 &= ~(UCA0TXIE);
    Clear_Buffers();
    index = 0;
    flag = FLAG_IDLE;//FLAG_SPI_FINISH_RX;
    data_request = IDLE;
```

```
  while(count < 15){     //Blink Fast to Indicate it is moving to FLAG_IDLE
      P1OUT |= BIT0;
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
      __delay_cycles(60000);
       P1OUT &= ~BIT0;
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       __delay_cycles(60000);
       count ++;
  }
 count = 0;



  IE2 |= UCA0RXIE;
```

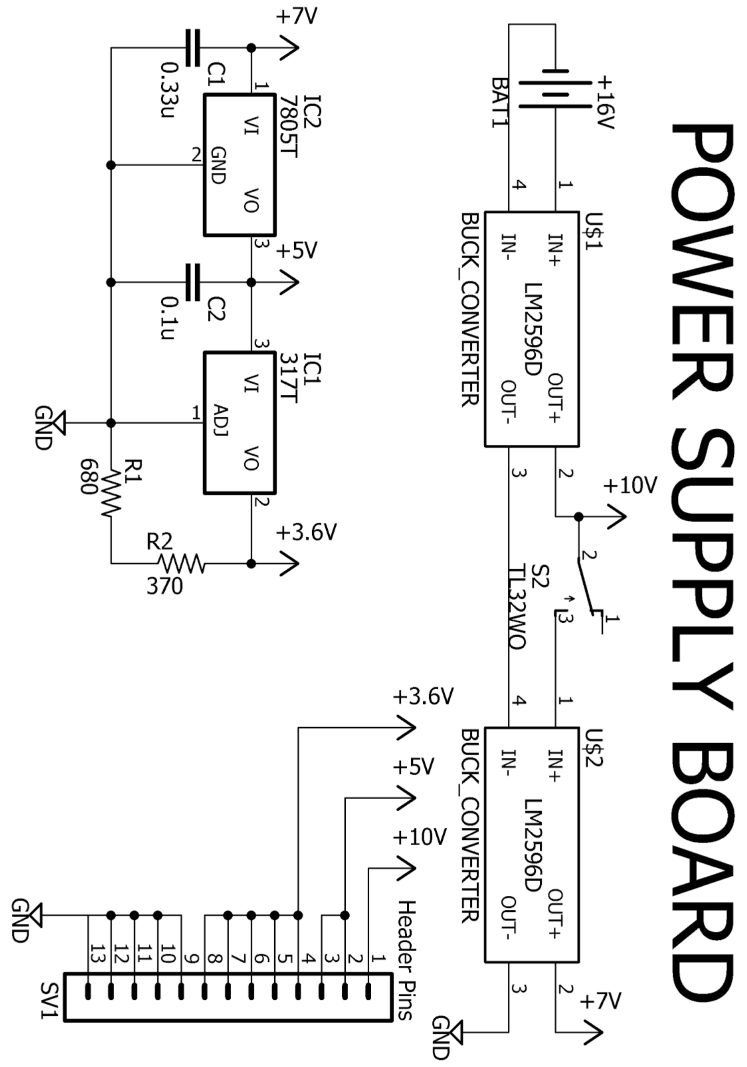
```
    while (1)
    {
        if (flag == FLAG_IDLE) {}
        else if (flag == FLAG_SPI_FINISH_RX) // After SPI RX finished The
data_request is found
        {

            if (first_time)
            {   // Initializes Accel/Gyro and Compass if not yet initialized
                main_Initialize_All();
            }
            if (SPI_flag(data_request_buff))
            {
                if(isExecuting){
                    flag = FLAG_DIST;
                }
                else{
                    isExecuting = TRUE;
                    P1OUT &= ~BIT0;
                    __delay_cycles(2000); //A Little bit of delay before tells
Master MSP it is about ready to start distance calculations
                    P1OUT |= BIT0;
                    __delay_cycles(2000);



                    //MPU6050_setting = TRUE;            //MPU6050 is setting is
ready
```

```
                    Clear_Buffers();

                    index  = 0;

                    flag = FLAG_DIST;                  // Changes to DISTANCE/ANGLE

state to initiate distance calculations


                    LPF_HPF_MPU6050_setting(); //Sets Digital Filtering for

Accel/Gyro



                    __delay_cycles(1000);




                    Interrupt_Main_uC();


                    IFG2 &= ~UCA0RXIFG;

                    IE2 |= UCA0RXIE;

                }


            }

            else

            {

               // Resets Because of bad command or stop command

                if(strstr(data_request_buff, "X")) //For resetting Sensors during

execution of calculations

                {

                    main_Finalize_All();

                    Interrupt_Main_uC();

                    isExecuting = FALSE;

                }

                else if(isExecuting){
                    flag = FLAG_DIST;

                }

                else

                {

                    main_Finalize_All();

                }


            }


        }

        else if (flag == FLAG_DIST && !dist_met)

        {

            if ((data_request != COMPASS) && (data_request != ALIGN_COMPASS))

            {

                MPU6050_data_loop(); //Accel/Gyro Distance/Angle Samples and
calculations

            }

            else

            {

                HMC5983_Direction();                    //Compass function

               // compass_default_settings()

                HMC5983_Total_Angle(CB.upper_bound, CB.lower_bound);

            }

        }

        if (dist_met == TRUE)

        {

            main_Finalize_All();

            Interrupt_Main_uC(); //Send interrupt to master telling that it
traveled the distance is met
```

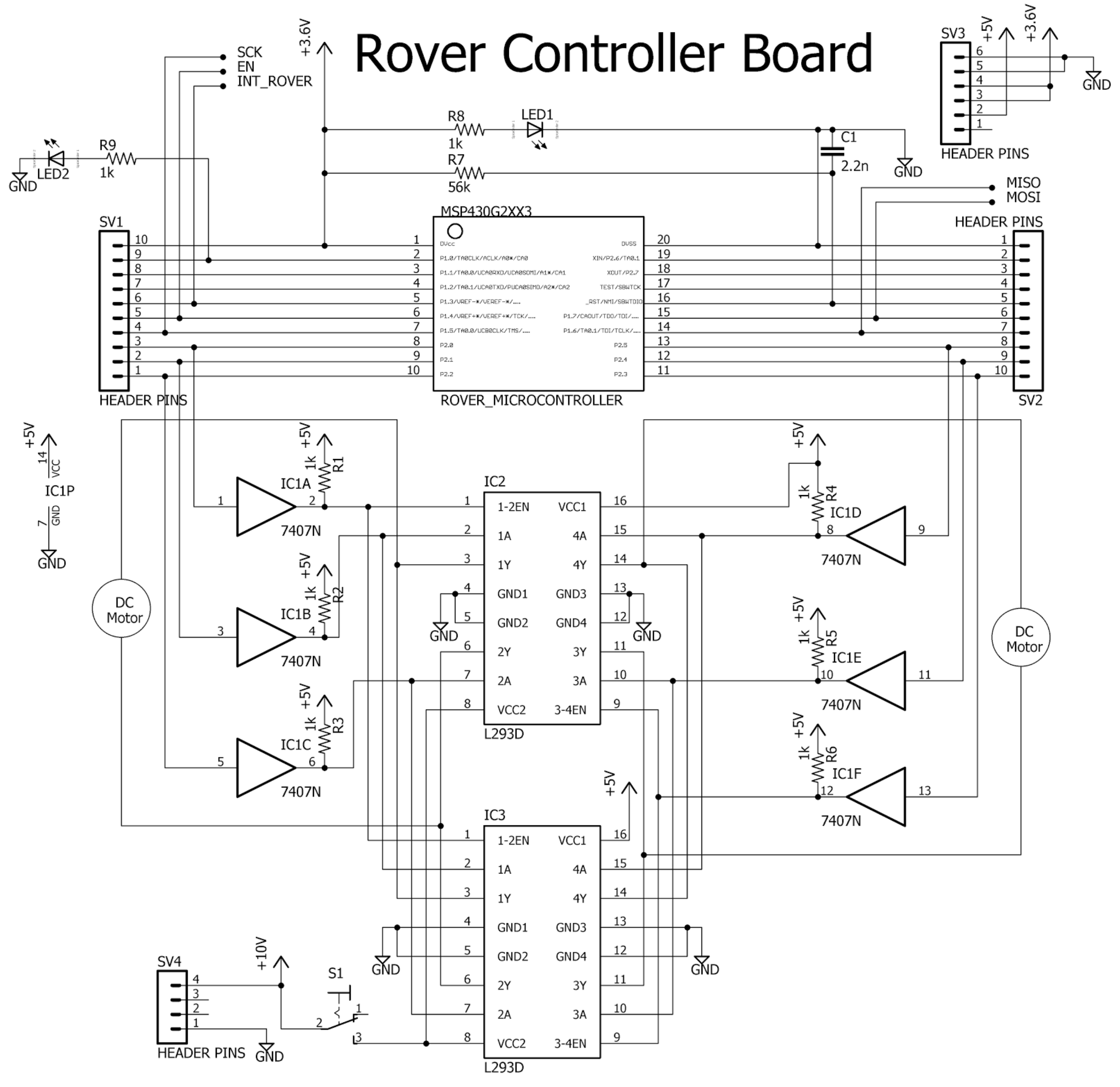
```
            isExecuting = FALSE;

        }

    }

}


// ISR: Transmit interrupt vector for USCI_A and USCI_B

#pragma vector = USCIAB0TX_VECTOR

__interrupt void USCI0TX_ISR(void)

{

    if (UCB0TXIFG & IFG2)

    { // If transfer flag in USCI_B,

        UCB0TXBUF = txData[index++];


        if (index == txData_len)

        {

            while (!(IFG2 & UCB0TXIFG))

                ;

            IE2 &= ~UCB0TXIE; // Disable TX Interrupts.

            IFG2 &= ~UCB0TXIFG; // Clear USCI_B0 TX int flag

            __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0

        }

    }

    else if (UCB0RXIFG & IFG2)

    { // If receive flag in USCI_B,

        rxData[index++] = UCB0RXBUF;


        if (index == rxData_len)

        {

            UCB0CTL1 |= UCTXSTP;
            IE2 &= ~UCB0RXIE; // Disable TX Interrupts.

            IFG2 &= ~UCB0RXIFG; // Clear USCI_B0 TX int flag

            __bic_SR_register_on_exit(LPM0_bits);      // Exit LPM0

        }

    }

}


// ISR: Receive interrupt vector for USCI_A and USCI_B

#pragma vector=USCIAB0RX_VECTOR

__interrupt void USCI0RX_ISR(void)

{

    char temp;


    if (IFG2 & UCA0RXIFG)

    { // SPI UCB receive ISR

        while ((UCB0STAT & UCBUSY));

        temp = UCA0RXBUF;


        if (index < RECEIVE_BYTE_NUM)

        {

            if (temp == '!' && !isExecuting)

            {

                //rxData_len = index;

                flag = FLAG_SPI_FINISH_RX; // flag for finish receiving data

                 //IE2 &= ~(UCA0RXIE);

                 IFG2 &= ~UCA0RXIFG;

                //MPU6050_setting = 0;

                memcpy(data_request_buff,rxData,RECEIVE_BYTE_NUM);

            }
            else if(temp == '!' && isExecuting){

                flag = FLAG_SPI_FINISH_RX;

                //IE2 &= ~(UCA0RXIE);

                memcpy(data_request_buff,rxData,RECEIVE_BYTE_NUM);

            }

            else

            {

                rxData[index++] = temp;

            }

        }

        if (index == RECEIVE_BYTE_NUM - 1)

        {

            index = 0;

        }

        IFG2 &= ~UCA0RXIFG;

    }

}
```

Appendix F: Schematics

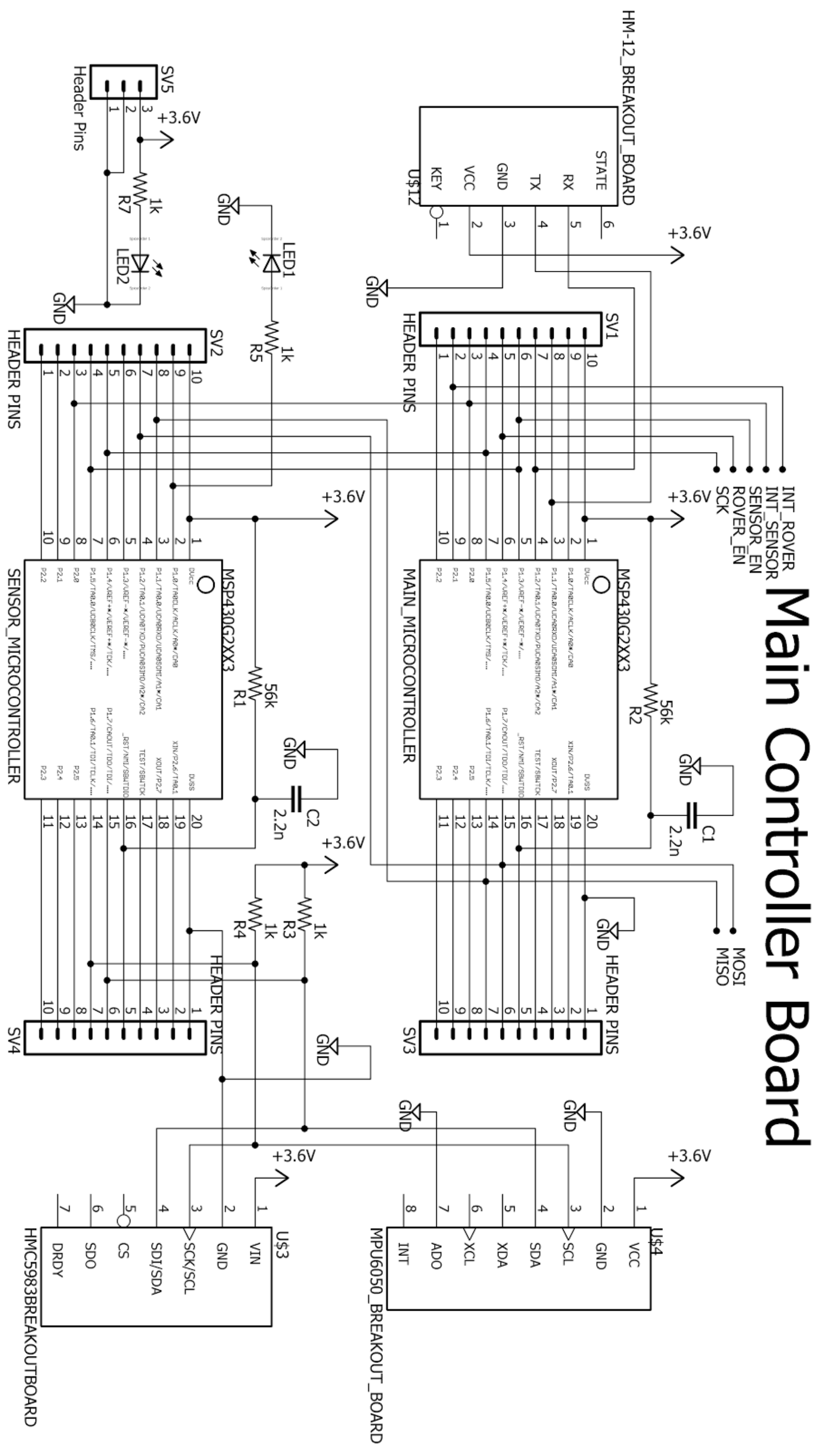
1. <u>Power Supply Board</u>


POWER SUPPLY BOARD
+16V
BAT1
U$1
IN+
IN-
OUT+
OUT-
LM2596D
BUCK_CONVERTER
+10V
S2
TL32WO
U$2
IN+
IN-
OUT+
OUT-
LM2596D
BUCK_CONVERTER
+7V
GND
+7V
IC2
7805T
VI
GND
VO
C1
0.33u
+5V
C2
0.1u
IC1
317T
VI
ADJ
VO
GND
R1
680
R2
370
+3.6V
+3.6V
+5V
+10V
Header Pins
GND
SV1

2. Rover Controller Board



3. Main Controller Board


Main Controller Board
HM-12_BREAKOUT_BOARD
U$12
STATE
RX
TX
GND
VCC
KEY
+3.6V
GND
SV1
HEADER PINS
SV2
HEADER PINS
SV3
HEADER PINS
SV4
HEADER PINS
SV5
Header Pins
R7 1k
LED2
LED1
R5 1k
MSP430G2XX3
MAIN_MICROCONTROLLER
MSP430G2XX3
SENSOR_MICROCONTROLLER
R2 56k
R1 56k
C1 2.2n
C2 2.2n
R3 1k
R4 1k
INT_ROVER
INT_SENSOR
SENSOR_EN
ROVER_EN
SCK
MOSI
MISO
U$3
VIN
GND
SCK/SCL
SDI/SDA
CS
SDO
DRDY
HMC5983BREAKOUTBOARD
U$4
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
MPU6050_BREAKOUT_BOARD