CENTROALGORITMI

Universidade do Minho
Escola de Engenharia

SELF-BALACING ROBOT

Authors:
Ailton Lopes A72852
José Oliveira A65308

Project Analysis
Project 1 MIEEICOM
2016/2017

Professor:
Adriano Tavares
José Mendes

Design Report

Contents

Design Report

Figure Index

Design Report

Table Index

Design Report

Hardware Specification

1. Hardware Resources

1.1. Development Board STM32F4-Discovery

Figure 1: STM32F4-Discovery

Features:

- The STM32F4DISCOVERY offers the following features:
- STM32F407VGT6 microcontroller featuring 32-bit ARM Cortex® -M4 with FPU core,
- 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
- On-board ST-LINK/V2 on STM32F4DISCOVERY or ST-LINK/V2-A on
- STM32F407G-DISC1
- ARM® mbed™ -enabled (http://mbed.org) with ST-LINK/V2-A only
- USB ST-LINK with re-enumeration capability and three different interfaces:
    1. virtual com port (with ST-LINK/V2-A only)
    2. mass storage (with ST-LINK/V2-A only)
    3. debug port
    a. Board power supply:
    4. Through USB bus
    5. External power sources (3 V and 5 V)
- LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
- MP45DT02 ST MEMS audio sensor omni-directional digital microphone
- CS43L22 audio DAC with integrated class D speaker driver
- LEDs
- Two push buttons (user and reset)
- USB OTG FS with micro-AB connector
- Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing.
- Comprehensive free software including a variety of examples, part of STM32CubeF4 package or STSW-STM32068 for legacy standard libraries usage.

1.1.1. LIS302DL **ST MEMS 3-axis accelerometer**

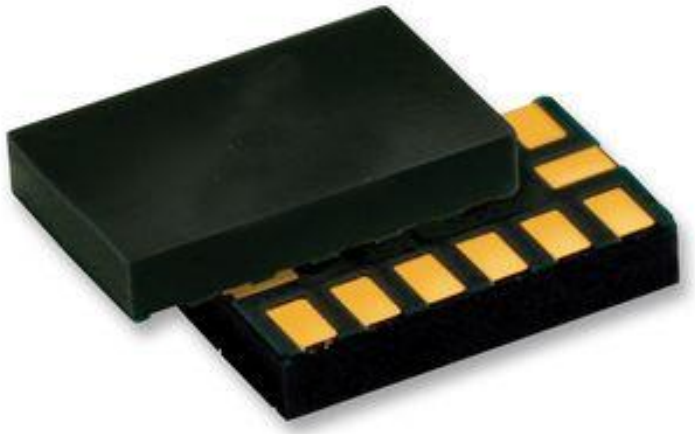

Figure 2: LIS302DL Accelerometer sensor

The LIS302DL is an ultra compact low-power three-axis linear accelerometer.It includes a sensing element and an IC interface able to provide the measured acceleration to the external world through I2C/SPI serial interface. The LIS302DL has dynamically user selectable full scales of ±2g/±8g and it is capable of measuring acceleration with an output data rate of 100 Hz or 400 Hz. The STM32F4 controls this motion sensor through the SPI interface**.**

Features

- Wide supply voltage, 1.71 V to 3.6 V
- Independent IOs supply (1.8 V) and supply
- voltage compatible
- Ultra-low power consumption
- I2C/SPI digital output interface
- 16-bit data output
- Programmable embedded state machines
- Embedded temperature sensor
- Embedded self-test
- Embedded FIFO
- 10000 g high shock survivability

Purpose:

Is used as input to controller that balance the robot(detect in real time the position of the Robot)

1.1.2. STM32F4 Board Schematics

In the schematics in in figure 5 is only present the pins of the board that are going the be used in the project

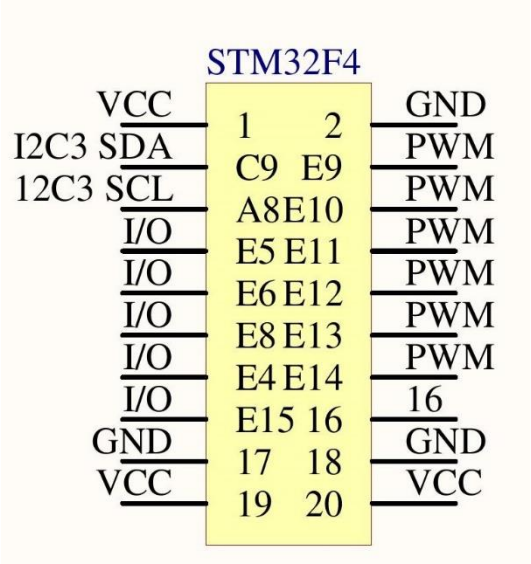

Figure 3: STM32F4 Board Schematics

1.1.3. Pin description

Pin	Name	Description
1	PC9 I2C3 SDA	Input from the IMU sensor (SDA)
2	PA8 I2C3 SCL	Input from the IMU sensor (SDA)
3	PA3 USART2 RX	Input from the Bluetooth Module
4	PA2 USART2 TX	Output from the Bluetooth Module
5	PE5	Input2 from the encoder B
6	PE6	Input1 from the encoder B
7	PE8	Input2 from the encoder A
8	PE4	Input1 from the encoder A
9	GND	Ground voltage
10	VCC	5 V
11	PE9 (PWM)	PIN to control the Motor
12	PE10(PWM)	PIN to control the Motor
13	PE11(PWM)	PIN to control the Motor
14	PE12(PWM)	PIN to control the Motor
15	PE13(PWM)	PIN to control the Motor
16	PE14(PWM)	PIN to control the Motor

Table 1: STM32F4 Board Pin Description

1.2. Bluetooth Module HC-05

1.2.1. Hardware features:

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baudrate

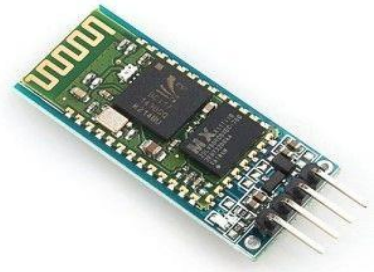

Figure 4: Bluetooth Module HC-05

1.2.2. Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1Parity: No parity, Data control: has supported baud rate: 9600, 19200, 38400, 57600, 115200, 230400, 460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

1.2.3. Purpose:

The Bluetooh Module HC-05 is used in the Robot system to communicate with the Desktop Application (send data or receive command).

1.2.4. Bluetooth module Pin assignment

PIN	Connected to:	Description
TX	STM32F4 – PA3 (USART2 RX)	Serial communication
RX	STM32F4 – PA2 (USART2 TX)	Serial communication
VCC	STM32F4 – VCC	Power Supply to the module

Table 2: Bluetooth module Pin assignment

1.2.5. Bluetooth Module Test Case

The following table shows the input and the expected output for the Bluetooth module

Input	Expected Output	Output
"Hello World"	"Hello World"	

Table 3: Bluetooth Test Case

1.3. IMU sensor c (accelerometer and gyroscope)

Figure 5: IMU sensor MPU-9255

1.3.1. Features

1.3.1.1. Gyroscope Features

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec and integrated 16-bit ADCs
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.2mA
- Sleep mode current: 8µA
- Factory calibrated sensitivity scale factor
- Self-test

1.3.1.2. Accelerometer Features

- Digital-output triple-axis accelerometer with a programmable full-scale range of ±2g, ±4g, ±8g and ±16g and integrated 16-bit ADCs
- Accelerometer normal operating current: 450µA
- Low power accelerometer mode current: 8.4µA at 0.98Hz, 19.8µA at 31.25Hz
- Sleep mode current: 8µA
- User-programmable interrupts
- Wake-on-motion interrupt for low power operation of applications processor
- Self-tes

1.3.2. Pin Description

Pin	Name	Description
1	VDD	Power supply (3.3V or 5V)
2	GND	Ground voltage
3	SCL	I2C serial Clock
4	SDA	I2C Serial Data
8	INT	Interrupt Digital Output

Table 4: IMU Sensor Pin Description

1.3.3. IMU Sensor Schematics

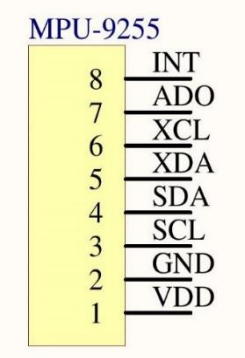

Figure 6: IMU sensor Schematics

1.3.4. I2C Serial Interface

In this project is used the I2C protocol to interface the Developmet board with the accelerometer sensor.

I2C is a two-wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I2C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master. The MPU-60X0 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400 kHz. The slave address of the MPU-60X0 is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin AD0.

I2C Communications Protocol

START (S) and STOP (P) Conditions

Communication on the I2C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until

the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below).

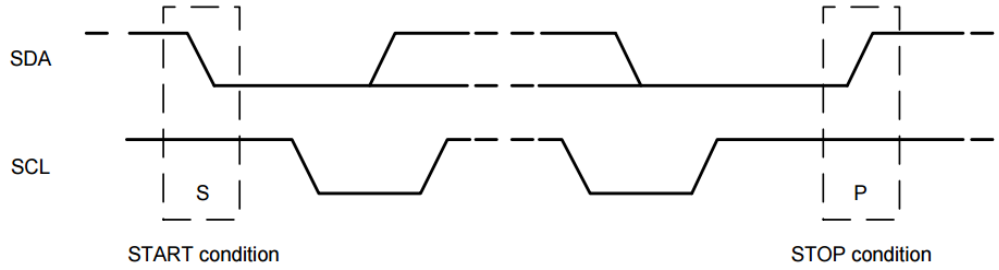

Figure 7: Start and stop condition

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device.

1.3.5. Purpose:

The MPU-9255 is used also as input to controller that balance the robot(detect in real time the position of the Robot using the accelerometer and gyroscope sensors)

1.3.6. IMU Sensor Pin assignment

PIN	Connected to:	Description/Type
VDD	STM32F4 – VCC	Power Supply (5V)
GND	STM32F4 – GND	GND
SCL	STM32F4 – PC9	I2C3 SDA (Serial Data)
SDA	STM32F4 – PA8	I2C3 SCL (Serial Clock)

Table 5: IMU Sensor Pin Assignmet

1.3.7. IMU Sensor Test Case

The following table shows the input and the expected output for IMU Sensors

Input	Expected Output	Output
Shake the sensor	LED On	

Table 6: IMU Sensor Test Case

1.4. L298N Motor Driver

Figure 8: Motor Driver

1.4.1. Description:

The L298N is an integrated monolithic circuit in a 15- lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

1.4.2. Block Diagram:

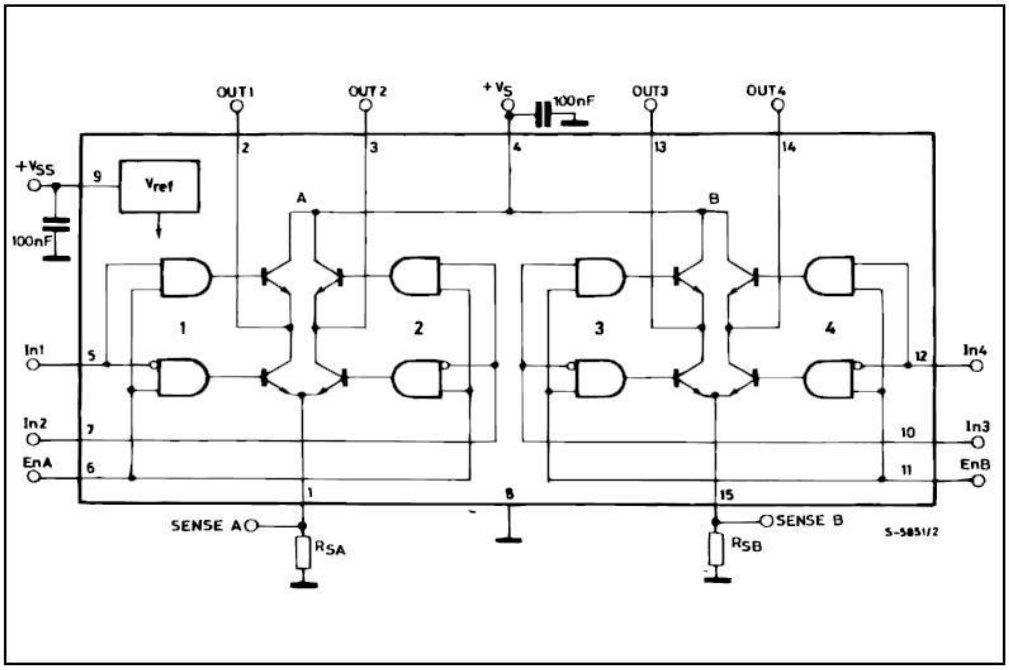

Figure 9: L298N Block Diagram

1.4.3. Pins Description

Pin	Description
VCC	5V
VS	Input Voltage (12V)
In1	Input to control Motor A
In2	Input to control Motor A
In3	Input to control Motor B
In4	Input to control Motor B
enA	Input to enable Motor A
enB	Input to enable Motor B
Out1	Motor A output +
Out2	Motor A output -
Out3	Motor B output +
Out4	Motor B output -
GND	Ground voltage

Table 7: Motor Driver Pin description

1.4.4. Motor Driver L298N Schematics

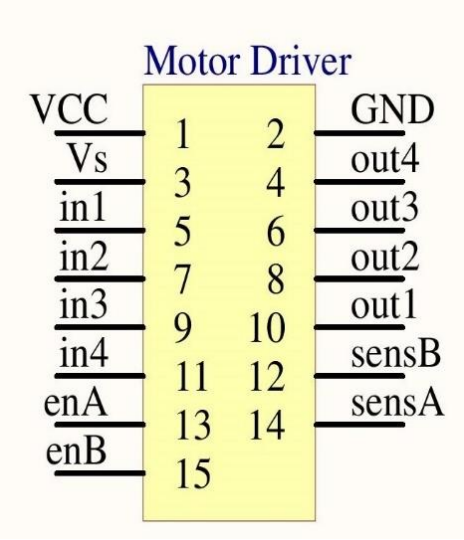

Figure 10: Motor with Encoder Schematics

1.4.5. Purpose:

The L298N is used to interface(control) the motors using the development board.

1.4.6. Motor Driver Pin assignment

Motor Driver L298N	Connected to:	Description/Type
VS	Battery + (12V)	-
In1	STM32F4 - PE14	PWM
In2	STM32F4 - PE13	PWM
In3	STM32F4 - PE12	PWM
In4	STM32F4 - PE11	PWM
enA	STM32F4 - PE10	Input
enB	STM32F4 – PE9	Input
GND	Battery –	-
Out1	Motor A M1	PWM
Out2	Motor A M2	PWM
Out3	Motor B M1	PWM
Out4	Motor B M2	PWM

Table 8: Motor Driver Pin assignment

1.4.7. Motor and Motor Driver Test Case

The following table shows the different combination of the Motor Driver and the expected output/state of the Motor.

Input	Expected Output	Output
ENA = 0	Motor A is disable	
ENA = 1 IN1 = 0 IN2 = 0	Motor A is stopped (brakes)	
ENA = 1 IN1 = 0 IN2 = 1	Motor A is on and turning backwards	
ENA = 1 IN1 = 1 IN2 = 0	Motor A is on and turning forwards	
ENA = 1 IN1 = 1 IN2 = 1	Motor A is stopped (brakes)	

Table 9: Motor Driver Test Case

1.5. JGA25-371 DC Gearmotor with Encoder

Figure 11: JGA25-371 DC Gearmotor with Encoder

1.5.1. Overview

This JGA25-371 DC gearmotor features an integrated encoder which provides a resolution of 12 counts per revolution, ensuring an accurate control of motor's speed.

1.5.2. Specifications

- Operating voltage: between 6 V and 24 V
- Nominal voltage: 12 V
- Free-run speed at 12 V: 126 RPM
- Free-run current at 12 V: 46 mA
- Stall current at 12 V: 1 A
- Stall torque at 12 V: 4.2 kg·cm
- Gear ratio: 1:34
- Reductor size: 21 mm
- Weight: 85 g

1.5.3. Pin description

Pin	Label	Description
1	OUTA	Hall sensor output A
2	OUTB	Hall sensor output B
3	M1	Motor +
4	M2	Motor -
5	VCC	Hall sensor VCC
6	GND	Hall sensor GND

Table 10: Motor with encoder Pin Description

1.5.4. Motor with Encoders schematics

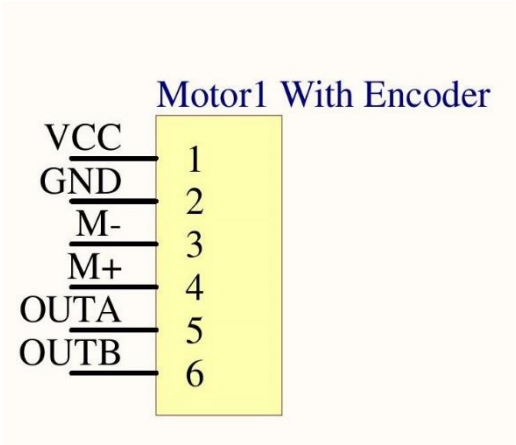

Figure 12: Motor with Encoder Schematics

1.5.5. Purpose:

The two motors is used to control the robot movements, and the encoder sensor is used as imput to the controller system by measuring the velocity of the robot.

1.5.6. Motor 1 with encoders Pin assignment

PIN	Connected to:	Description/Type
OUTA	STM32F4 – PE4	Output
OUTB	STM32F4 – PE8	Output
M1	Motor Driver – Out1	PWM
M2	Motor Driver – Out1	PWM
VCC	STM32F4 – VCC	Power Supply (5V)
GND	STM32F4 – GND	GND

Table 11: Motor 1 Pin assignmet

1.5.7. Motor 2 with encoders Pin assignment

PIN	Connected to:	Description/Type
OUTA	STM32F4 – PE6	Output
OUTB	STM32F4 – PE5	Output
M1	Motor Driver – Out1	PWM
M2	Motor Driver – Out1	PWM
VCC	STM32F4 – VCC	Power Supply (5V)
GND	STM32F4 – GND	GND

Table 12: Motor 2 Pin assignmet

2. System Hardware Schematics

The figure below shows how the hardware componets are connected with each other.

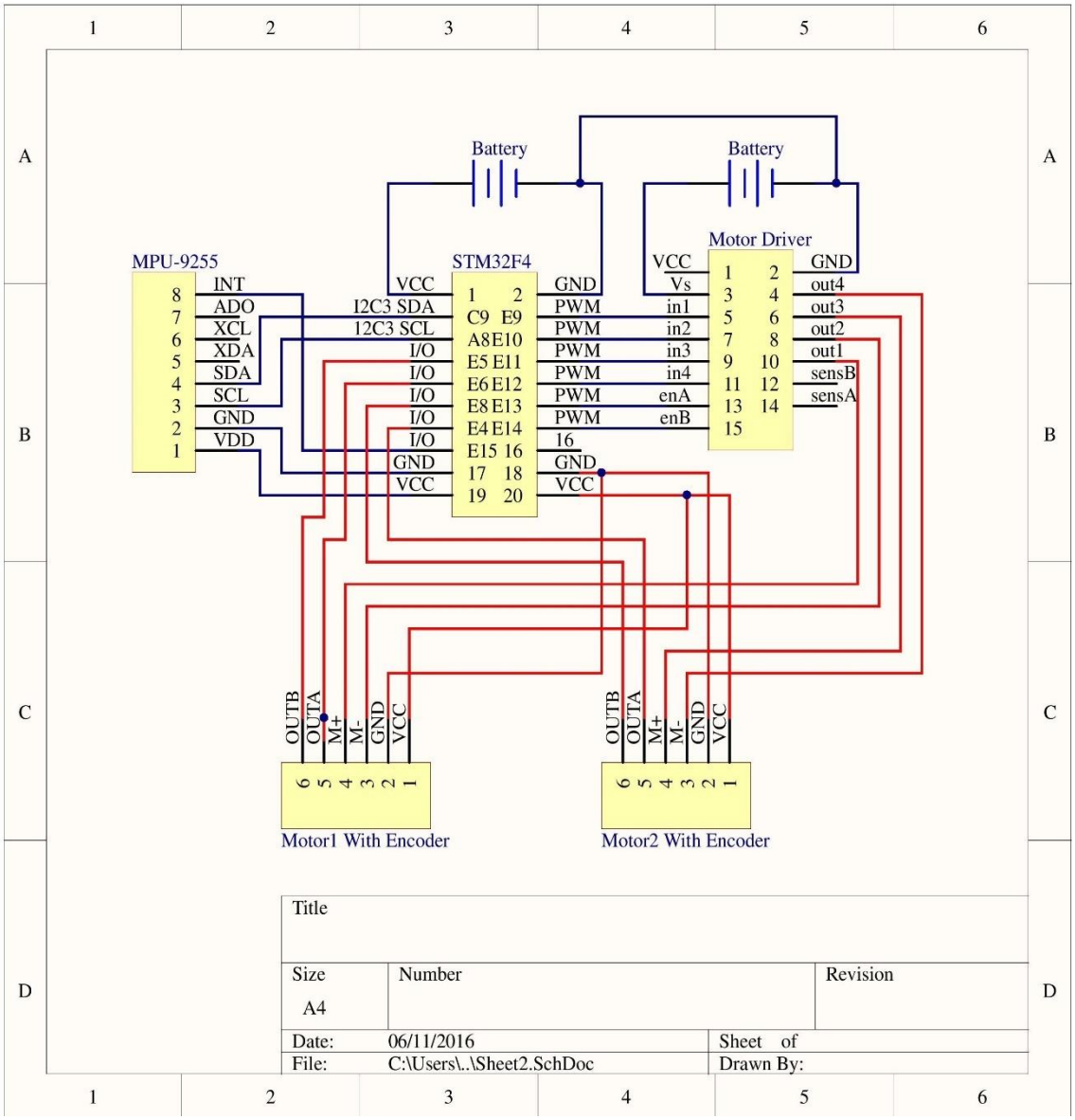

Figure 13: System Schematics

3. Peripherals and Cmmunication Protocols

The Figure 6 presents the Pheripherals used in this project and the communication protocol used.

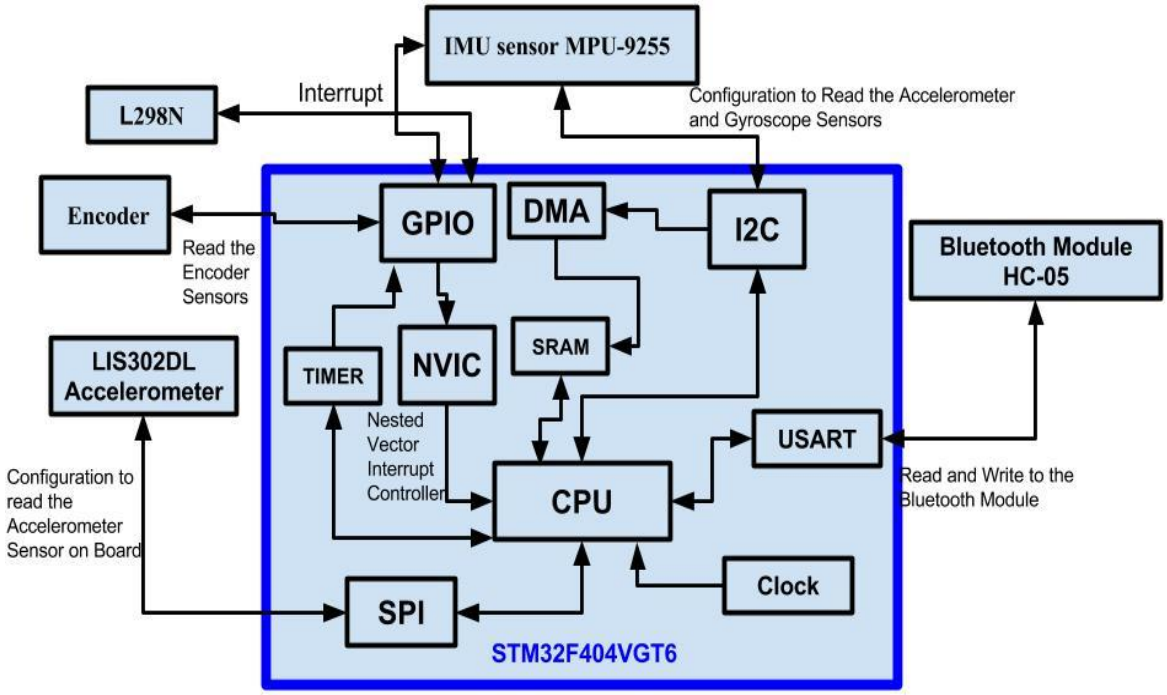

Figure 14: Pheripherals and Communication Protocol

Design Report

Software specification

1. Tools:

- Altium
- Matlab
- StarUML

2. Data Format

The Robot System and the Desktop application are constantely exchanging data using the Bluetooth communication. The following Table shows the extructure od the Data format used.

Type	Command	Source	Response	Description
PID Write	0x-val1-val2-val3	Desktop App	0xS	(1)
PID Read	1x	Desktop App	1x-val1-val2-val3	(2)
Status	2G	Desktop App	2-val-x	(3)
Movement	3x	Desktop App	3S	(4)

Table 13: Data Format

Description:

(1)
The first command is used when the used wants to write the value of the Gains (Kp, Kd and Ki) of one of the PID controller in the robot system.
The command sent by the Desktop application has the Format showed in the previous Table (0-x-val1-val2-val3).
The fist character **[0]** is to informs the Robot system that the command is the type PID Write, the second character **[x]** tells the Robot System which PID controller the User wants to change the gains (x = 1 – Balance Controller, x=2 Steering Controller and x = 3 – Distance Controller), and the values of the gains are the **val1** – Kp new value, **val2** – Kd new value and **val3** - Ki new value.

The Response from the Robot System to the PID Write command first character **[0]** informs the Desktop system that the commands is a response to the PID Write Command, the second one indicates witch PID Controller was Updated, and last character **[S]** tells that the gains were successful updated to the new values.

Example: Desktop Application send **11-0.4-10-14**, this commands tells the Robot System Update the Gains of the **Balance Controller** to **Kp = 0.4, Kd = 10 and Ki = 14**, after performing the command the Robot System will answer with **01S,** telling the Desktop Application the command was successful performed.

(2)

The second command first character **[1]** tells the Robot system that is a PID Read Command and the second (x) expecify witch Controler the user want to read the value of the gains (x = 1 – Balance Controller, x=2 Steering Controller and x = 3 – Distance Controller).

The Robot System answer to this command has the format similar to the PID Write Command (1x-val1-val2-val3). The first character **[1]** informs the Desktop Application that this command is an answer to a previous Read Command sent by the Destop Application, the second **[x]** character tells the App which PID Controller is the value received from (x = 1 – Balance Controller, x=2 Steering Controller and x = 3 – Distance Controller), and the others are the value of each Gain (**val1** – Kp value, **val2** – Kd value and **val3** - Ki value) separated by the character **[-].**

Example: The User Send the command **11** (command PID Read the gains of the Sterring controller) to the Robot System, the Robot System will answer with **11-Kp-Kd-Ki** and Kd, Kp and Ki are the atual value of the gains of the Sterring Controller.

(3)

The command number three is used when the used wants to know in that particular moment what is the Robot Current Status, the first character specify the type of the command, in this case is **[3]** for Status command, the second character **[G]** tells the Robot System that the User what to Get the Robot System status.

The Robot System answer to this command with **2-val-x,** where [2] is tells the App that is an answer to a Get Status command, [val] is the atual angle of the Robot sytem and the [x] character specify what the Robot System is performing at the moment (x = 1 - Just Robot Balancing, x=2 Robot Moving forward, x=3 Robot Moving Backward, x=4 – Robot Turning Right and x= 5 Robot Turning Left).

Example: User send the command **2G** asking the Robot Sytem to send the Robot Status, the Robot Sytem Answer with **2-5-4** telling the used that the Robot atual Angle is 5 degrees an that the Robot is Performing the Turning Right Command.

(4)

This type of command is sent when the used what to change the Robot System current Position, the command format is **3x,** where **[3]** specify the type of command (movement) and **[x]** specify the type of movement (1 forward, 2 backward, 3 turn right and 4 to turn left).

The Robot System answer with **3S**, where **[3]** tells that is an answer to the movement command, and **[S]** tells that is Succesfull performed the command.

Example: the user Send **31** telling the Robot System to move forward, after the Robot Sytem perform the command it answer with **3S** telling that the command was successful performed.

3. Controller Algorithm

To control the Robot is going to be used a PID controller. The use of the PID controller is one of the constraint of the project and for this project is going to be used three PID controllers to control the Robot.

3.1. PID Controller

A proportional–integral–derivative controller (PID controller) is a control loop feedback mechanism commonly used in industrial control systems. A PID controller continuously calculates an error value e(t) as the difference between a desired set point and a measured process variable and applies a correction based on proportional, integral, and derivative terms.

As said before the robot system has three PID Controllers:

3.1.1. Distance PID Controller

This controller is used to make to robot move forward or backward.

The output of this controller (desired angle) is used in the Balancing PID controller, the input of this controller is based on the movement chosen by the user.

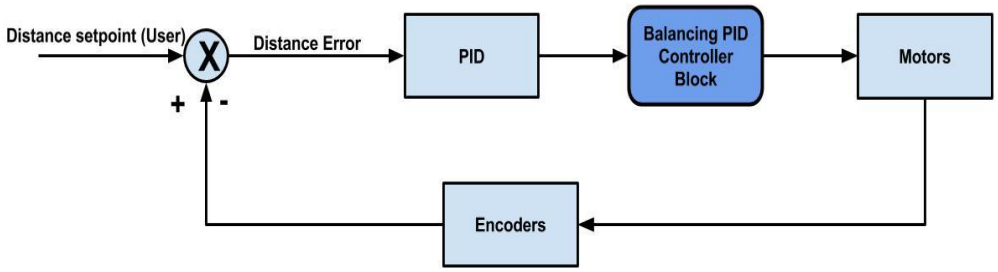

Figure 15: Distance PID controller

The distance can be using the following equation: $\boldsymbol{dist = \frac{dr+dl}{2}}$, where **dr** is the distance measured by the right encoder and **dl** the left encoder.

That way the **error** is equal to $\boldsymbol{dist_{Set} - dist}$, where $\mathbf{dist_{Set}}$ is the desired distance/position chosen by the user.

3.1.2. Balancing PID Controller

After the user desired angle has been calculated that value is used as input in this PID controller to calculate the input for the motor, this PID controller uses as input the robot current angle measured by the IMU sensors (accelerometer and gyroscope).

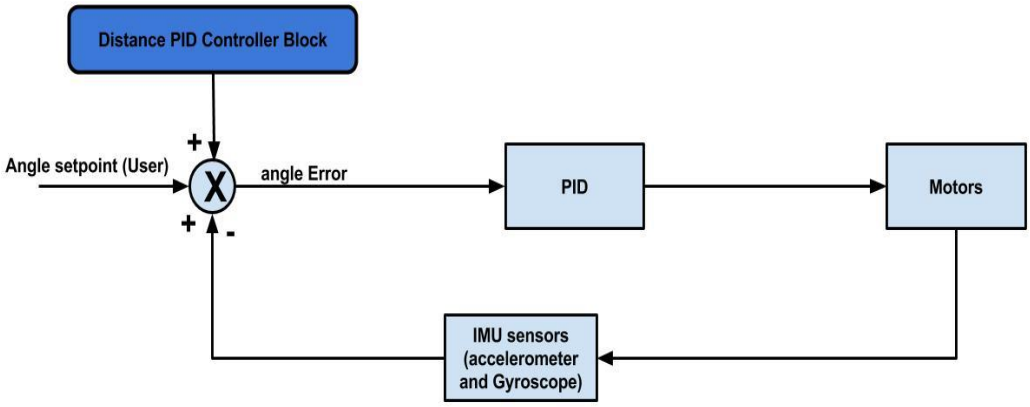

Figure 16: PID controller for the Robot

3.1.3. Steering PID Controller

This PID controller allows the used to control the robot maneuvers (turning right or left), the controller's input are the desired setpoint (turning angle) defined by the user and the actual yaw angle (heading) of the robot measured using the Encoders. Different from the Balancing PID controller this controller output will be applied different in the two motors as shown in the figure 6.

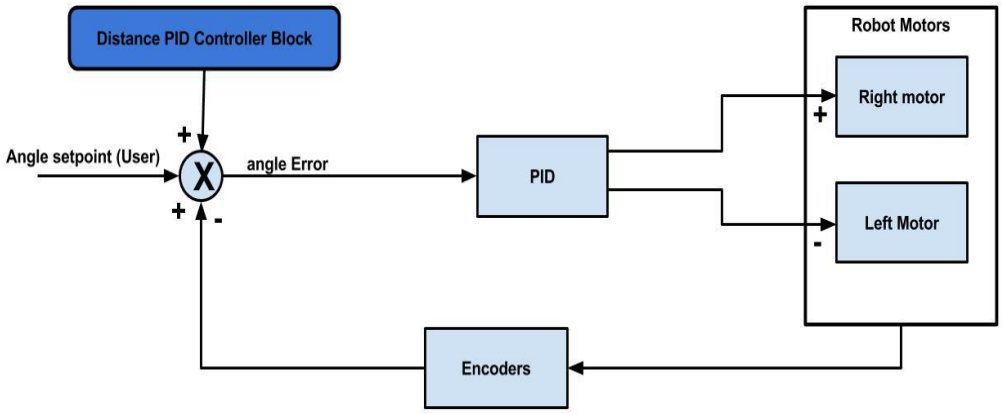

Figure 17: PID controller for the Robot

4. Angle measurement Algorithm

To calculate the Robot angle is used three sensor, two accelerometers and a gyroscope sensor, is used two acceleromenter because the IMU sensor used in the project include an accelerometer and the the development board also has an accelerometer, the two accelerometers readings will be used to give a better measurement by doing the fusion of the data from both accelerometer.

To combine the data from the accelerometer and the gyroscope is used a Filter and the output of the filter it's the robot's current angle.

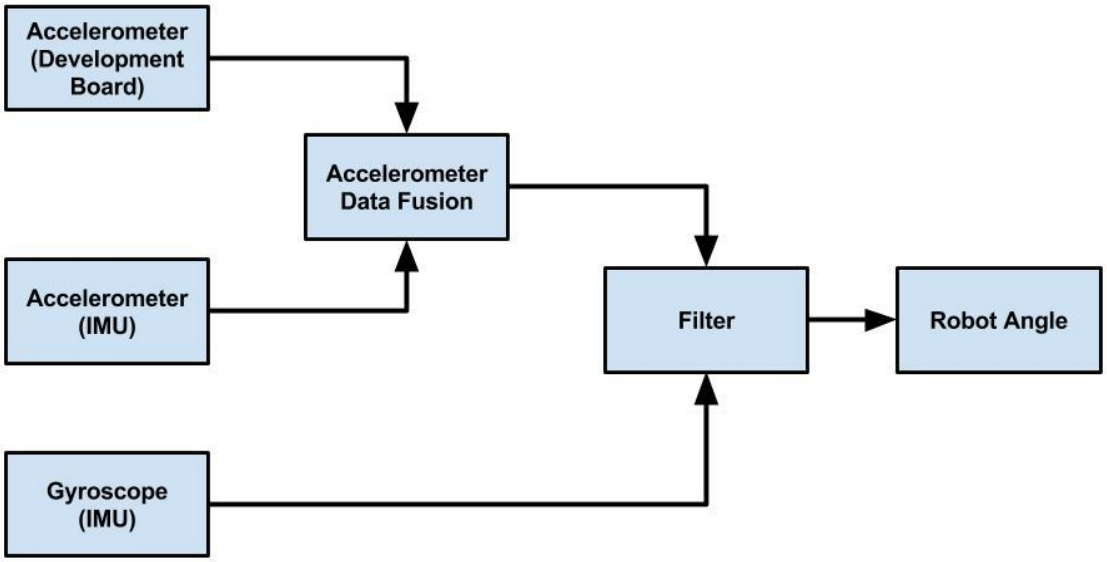

Figure 18: Angle measurement algorithm

5. Tasks Specification

5.1. Robot System:

- **Acquisition_Task:** Is responsible to read the data from the accelerometers and gysroscope and send the data to the Sensor_Fusion task
- **Acquisition_EncoderTaks:** Is responsable to read the data from the enconder and sends it to the Distance_Measurement task
- **Sensor_FusionTaks:** Recieves data from the acquisition_IMU task and calculate the robot angle by making the fusion of the data from the accelerometers and then use a filter to calculate the angle.
- **Distance_MeasurementTask**: Receive the data from the Acquisition_Encoder task and calculate the distance accomplished by the robot.
- **Balance_ControllerTask**: Receive data from Sensor_Fusion task, use the distance calculated by the Distance_Measurement task and using the PID matematical equasion calculate the input for the motors, the value calculated is sended to the Motor_Controller task.
- **Steerling_ControllerTaks:** Uses the data from the Acquisition_Encoder task and calculate the input for the motor using the PID matematical equation, the data calculated is sended to the Robot_Controller task.
- **Motor_ControllerTaks:** Recieves data from both controlers tasks and calculate the PWM for the motor.
- **Send_BluetoothTask :** Sends via bluetooth the data calculated by the Distance_Measurement task and Sensor_fusion.
- **Receive_BluetoothTask:** Reads the data received from the remote system and sends it to the CommandTask.
- **CommandTask:** Recieves data from the Receive_BluetoothTask and according to the command received performs the command.

5.2. Remote Desktop Application:

- **Send_BluetoothTask**: Recieves the data from the Command_HandlerTask and send it to the Robot system via Bluetooth
- **Receive_BluetoothTask**: Recieves data from the Robot System and sends it to the GUI_UpdateTask.
- **Comand_HandlerTask**: Reads the command sent by the user and if valid send is to the Send_BluetoothTask.
- **GUI_UpdateTask**: Recieves data from the Receive_BluetoothTask and Update the GUI with the new data received.

6. Task Communication and Syncronization

In onder to exchange data between tasks is used Semaphore, Queues and Mutex.

6.1. Robot System

Semaphore: in the project is used three semaphores to notify task about an event.

- **Int_IMUSem –** Notifies the task Acquisition_IMU that a new data is available.
- **Int_EncoderSem**: Notifies the task Acquisition_Encoder that a new data is available.
- **Timer_SamPeriodSem:** Notifies the Acquisition_IMU to do a new sample.

Queue: queue is going to be used to exchange data between tasks, the queues used in this project are:

- **AcquiIMU_Queue:** is used to excahnge data between the Acquisition_IMUTask and the Sensor_FusionTask.
- **AcquiEncoder_Queue:** uses to exchange data with the Distance_MeasurementTask.
- **R_Bluetooth_Queue:** used to send data from the Receive_BluetoothTask to the CommandTask.
- **ComHand_Queue:** is used to excahnge data between the Command_HandlerTask and the Steerling_ControllerTask.
- **SensFusion_Queue:** uses to send the value of the angle calculated in the Sensor_FusionTask to the Balance_ControllerTask.
- **DisMeas_Queue:** uses to send the value of the distance (that the Robot alredy had done) calculated in the Distance_MeasurementTask to the Balance_ControllerTask.
- **SteerlingCM_Queue:** is used to send the Input for the Motor calculated in the Steerling_ControllerTask to the Motor_ControllerTask.
- **SteerlingCB_Queue:** is used to send the Input for the Motor calculated in the Steerling_ControllerTask to the Send_BluetoothTaks to be displayed in the GUI.
- **BalancingCM_Queue:** is used to send the Input for the Motor calculated in the Balance_ControllerTask to the Motor_ControllerTask.
- **BalancingCB_Queue:** is used to send the Input for the Motor calculated in the Balance_ControllerTask to the Send_BluetoothTaks to be displayed in the GUI.

Mutex: The mutex are used to protect the access to the critical section of the code, since we are going to use some global variables that are accessed by different tasks is used mutex to control the access of thoses variables. The Mutex used in this project are:

- **Balancing_PID_ParamMutex:** This mutex is used to protect the Struct that has the Parameters for the Balancing PID controler.
- **Distance_PID_ParamMutex: :** This mutex is used to protect the Struct that has the Parameters for the Distance PID controler.
- **Steering_PID_ParamMutex: :** This mutex is used to protect the Struct that has the Parameters for the Steering PID controler.
- **Distance_Mutex:** This mutex is used to protect the variable that stores the value of the Distance calculated by the Distance_MeasurementTask.

ISRs:

- **ISR_Timer_SamPeriod:** Since the time is really important for this system to be able to work is defined a Sampling Period that signal the program when to start a cicle, The control system is is better for a shorter cicle. For this project the cicle will be under 20ms, that means that the the entire cicle is done at least 50 times every second, to manage the Sampling Period is used a timer that will signal the program to start a new Cicle.
- **Interrupt_Encoder:** This routine is to signal the Acquisition_EncodeTask when the encoder interrupt happens.
- **Interrupt_Encoder:** This routine is to signal the Acquisition_IMUTask when the data (accelerometer and Gyroscope value) is available
-

6.2. Remote Desktop System

Queue: in the Remote desktop system is used the following queue:

- **Command_HandlerQueue:** used to send data from the Command_HandlerTask to the Send_bluetoothTaks.
- **Receive_BluetoothQueue:** used to send data from Receive_HandlerTask to the GUI_UpdateTask.

6.3. Robot System Task Communication and Syncronization

The figure 14 and 15 shows how the different task of the system interface/communicates with each other and the communication mechanisms used. In the figure is presented also the Interrupt Service Routines (ISRs) used to communicate with the task of the Robot System.

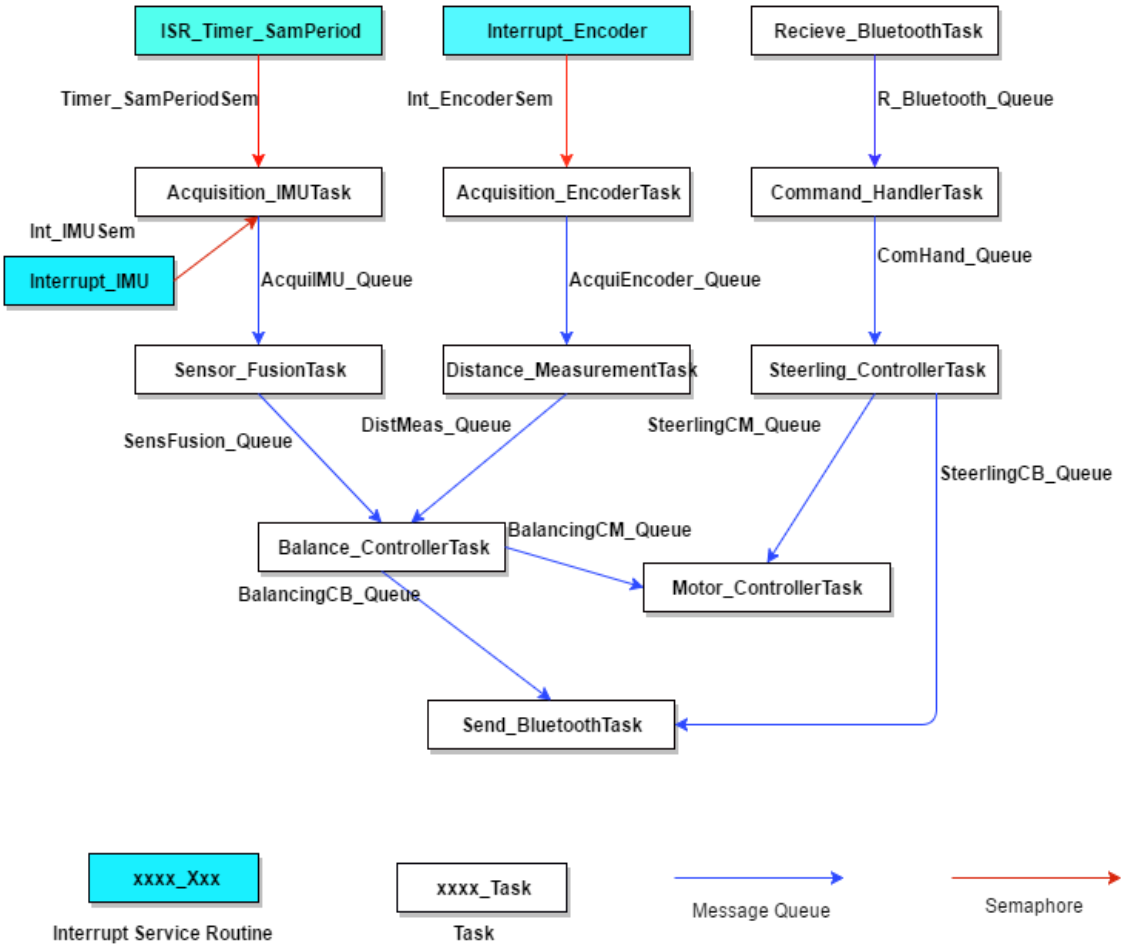

Figure 19: Robot System Tasks Communication and Sincronization Overview

6.4. Remote Desktop System Task Communication and Syncronization

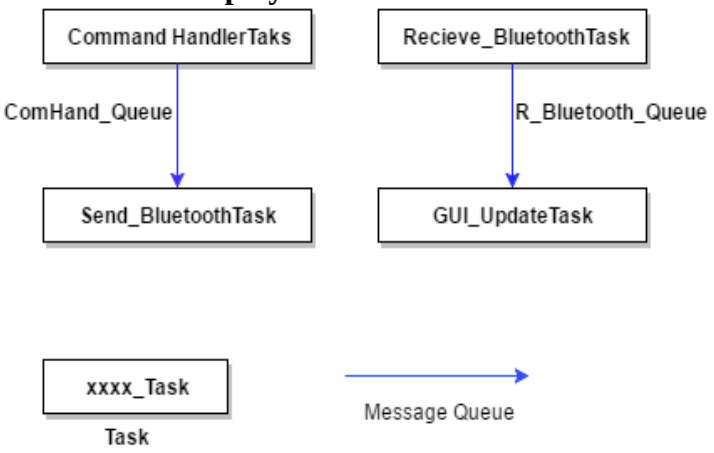

Figure 20: Remote Desktop System Tasks Communication and Sincronization Overview

7. Task Priorities

7.1. Robot System

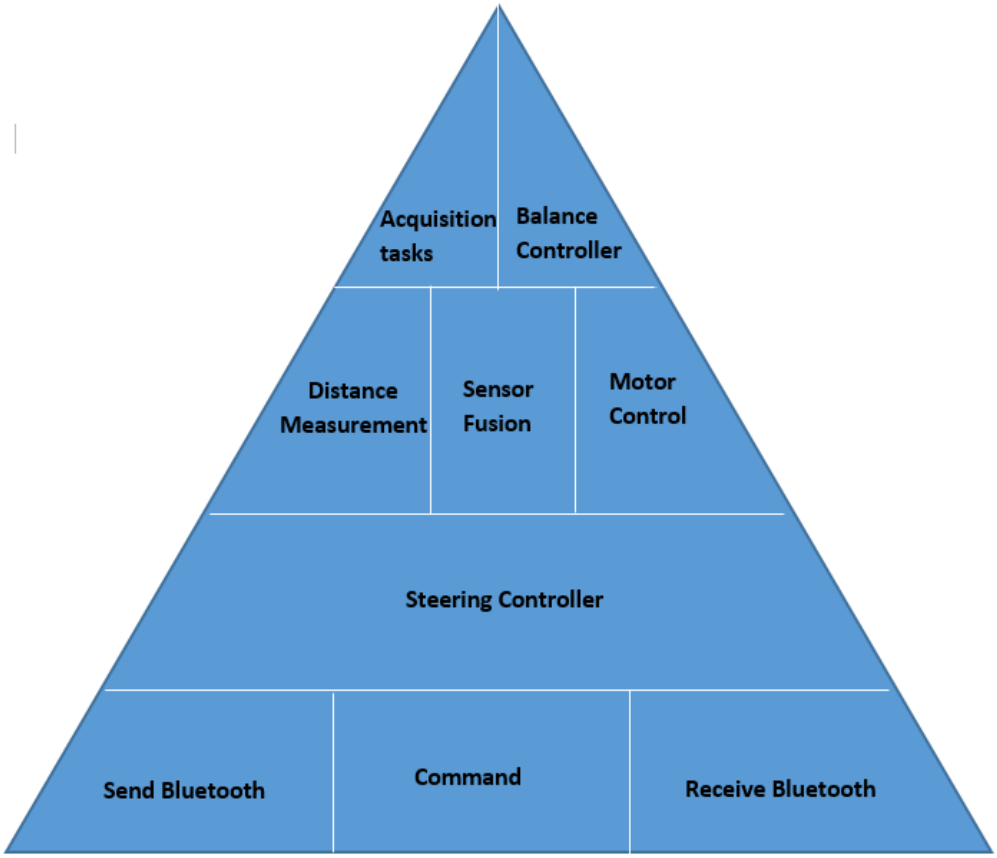

Figure 21: Robot system Task Diagram

Since the most important feature of the system is being able to balance itself the tasks used to perform the balance of the Robot system have higher priority, that's why the Data acquisition tasks and the Balance controller tasks have the highest priority.

The second layer has the second higher priority because they are the tasks that provide data to the Balancing Controller Task, the Sterling as a middle priority since it's not part of the Balancing Controller but in the movement.

The bottom layer has the lowest priority tasks, the tasks for communication since the communication is less important than the balancing.

7.2. Desktop Application

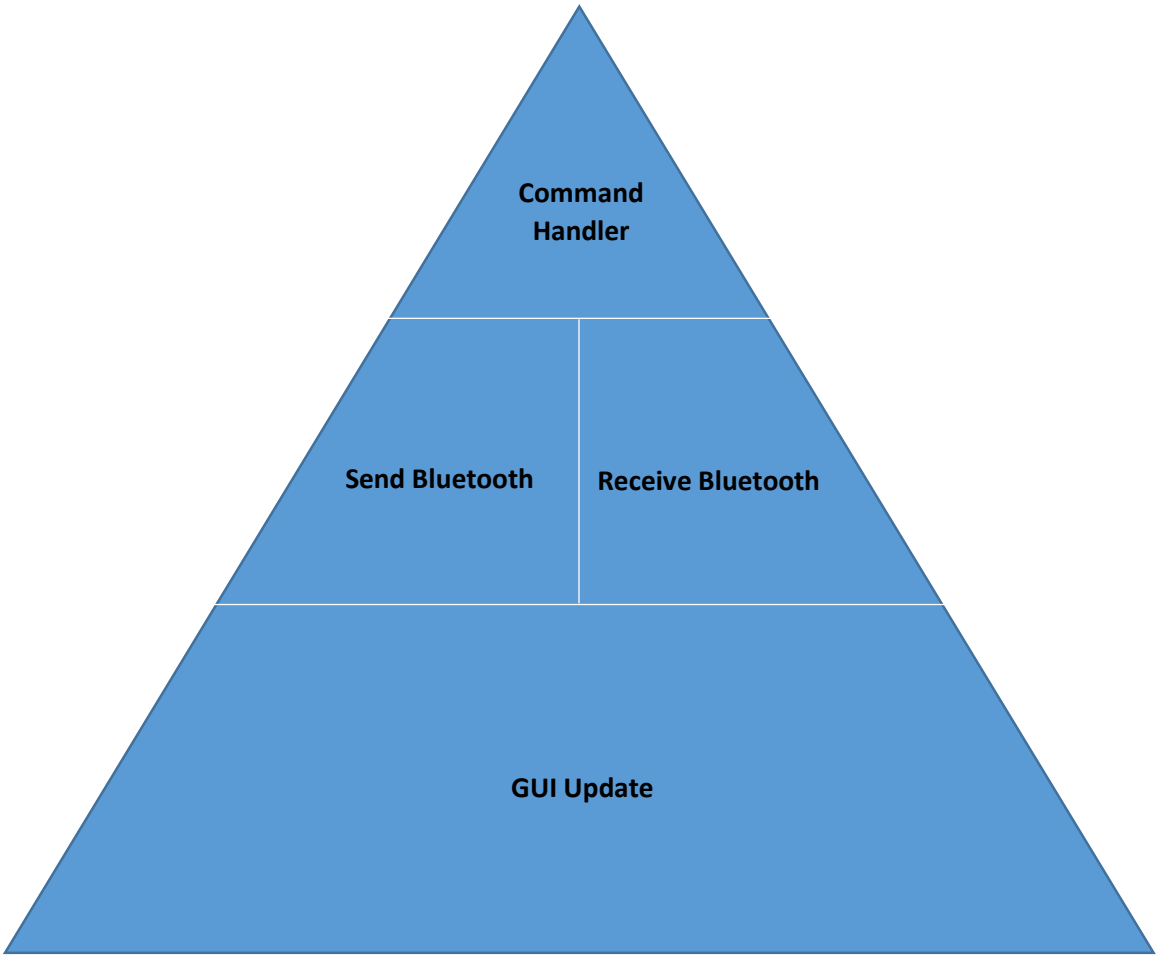

Figure 22: Desktop Application Task Diagram

Since we want the application to be Real Time the interface with the User has the highest priority, the communication Tasks have an intermedian Priority and the GUI Update has the lowest priority since they just give the user the Robot Status.

8. Robot System Flowchart

8.1. Initializer Flowchart

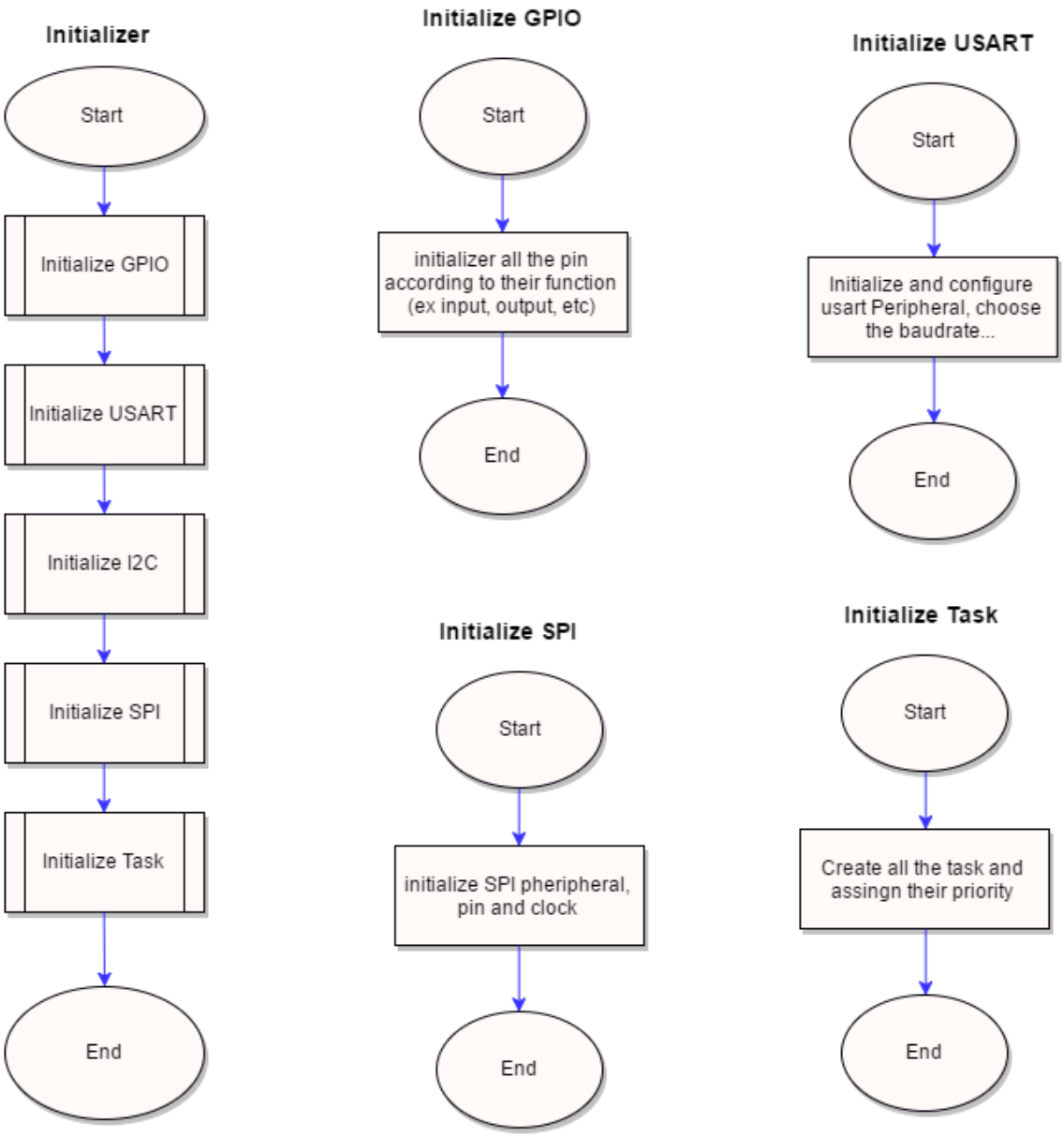

Figure 23: Initializer Flowchart

8.2. Robot Balancing Controller Flowchart

This flowchart present the steps to balance the Robot System.

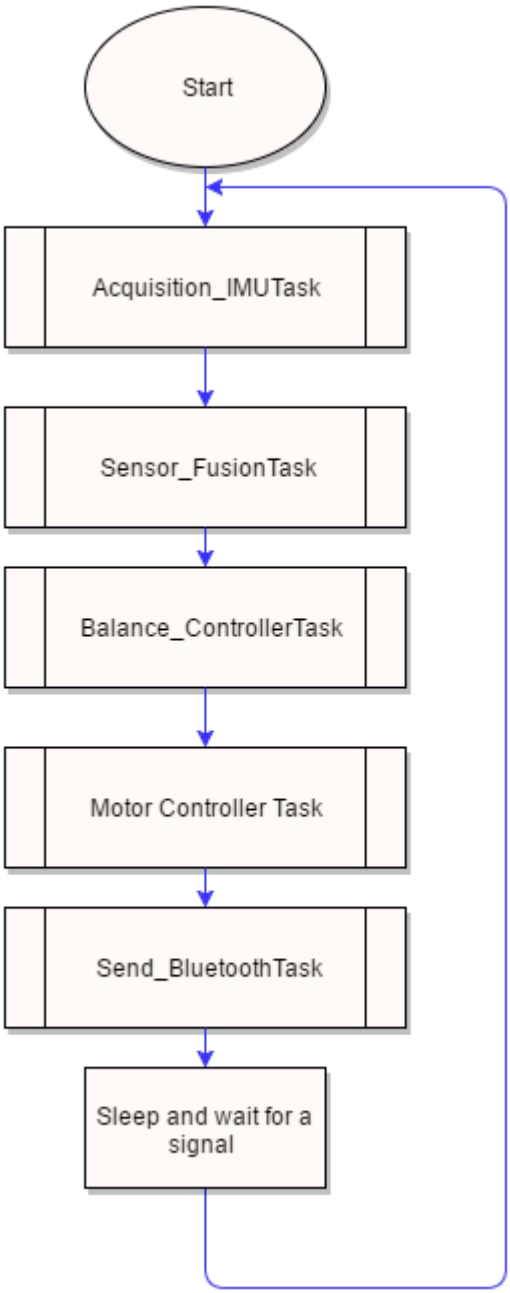

Figure 24: Balancing Controller Flowchart

8.2.1. Acquisition_Task Flowchart

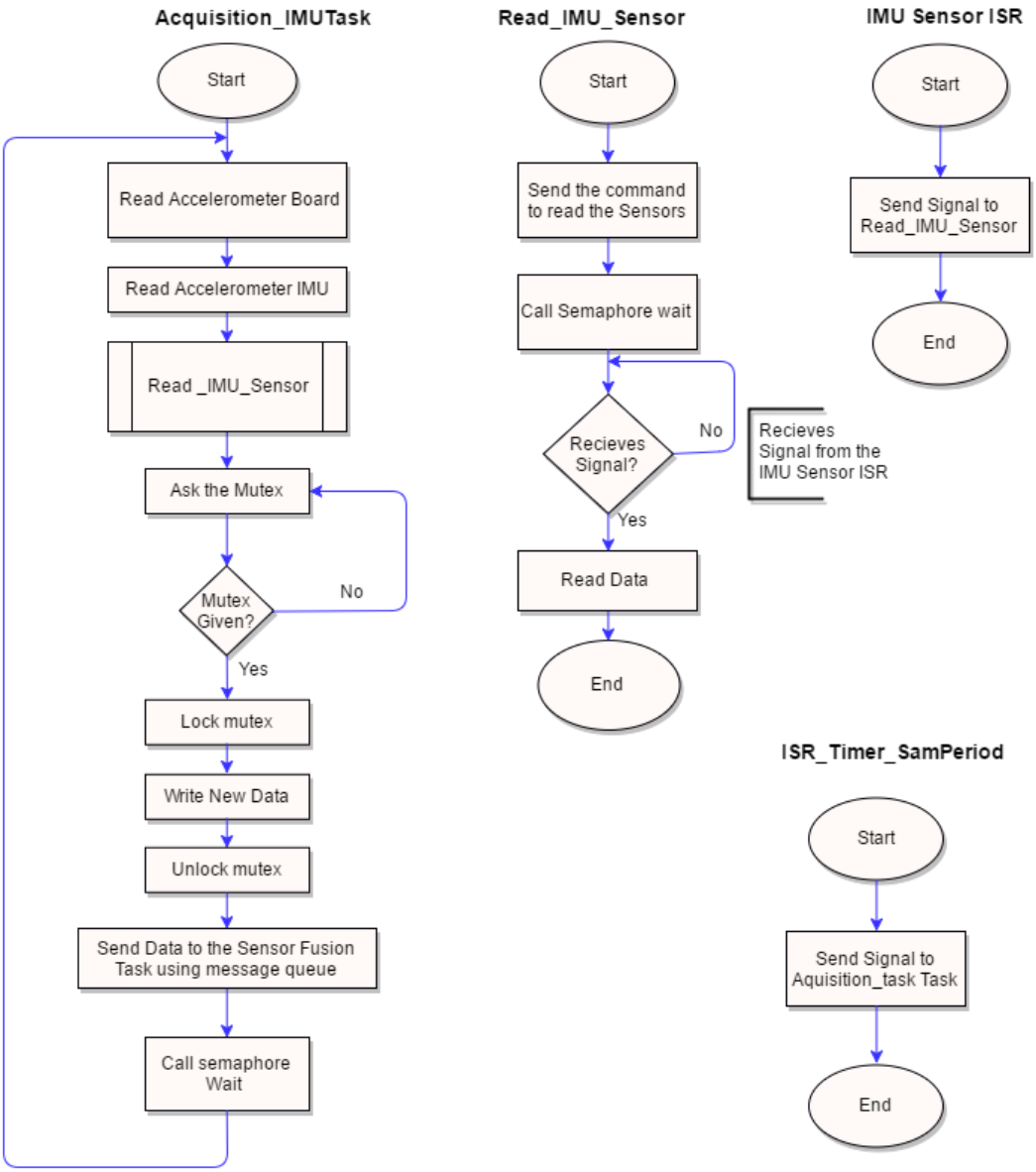

Figure 25: Acquisition Task Flowchart

8.2.2. Acquisition_EncoderTask Flowchart

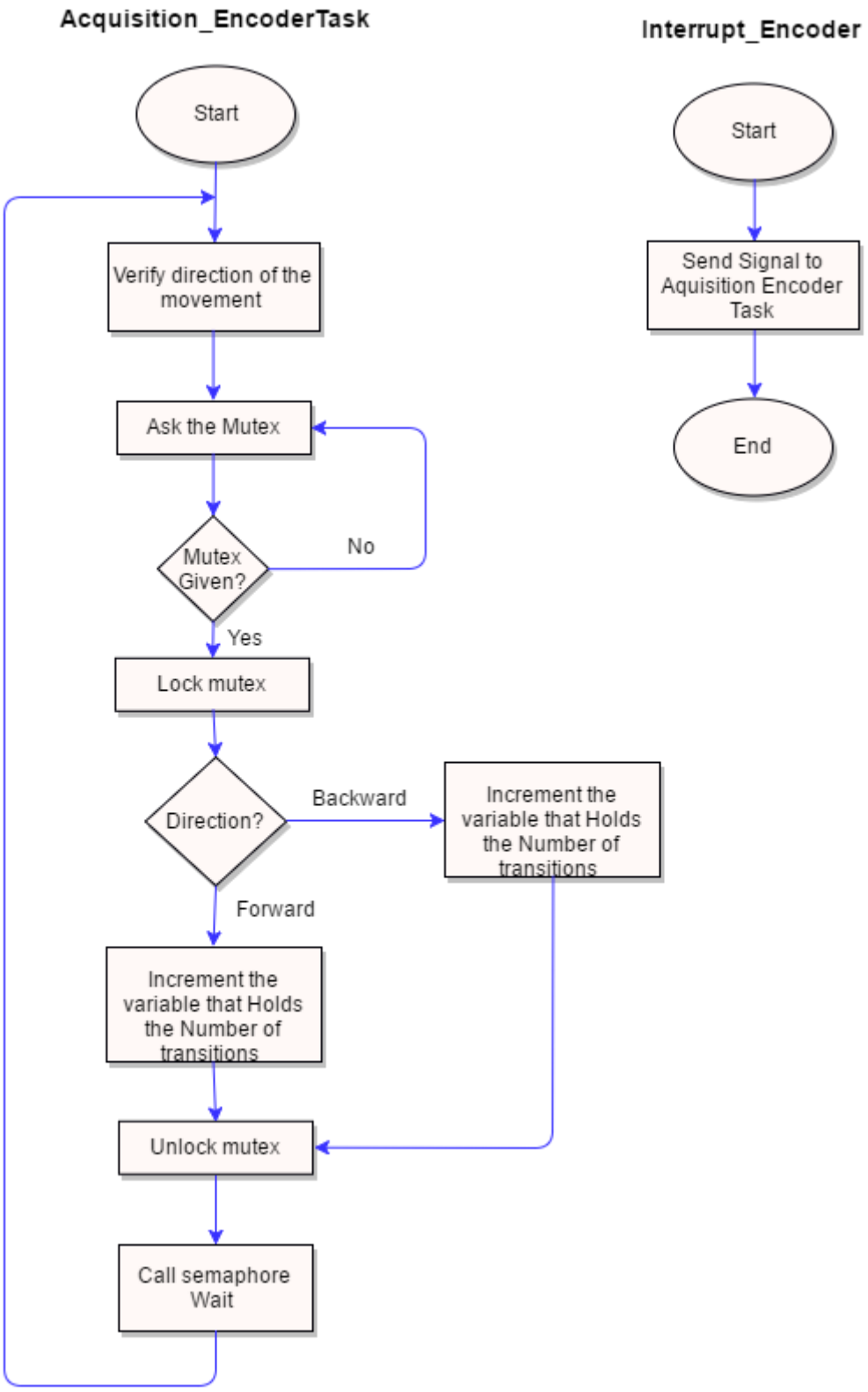

Figure 26: Acquisition Encoder Flowchart

8.2.3. Sensor_Fusion Task Flowchart

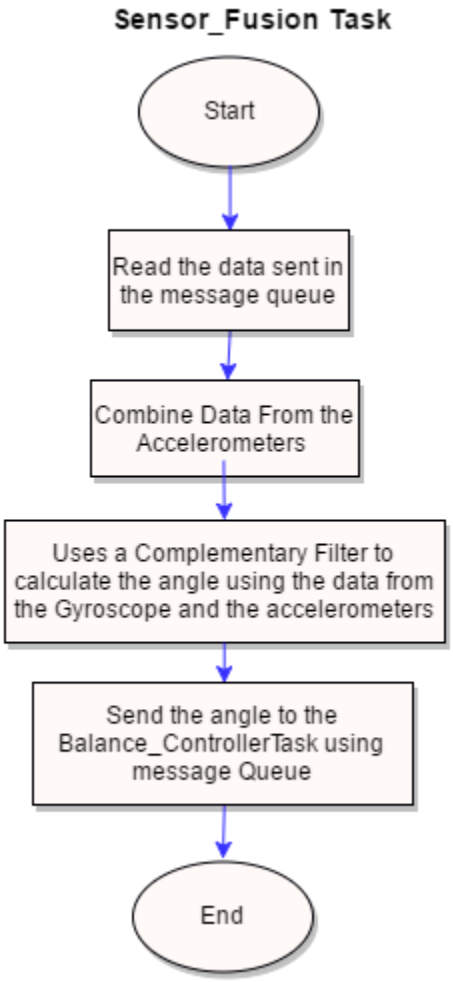

Figure 27: Sensor Fusion Task Flowchart

8.2.4. Balance_PIDTask Flowchart

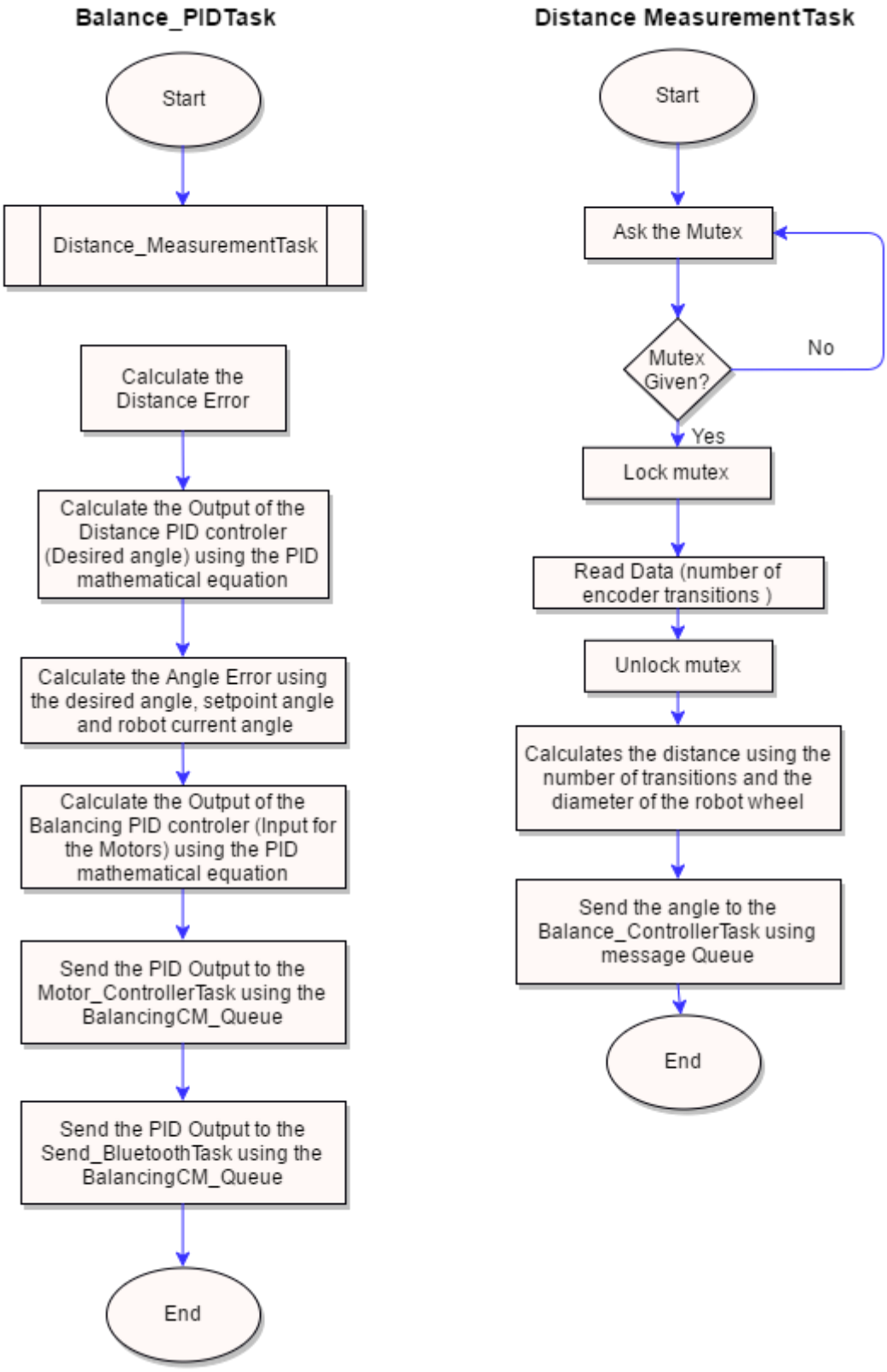

Figure 28: Balance PID Task Flowchart

8.2.5. Send_BluetoothTask and Receive_Bluetooth Flowchart

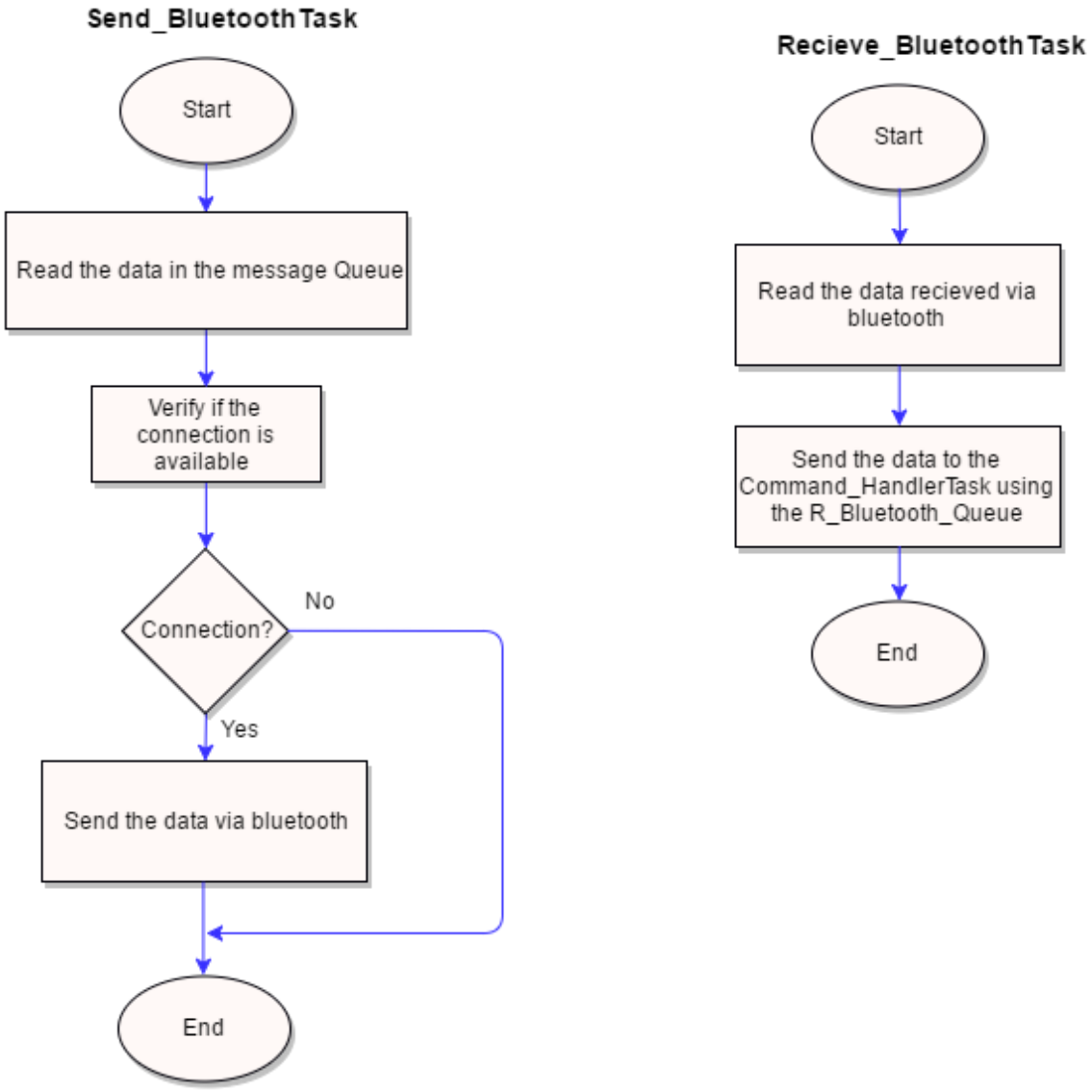

Figure 29: Send and Receive Bluetooth Tasks Flowchart

8.3. Robot Steering Control Flowchart

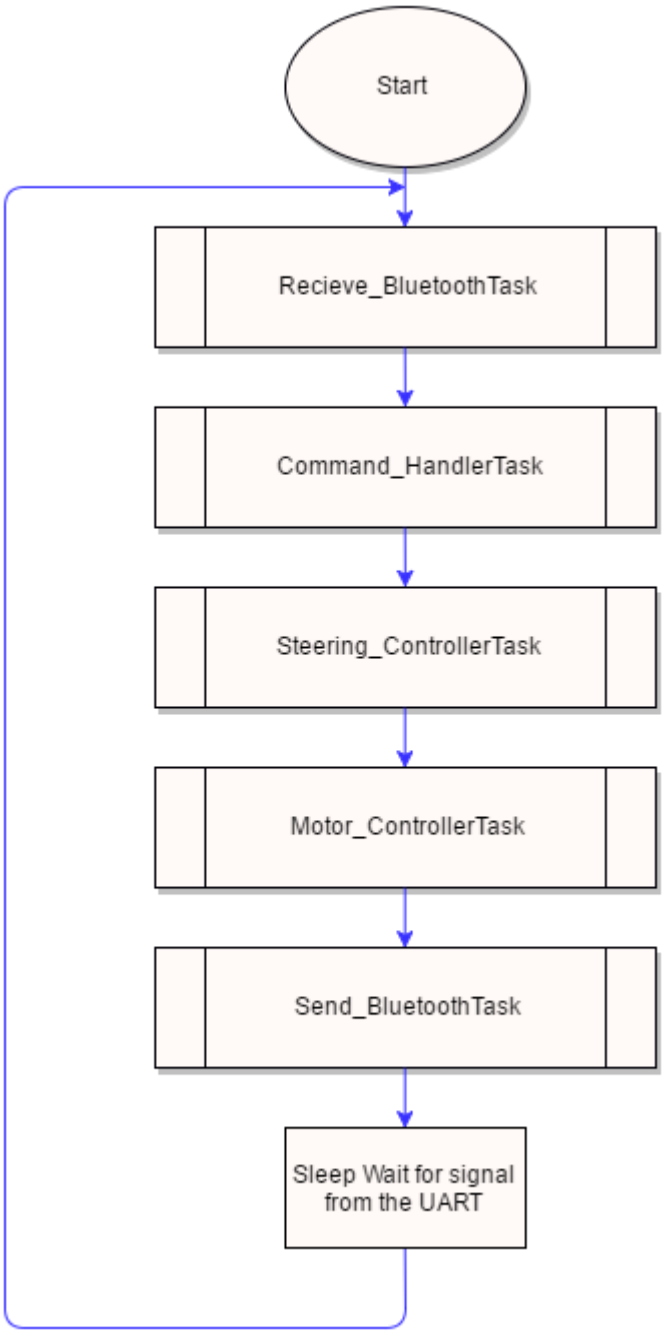

Figure 30: Steering Control Flowchart

8.3.1. Command_Handler_Task Flowchart

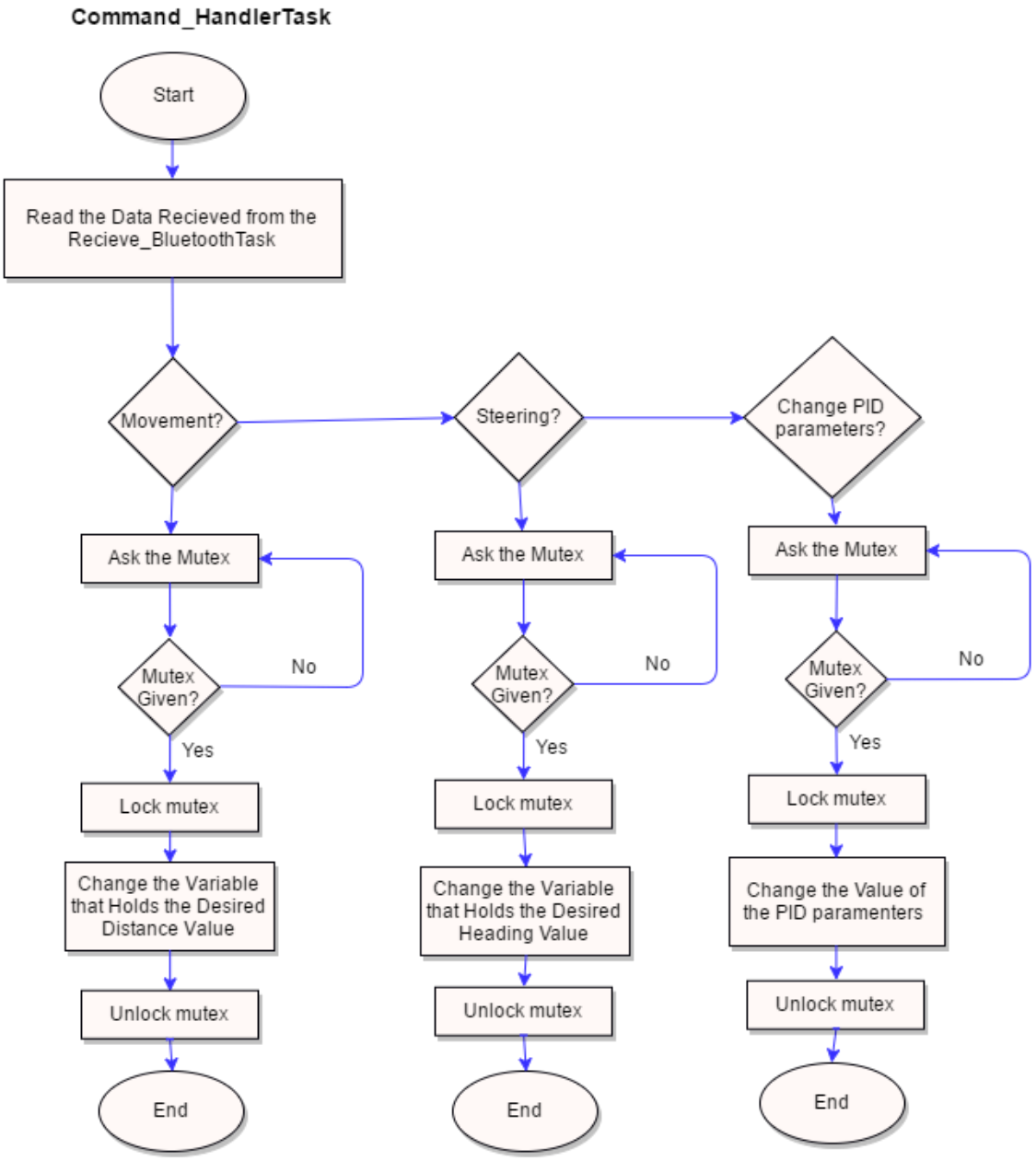

Figure 31: Command Handler Taks Flowchart

8.3.2. Steering_PID Task Flowchart

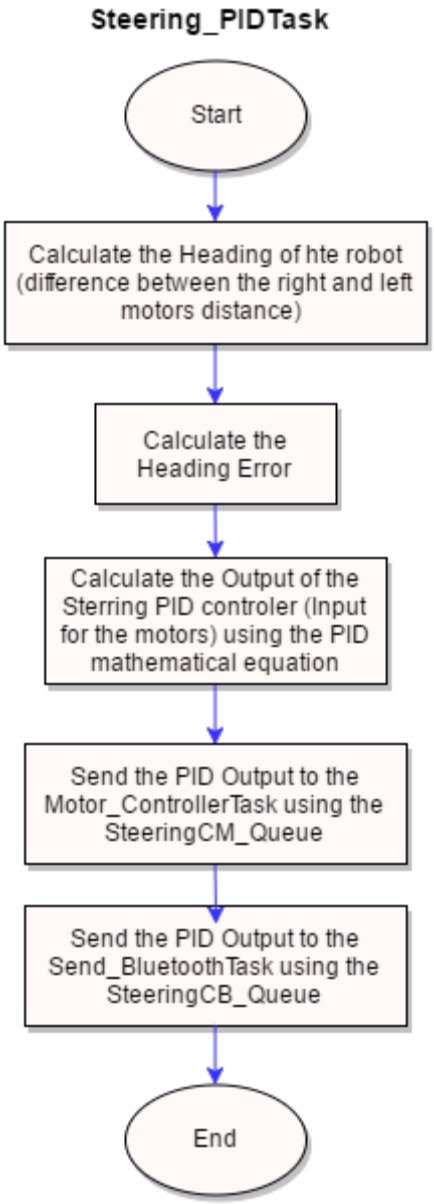

Figure 32: Steering PID Taks Flowchart

9. Desktop Application Flowchart

9.1. Communication (Send and Receive Bluetooth Tasks)

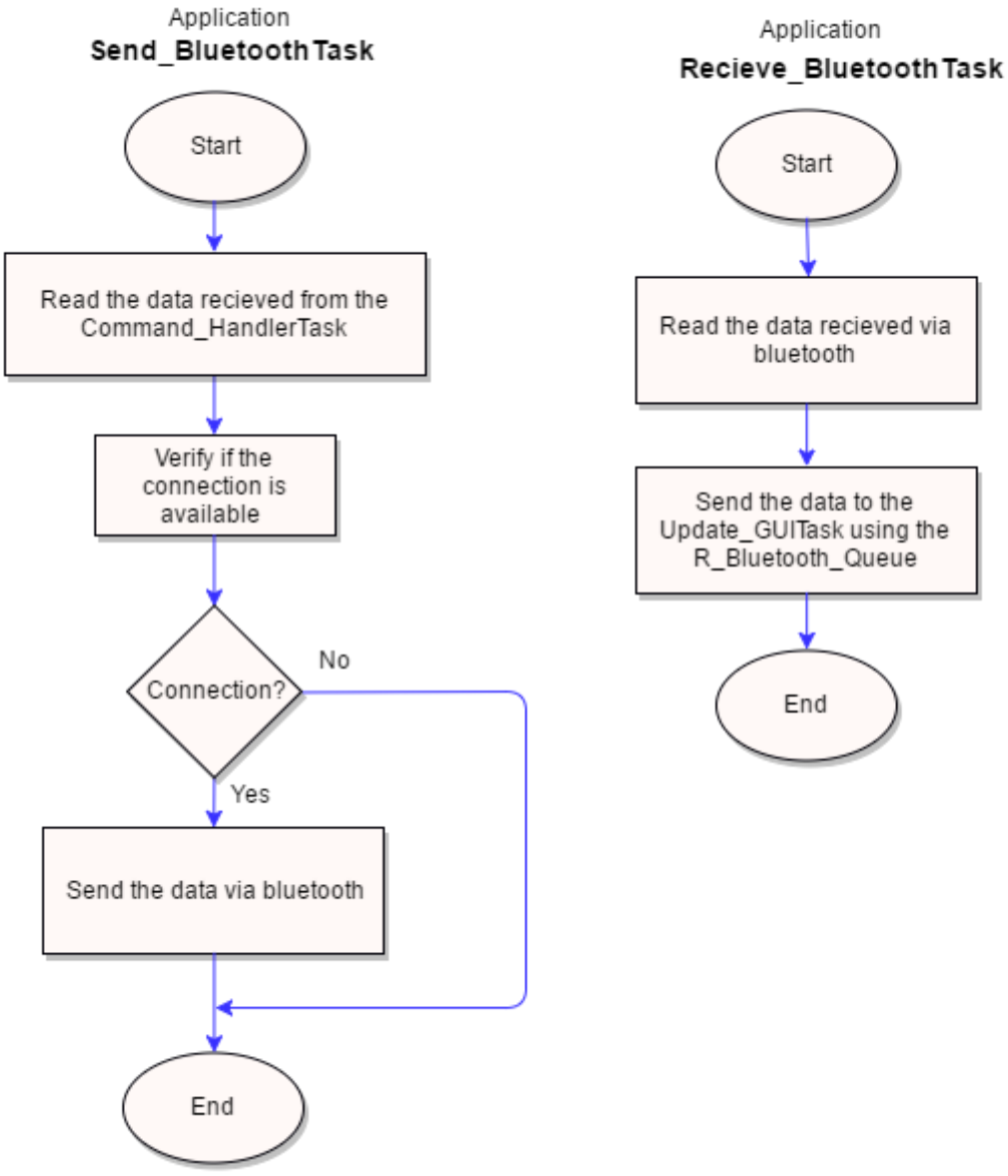

Figure 33: Bluetooth Communication Taks Flowchart (Desktop Applicatio)

9.2. Command_HandlerTask Flowchart

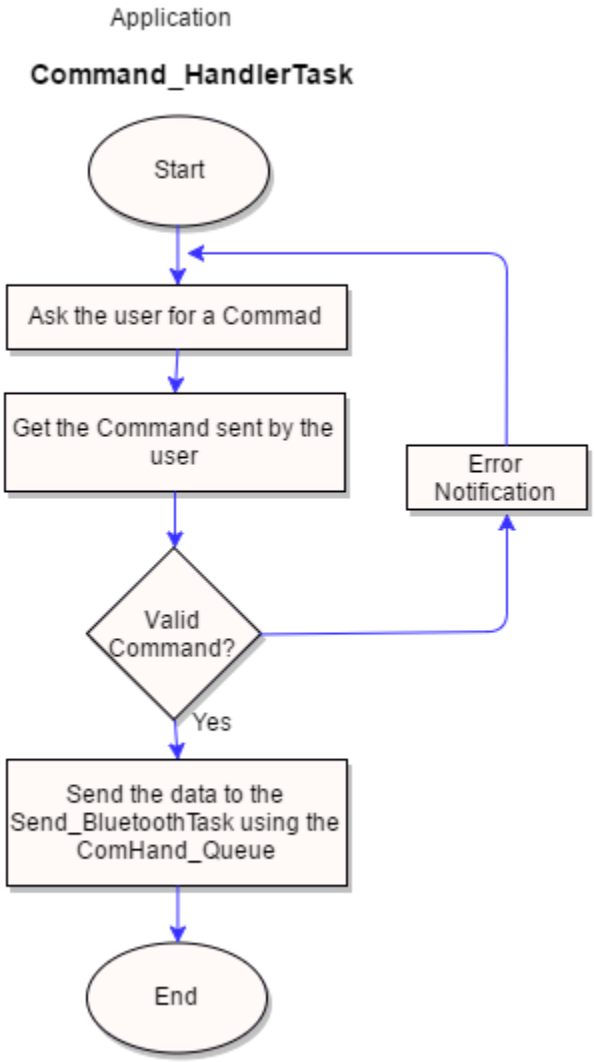

Figure 34: Command Handler Taks Flowchart (Desktop Application)

Header File

Internal Accelerometer

```
//Acelerometer Register Definition
//Control Registers Address (ALL R/W)
#define ACC_CTRL_REG1           0x21 //SM1 CONTROL REGISTER
#define ACC_CTRL_REG2           0x22 // SM2 CONTROL REGISTER
#define ACC_CTRL_REG3           0x23
#define ACC_CTRL_REG4           0x20
#define ACC_CTRL_REG5           0x24
#define ACC_CTRL_REG6           0x25
//Output Registers Address (READ ONLY)
#define ACC_OUT_X_L        0x28
#define ACC_OUT_X_H        0x29
#define ACC_OUT_Y_L        0x2A
#define ACC_OUT_Y_H        0x2B
#define ACC_OUT_Z_L        0x2C
#define ACC_OUT_Z_H        0x2D
```

IMU Sensor

```
#define IMU_ACC_DATA_FORMAT 0x31
// Accelerometer Output Registers addresses
#define IMU_ACC_DATA_XL    0x32
#define IMU_ACC_DATA_XH    0x33
#define IMU_ACC_DATA_YL    0x34
#define IMU_ACC_DATA_YH    0x35
#define IMU_ACC_DATA_ZL    0x36
#define IMU_ACC_DATA_ZH    0x37
// Accelerometer FIFO Registers addresses
#define IMU_ACC_FIFO_CTL   0x38
#define IMU_ACC_FIFO_STATUS 0x39
// Accelerometer Interrupt Registers addresses
#define IMU_ACC_INT_ENABLE      0x2E
#define IMU_ACC_FIFO_STATUS 0x39
```

User Variables

```
#define CS
#define I2C_SDA
#define I2C_SCL
#define USART_RX
#define uSART_TX
#define ENCODER1_A
#define ENCODER2_A
#define ENCODER1_B
#define ENCODER2_B
#define MOTOR1_EN
#define MOTOR2_EN
#define INT_IMU
int Encoder1_Transition;
int Encoder2_Transition;
float Distance;
float Heading;
char Bluetooth_Buffer_IN[16];
char Bluetooth_Buffer_OUT[16];
struct Pid{
     float mKP, mKD, mKI;
     float mError;
     float mVarControl;
     float msetpoint;
}
//Semaphores
sem_t s_Timer_SamPeriod;
sem_t s_Int_Encoder;
sem_t s_Int_IMU;
sem_t s_Int_Serial;
```

User Functions

```
void Timer_SamPeriod_Config(void);
float Distance_Measurememt(void);
float Get_Angle(void);
float Get_Heading(void);
void Get_Internal_Accelerometer(void);
float Get_IMU_Accelerometer(void);
Void Write_SPI(uint8_t address, unsigned char data);
Void Read_SPI(uint8_t address, unsigned char *data);
Void Write_Uart(unsigned char *data);
Char* Read_Uart(unsigned char *data);
```

GUI Sketch

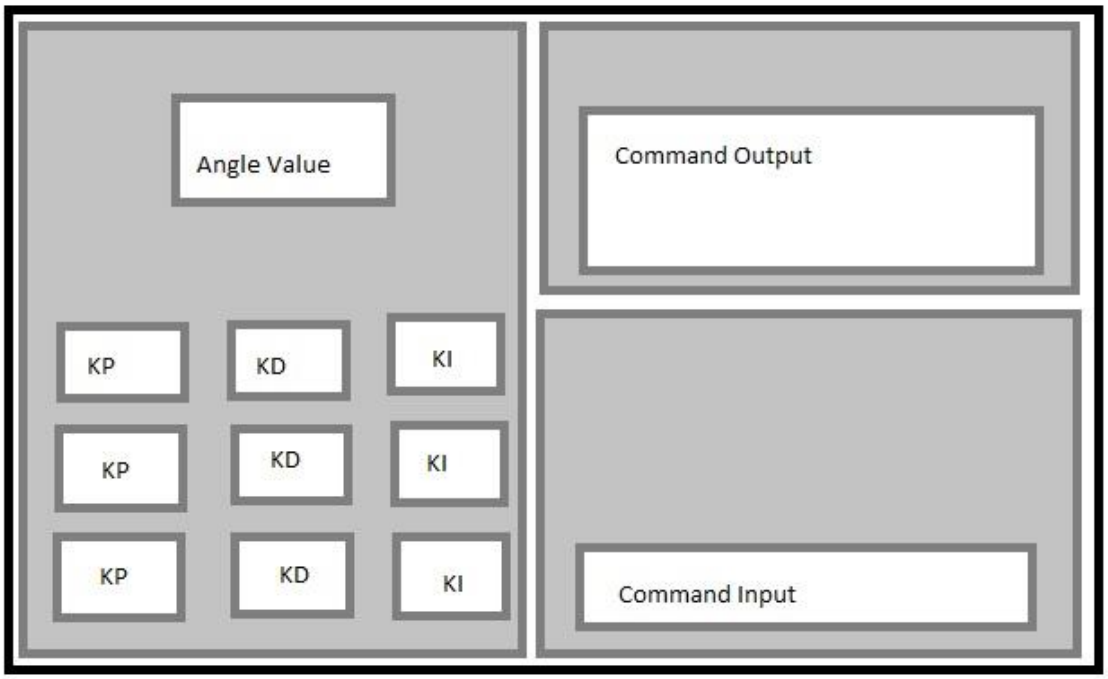

Figure 35: GUI Sketch

Gantt Diagram