



**DE MONTFORT  
UNIVERSITY  
LEICESTER**

**FACULTY OF TECHNOLOGY**

**M.Sc MECHATRONICS**

## **INDIVIDUAL PROJECT**

# **IN-PIPE INSPECTION ROBOT CONTROL AND SYSTEM MODELING**

**SUPERVISOR: DR SENG CHONG**

**STUDENT: GIDEON ANTHONY D'SOUZA**

**P-NUMBER: P13175646**

## **Abstract**

This project deals with the design and simulation of a simple but efficient smart in-pipe inspection robotic system using the RF communication. It provides theoretical studies of robotics' and modelling techniques using equivalent electric circuits with wireless communication. Robots are devices that cleanly perform the function as per the user requirements. In this concept the in-pipe inspection robot is designed in to travel within pipes of prescribed diameters and collect data pertaining to the environment inside the pipe and transmit it with the interfacing circuit of RF communication. The RF transmitter in the robot will give the information to the RF receiver placed in the control hub. The control hub is a remote control station used to control the motion of the robot. It offers a practical solution to the industrial applications for pipe inspections.

## Table of Contents

|  |    |
|--|----|
| 1. INTRODUCTION:   | 8  |
| 1.1. MOTIVATION:   | 8  |
| 1.2. SCOPE:  | 9  |
| 1.3. OBJECTIVES:   | 9  |
| 1.4. THESIS ORGANISATION:  | 10 |
| 2. Literature review:  | 11 |
| 3. METHODOLOGY:  | 14 |
| 4. SYSTEM REQUIRMENTS:   | 16 |
| 4.1. Hardware:   | 16 |
| 4.1.1. Arduino microcontroller                                     | 16 |
| 4.1.2. L298N DUAL H BRIDGE   | 17 |
| 4.1.3. INDICATOR SENSOR MODULE L3G4200D, ADXL345, HMC5883L, BMP085 | 17 |
| 4.1.4. BIPOLAR HYBRID STEPPER MOTOR                                | 18 |
| 4.1.5. DC MOTOR  | 19 |
| 4.1.6. MICRO SD BOARD READER                                       | 19 |
| 4.1.7. RF TX/RX  | 20 |
| 4.2. SOFTWARE REQUIREMENTS   | 20 |
| 5. USE CASE DIAGRAM:   | 22 |
| 6. REQUIRMENT DIAGRAM:   | 24 |
| 7. BLOCK DIAGRAM:  | 26 |
| 8. BAROMETRIC SENSOR   | 27 |
| 8.1. INTRODUCTION  | 27 |
| 8.2. APPLICATIONS  | 27 |
| 8.3. SPECIFICATIONS AND RANGE                                      | 27 |
| 8.4. PIN CONFIGURATION:  | 28 |
| 8.5. WIRING TO ARDUINO :   | 29 |
| 8.6. SAMPLE CODE   | 30 |
| 8.7. OUTPUT:   | 32 |
| 8.8. CONCLUSION  | 32 |
| 9. IMU (Inertial Measurment Unit)                                  | 33 |
| 9.1. INTRODUCTION:   | 33 |
| 9.2. ACCLEROMETER ADXL345  | 33 |

|         |   |    |
|---------|---|----|
| 9.2.1.  | FEATURES .....                          | 33 |
| 9.2.2.  | APPLICATION:.....                       | 34 |
| 9.2.3.  | ABSOLUTE POWER RATING.....              | 34 |
| 9.2.4.  | THERMAL RESISTANCE.....                 | 35 |
| 9.2.5.  | PIN CONFIGURATION:.....                 | 35 |
| 9.2.6.  | ADXL345 TO ARDUNIO .....                | 36 |
| 9.3.    | GYROSCOPE L3G4200D .....                | 36 |
| 9.3.1.  | INTRODUCTION:.....                      | 36 |
| 9.3.2.  | FEATURES:.....                          | 36 |
| 9.3.3.  | PIN DISCRIPTION:.....                   | 37 |
| 9.3.4.  | ELECTRICAL CHARACTERISTICS.....         | 37 |
| 9.3.5.  | TEMPERATURE CHARECTERSTICS.....         | 38 |
| 9.4.    | Compass HMC5883L .....                  | 38 |
| 9.4.1.  | INTRODUCTION.....                       | 38 |
| 9.4.2.  | Features.....                           | 38 |
| 9.4.3.  | PIN CONFIGURATION.....                  | 39 |
| 9.4.4.  | SPECIFICATIONS.....                     | 40 |
| 9.4.5.  | SAMPLE CODE.....                        | 40 |
| 9.4.6.  | OUTPUT .....                            | 51 |
| 9.5.    | CONCLUSION:.....                        | 52 |
| 10.     | Motion control unit.....                | 53 |
| 10.1.   | L298n .....                             | 53 |
| 10.1.1. | INTRODUCTION:.....                      | 53 |
| 10.1.2. | FEATURES:.....                          | 53 |
| 10.1.3. | WORKING .....                           | 53 |
| 10.2.   | SIMULINK MODELING .....                 | 55 |
| 10.2.1. | SIMULINK MODELLING OF DC MOTOR. ....    | 55 |
| 10.2.2. | OUTPUT .....                            | 56 |
| 10.2.3. | SIMULINK MODELING OF STEPPER MOTOR..... | 57 |
| 11.     | System internal block diagram:.....     | 58 |
| 11.1.   | MOTOR CONTROL:.....                     | 58 |
| 11.2.   | SENSOR BLOCK SYSTEM .....               | 60 |
| 11.3.   | CONCLUSION:.....                        | 61 |
| 12.     | Experimental results: .....             | 62 |

|         |   |     |
|---------|---|-----|
| 12.1.   | INTRODUCTION.....                             | 62  |
| 12.2.   | ARDUINO UNO SPECIFICATIONS [APENDIX 1]: ..... | 63  |
| 12.3.   | CONTROL HUB .....                             | 64  |
| 12.3.1. | CONTROL HUB PIN CONNECTIONS:.....             | 64  |
| 12.3.2. | SEQUENCE OF THE CODE.....                     | 65  |
| 12.3.3. | PROGRAM:.....                                 | 77  |
| 12.3.4. | OUTPUT .....                                  | 95  |
| 12.4.   | ON BOARD MODULE .....                         | 96  |
| 12.4.1. | MOTOR CONTROL PIN CONFIGURATION.....          | 97  |
| 12.4.2. | SEQUENCE OF THE CODE.....                     | 98  |
| 12.4.3. | SEQUENCE FLOW DIAGRAM.....                    | 99  |
| 12.4.4. | PROGRAM.....                                  | 100 |
| 12.4.5. | SENSOR MODULE .....                           | 107 |
| 12.5.   | DATA ANALYSIS AND PLOTING .....               | 120 |
| 12.4.4. | OUTPUT.....                                   | 120 |
| 13.     | Future work: .....                            | 122 |
| 14.     | CONCLUSION:.....                              | 130 |
|         | BIBLOGRAPHY .....                             | 131 |
|         | APPENDIX .....                                | 132 |

## LIST OF TABLES

|   |    |
|---|----|
| Table 1 PIN CONFIGURATON BMP085.....              | 29 |
| Table 2 ABSOLUTE POWER RATING ADXL345 .....       | 34 |
| Table 3 THERMAL RESISTANCE ADXL345.....           | 35 |
| Table 4 PIN CONFIGURATION ADXL345.....            | 35 |
| Table 5 PIN CONFIGURATION FOR L3G4200D .....      | 37 |
| Table 6 ELECTRICAL CHARACTERISTICS L3G4200D ..... | 38 |
| Table 7 TEMPERATURE CHARECTERSTICS L3G4200D.....  | 38 |
| Table 8 HMC5883L PIN CONFIGURATION .....          | 40 |
| Table 9 SPECIFICATIONS HMC5883L.....              | 40 |
| Table 10 H BRIDGE SWITCH CHARESTERISTICS.....     | 54 |
| Table 11 ARDUINO UNI SPECIFICATIONS.....          | 63 |

## TABLE OF FIGURE

|   |     |
|---|-----|
| Figure 1 WALL PRESSED IN PIPE INSPECTION ROBOT.....   | 9   |
| Figure 2 ROBOTS MECHANISM .....   | 11  |
| Figure 3 BLOCK DIAGRAM OF THE OVERALL SYSTEM.....   | 14  |
| Figure 4 ARDUINO UNO .....  | 16  |
| Figure 5 L298N DUAL H BRIDGE .....  | 17  |
| Figure 6 10DOF 9-AXIS ATTITUDE INDICATOR L3G4200D ADXL345 HMC5883L<br>BMP085 MODULE ARDUINO ..... | 17  |
| Figure 7 BIPOLAR HYBRID STEPPER MOTOR .....   | 18  |
| Figure 8 12V DC MOTOR.....  | 19  |
| Figure 9 MICRO SD BOARD READER.....   | 19  |
| Figure 10 RF TX/RX .....  | 20  |
| Figure 11 USE CASE DIAGRAM .....  | 22  |
| Figure 12 REQUIRMENT DIAGRAM.....   | 24  |
| Figure 13 SYSML BLOCK DIAGRAM .....   | 26  |
| Figure 14 PIN CONFIGURATION OF BMP085 WITH PACKAGE DIMENSIONS IN mm ...                           | 28  |
| Figure 16 WIRING FOR BAROMETRIC SENSOR .....  | 29  |
| Figure 17 OUTPUT FOR BMP085 SENSOR .....  | 32  |
| Figure 18 ADXL345 PIN CONFIGURATION .....   | 35  |
| Figure 19 ADXL345 WIRING .....  | 36  |
| Figure 20 PIN CONFIGURATION FOR L3G4200D .....  | 37  |
| Figure 21 HMC5883L PIN CONFIGURATION.....   | 39  |
| Figure 22 OUTPUT FOR THE IMU .....  | 51  |
| Figure 23 SCHEMATIV REPRESENTATION OF H BRIDGE .....  | 54  |
| Figure 24 SIMULINK MODEL FOR DC MOTOR.....  | 55  |
| Figure 25 DC MOTOR I AND RPM PLOT SIMULINK .....  | 56  |
| Figure 26 SIMULINK MODELING OF STEPPER MOTOR.....   | 57  |
| Figure 27 INTERNAL BLOCK DIAGRAM - MOTOR CONTROL .....  | 58  |
| Figure 28 INTERNAL BLOCK DIAGRAM - SENSOR CONTROL .....   | 60  |
| Figure 29 OVERALL SYSTEM BLOCK DIAGRAM .....  | 63  |
| Figure 30 PIN CONNECTIONS FOR CONTROL HUB .....   | 64  |
| Figure 31 REMOTE CONTROLLER MENU .....  | 95  |
| Figure 32 OUTPUT SHOWING IT'S CONNECTED.....  | 96  |
| Figure 33 RECEIVER MODULE ON THE CONTROL HUB .....  | 96  |
| Figure 34 MOTOR PIN CONNECTION.....   | 97  |
| Figure 35 SEQUENCE FLOW OF MOTOR CONTROL .....  | 98  |
| Figure 36 SEQUENCE DIAGRAM FOR MOTOR CONTROL .....  | 99  |
| Figure 37 PIN CONNECTIONS SENSOR MODULE .....   | 107 |
| Figure 38 SEQUENCE FLOW DIAGRAM FOR SENSOR BLOCK.....   | 108 |
| Figure 39 SEQUENCE FLOW OF SENSOR BLOCK .....   | 109 |
| Figure 40 SENSOR MODULE ON ROBOT .....  | 119 |
| Figure 41 LOGGER FILE WHICH STORES SENSOR DATA.....   | 121 |
| Figure 42 PLOT OF TEMPERATURE AND PRESSURE DATA.....  | 121 |

|                                  |     |
|----------------------------------|-----|
| Figure 43 GUI LOG IN PAGE .....  | 122 |
| Figure 44 GUI CONTROL PAGE ..... | 126 |



# 1. INTRODUCTION:

## 1.1. MOTIVATION:

Pipe lines are employed in the transportation of oil, gases, chemicals and other fluids for a long time in several countries. Due to the constant usage of these pipe lines, they tend to become victim to wear and tear. In the case of underground pipes or pipes in plants. There could be undesired pressure or temperature build ups or drops which we would like to monitor and control. Most of the pipeline inspection is done using inline pipeline inspection tool to check for corrosion or change in the environmental conditions inside the pipe such as temperature, pressure etc. We all know and understand that the data obtained from inspection of the pipe from outside is accurate and reliable but if there is a section of pipe inaccessible for inspection, it would not be wise and sensible to alter or modify the pipe system for inspection. In this case we would need to conduct the inspection from inside the pipe. The most sort-after way to manage this problem now is to employ robots. Using an in pipe inspection robot for inaccessible pressurised pipes in plants, we can still check the quality of the pipes and also monitor the conditions within the pipe without interrupting the on-going process and there by controlling it.

Robotics and control is a fast growing field which is helps to remove the factor of human effort from the job or in the case of dangerous jobs it helps remove the actual presence of the human by allowing him control the entire operation remotely, thereby increasing safety. Since the inspection and maintenance of pipe line systems which are used for industrial plants and transportation of oil and gas and other fluids or just checking finished pipes for flaws is expensive, we use pipe inspection robots to make the task easy for us. The use of inspection robots in pipe line system helps improve the efficiency and productivity of industrial plants.

Such robots are called in pipe robots. There are many different types of mechanisms of in pipe robots which are used for specific purposes. Here we use the wheeled wall press type robot in order to obtain data from the pipe. Most wall pressed robots use spring tension to press against the walls of the pipe however it comes with certain restrictions [1]. The force applied to press against the walls can be obtained by the use of a dc or stepper motor in order to keep the robot firmly fixed against the walls of the pipe. The advantages of this type of robot is that

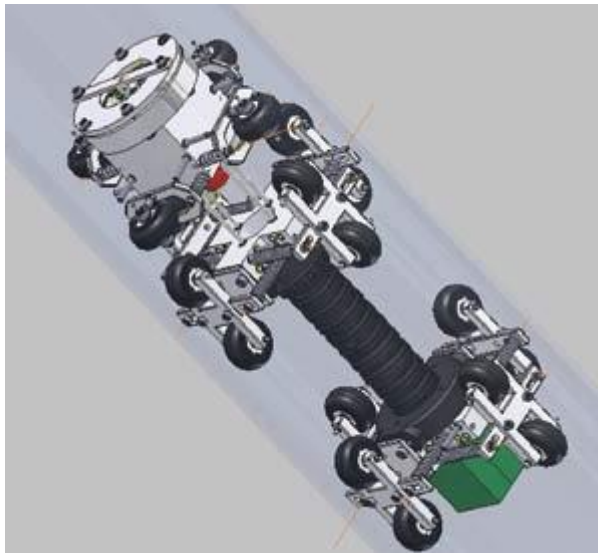
- It is a simple and flexible system.
- It is adjustable to the change of pipe diameter and can move quickly.

- It is energy efficient and can carry required load for future analysis and data collection.

The drawbacks of the system are:

- Due to the force exerted against the inner surface of the pipe it could cause damage to the inner surface of the pipe due to the frictional forces.
- The wheels may get caught or stuck in any irregularities in the surface of the pipe or it could be affected by slippage.

Here figure 1 gives us an example of a wheeled wall pressed inspection robot.



**Figure 1 WALL PRESSED IN PIPE INSPECTION ROBOT**

## **1.2. SCOPE:**

To design and construct a remotely operated in-pipe robot which is able to travel in pipes ranging from 7 inch to 12 inch in diameter and collect and transmit the data to the remote controller.

## **1.3. OBJECTIVES:**

- To design a robot that is able to travel in pipes of a particular range of diameter.
- To control the robot remotely using RF technology.
- To be able to collect temperature and pressure data from pipes and transmit them to the controller and warn the operator when required and to save and plot the obtained data for future analysis.

## 1.4. THESIS ORGANISATION:

CHAPTER 1: In this chapter we shall be giving an introduction about the project.

CHAPTER 2: Here we shall give the literature review of the project.

CHAPTER 3: In this chapter we shall explain the methodology of the project.

CHAPTER 4: here we shall discuss about the hardware and software's requirements needed.

CHAPTER 5: here we explain the use case diagram.

CHAPTER 6: here we will talk about the requirements of the system using Sysml.

CHAPTER 7: in this chapter we shall discuss the block diagram of the system in Sysml.

CHAPTER 8: in this chapter we shall discuss about the barometric sensor which is used to measure the temperature and pressure.

CHAPTER 9: here we shall talk about the IMU which is used to measure the positioning and orientation of the system

CHAPTER 10: in this chapter we shall discuss about the motion control system in the pipe inspection robot.

CHAPTER 11: here we shall discuss about the two integral internal blocks of the system.

CHAPTER 12: At last we shall give discuss about the final project on a whole its results output and working.

CHAPTER 13: Here we discuss future developments.

## 2. Literature review:

In this chapter we shall take a good look in to the research that is needed and which was conducted for the undertaking of this project. We shall give a detailed literature review of all the various aspects involved in the project.

In the first paper [1] the author explains as the different types of pipe inspection robots from which we selected a wall press robot which we thought would apt for the project. The author gives a detailed inspection of the development of in pine inspection robots and states that the wall press robot is adaptable to changes of inner pipe diameter. It has a simple mechanism. If track is used it can avoid getting stuck in holes. It suitable for long range inspection. It is energy efficient and can carry heavy load.

In this paper [2] the author explains the entire mechanism of the inspection robot off which our robot is based. Here the mechanism as to which we can help the robot to gain grip against the walls to hold it in place.

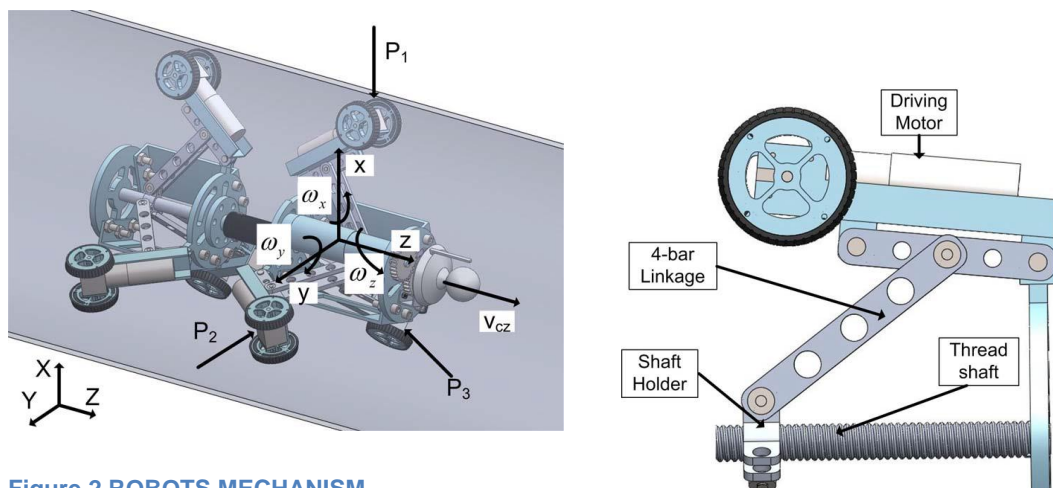


Figure 2 ROBOTS MECHANISM

Rubber wheels make the wheel give bigger wall press forces. Expansion and contraction of the 4-bar linkage is done by rotating the thread shaft which is actuated by expansion motor and thus making the shaft holder move forward and backward. Shaft holder, as shown is used to transfer energy from thread shaft to the 4-bar linkage[2].

In the next paper [3] the author talks about methods to make the entire process of pipe inspection remotely operated.

[3]The full suite of smart technologies implemented inside include the following:

- Improved user interface (more intuitive 2D and 3D displays, providing better visualization and situation awareness),
- Range of input devices (joystick, gamepad, mouse, touch screen controls, microphone, KINECT),
- Robust speed/course controller with independent heading control,
- Higher level of automation (auto compensation of ocean currents & umbilical drag effects),
- Advanced control modes (enabling ROV pilots with average skills to achieve exceptional results),
- ROV high precision dynamic positioning (DP) in absolute earth-fixed frame or relative to ship (yielding improved safety in challenging sea states during critical deployment & recovery stages),
- Improved ROV-ship link (easy synchronization of ROV & ship motions),
- Built-in thruster fault tolerance (optimal control allocation for any thruster configuration),
- Set of pilot assistive tools using sound:
  - Voice commands (pilot generates commands via microphone),
  - Sound navigation (computer provides voice instructions via speakers to assist the pilot to reach the target),
- Real-time control from remote command centre through remote presence (Ethernet/Internet)[3].

Here in this paper [4] the author gives a detailed description as to the transmission system of the robot he explains as to how a H Bridge can be used.[4] H Bridge circuit are used to control the motor. When the switch S1 and S4 is on it runs the motor in one direction and when S3 and S2 is on the motor is driven in the other direction [4].

In the next paper [5] the author explains as to how we can use wireless technology to operate the system.

[5]Two wireless communication systems are proposed based on a long pipe or a manhole. The type-1 wireless communication system is designed. In order to ensure the information that passes through a manhole, the two-way amplifier between antennas is introduced. Another type wireless communication system. Using this wireless system, leakage coaxial cable of simple structure is paved in and along the tested pipe. It is easy to design wireless communication system circuit. Considering integration with inspection robot, it is possible to test a long pipe. It is predicted that it can realize information transmission as long as 200-300 m. Moreover, with leakage coaxial cable, robot movement becomes easy to control

and the reflection effect of pipe wall would be reduced. Besides, it is possible to be applied to mesh cable and parallel two-wire cable. Pipe shape is always complex, it is a real case that some manholes exist in a long pipe. Therefore, the two wireless communication systems should be integrated together, to compensate data transmission loss in pipe[5].

### 3. METHODOLOGY:

This project deal with the modelling and construction of an in pipe inspection robot. The robot is controlled remotely, in order to obtain the pressure and temperature data from within the pipe. The robot is designed to travel within pipes ranging from diameters 7 inches to 12 inches; this can be obtained by expanding and contracting the limbs which hold on the walls of the pipe in order to hold it firmly in place.

The robot which we are modelling consists of two major modules. They are the remote control hub which is used by the operator to control the robot and then the on-board module which is the robot itself. The remote control hub and the robot communicate with each other with the help of RF technology using transmitter and a receiver on each module. Below is the block definition diagram of the overall module.

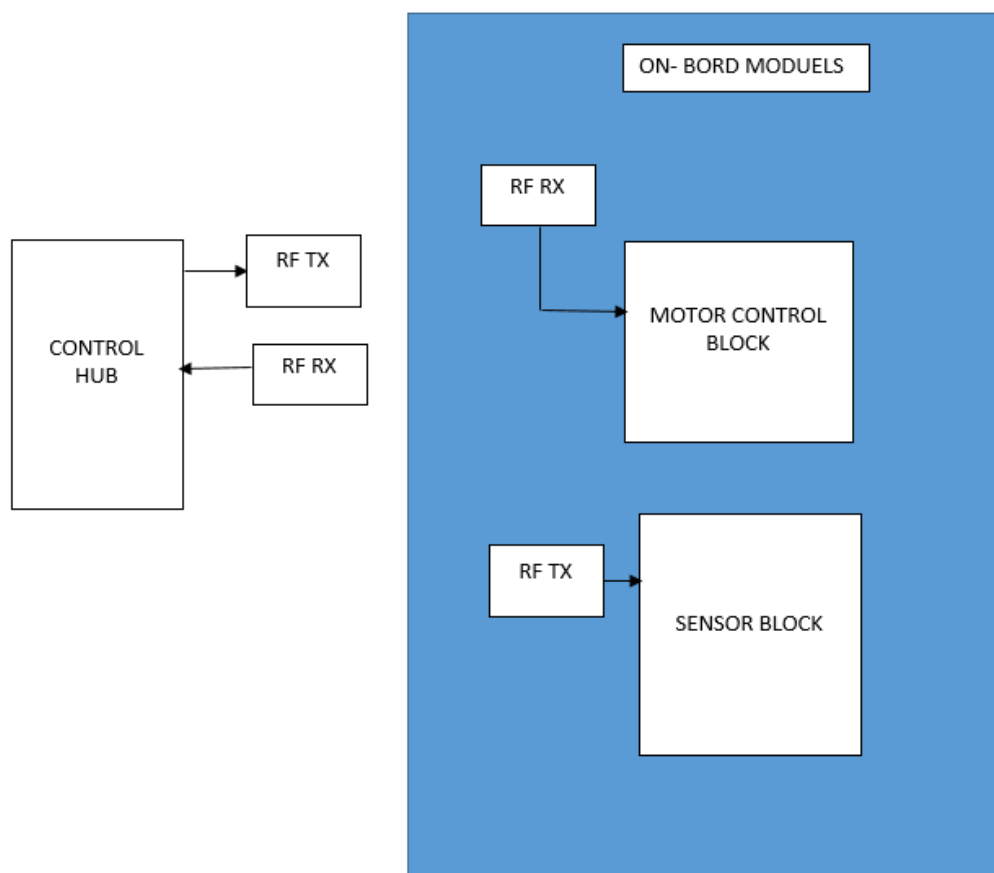


Figure 3 BLOCK DIAGRAM OF THE OVERALL SYSTEM

From the above diagram we can see that the on board module (robot) is divided into two separate block the motor control block and sensor block. The control hub is connected with a RF transmitter and receiver. In the on board module the motor control block is fitted with a receiver to receive commands from the operator to drive the motor in required directions and also to drive the stepper motors to expand and contract the limbs. The sensor block is fitted with a transmitter. The transmitter transmits the temperature pressure and other required and desired data from within the pipe to the control hub for the operator to view and interpret.

In future chapters we shall go into the details of each block and their functions.



## 4. SYSTEM REQUIRMENTS:

### 4.1. Hardware:

#### 4.1.1. Arduino microcontroller

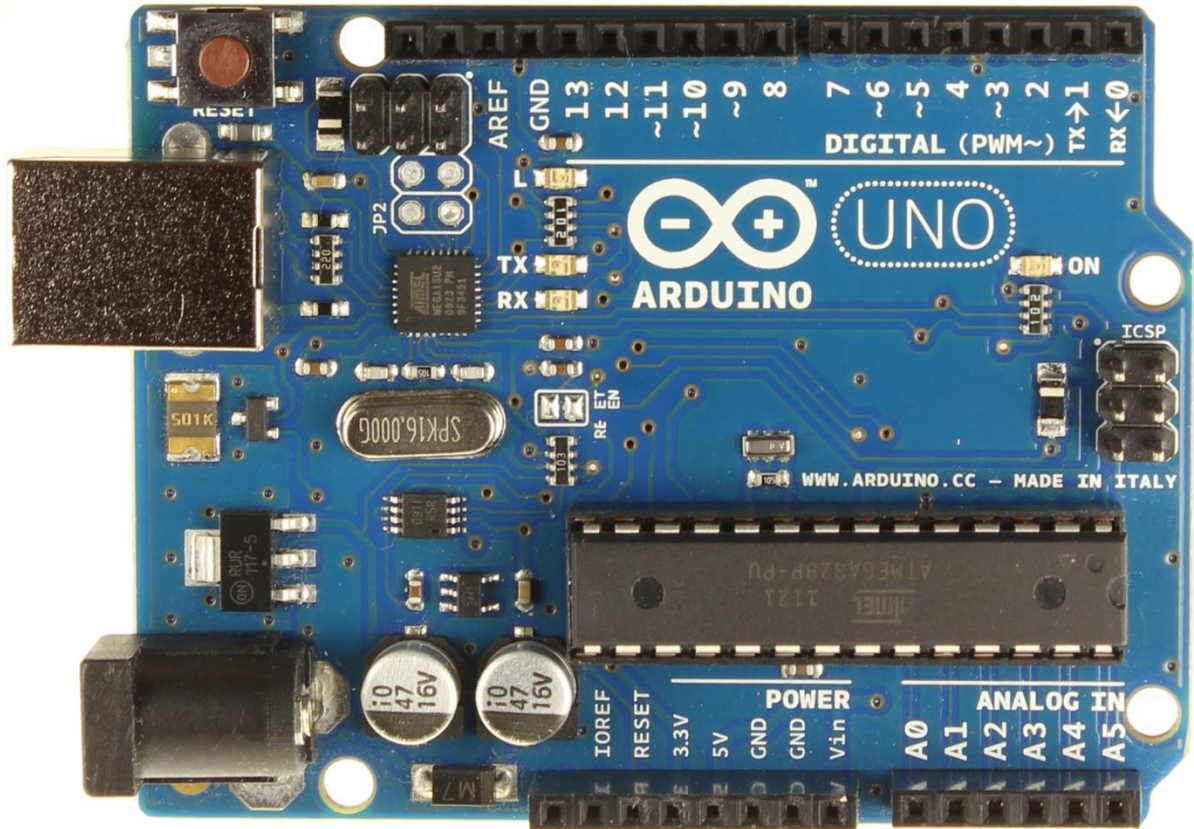


Figure 4 ARDUINO UNO

An Arduino Uno is a microcontroller which runs on Atmega328. It is the main controller for this project. The sensors, driver circuits and RF Tx/Rx are all connected to the controller and based on the signals sent from the controller according to the code the other circuitry performs its operations.

#### 4.1.2. L298N DUAL H BRIDGE

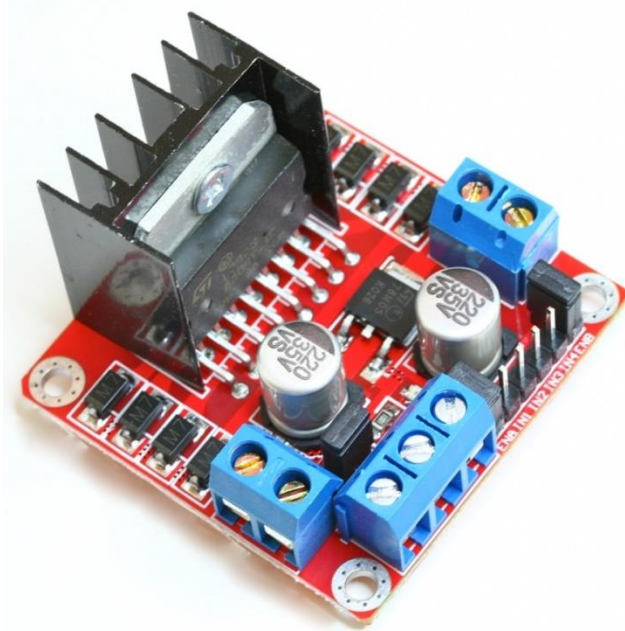


Figure 5 L298N DUAL H BRIDGE

The motor shield is based on L298 chip. The L298 chip is used to drive inductive loads such as DC motors and stepper motors. Each shield can drive or control two DC Motors or one Stepper Motor. The speed and direction of the two DC Motors connected to a particular board can be controlled independently.

#### 4.1.3. INDICATOR SENSOR MODULE L3G4200D, ADXL345, HMC5883L, BMP085

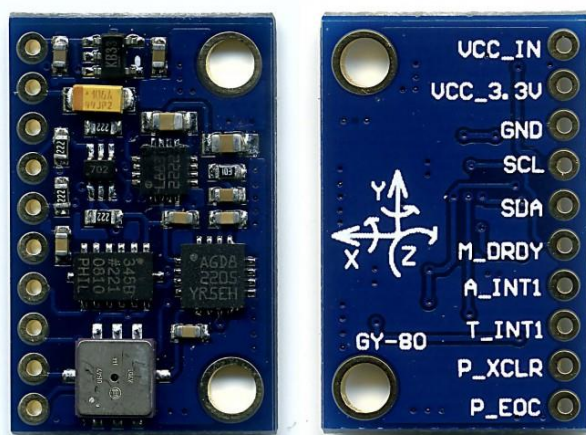
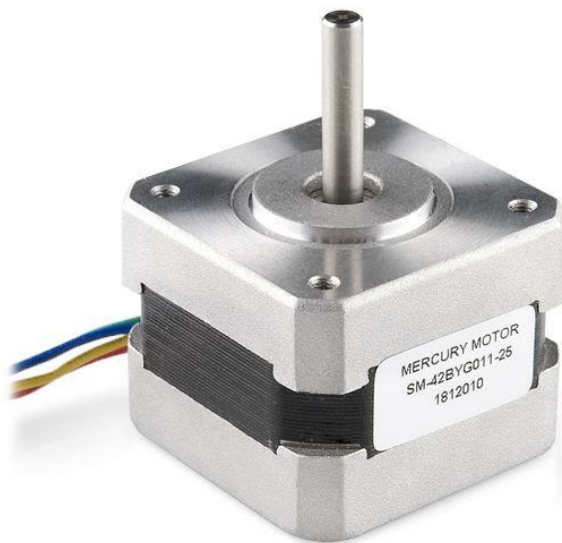


Figure 6 10DOF 9-AXIS ATTITUDE INDICATOR L3G4200D ADXL345 HMC5883L BMP085 MODULE ARDUINO

The Indicator Sensor Module is a sensor device which can be used to measure velocity, orientation and G Force. This is done by using a combination of sensors integrated on to one board. Here an ADXL345 accelerometer, L3G4200D gyroscope and a HMC5883L compass is used to pinpoint orientation and heading. In addition to these sensors on this board it also comes with a BMP085 barometric sensor which can be used to measure the temperature, pressure and altitude.

#### 4.1.4. BIPOLAR HYBRID STEPPER MOTOR



**Figure 7 BIPOLAR HYBRID STEPPER MOTOR**

The stepper motor is used for pressing control of the limbs. A bipolar stepper motor acts on both positive and negative voltage thereby showing both positive and negative polarity. The bipolar stepper motor has two separate windings or coils which generate the required polarities to turn the motor. The torque in a bipolar stepper motor is high as the current flowing through the entire coil. This produces a strong magnetic field unlike the unipolar stepper motor.

#### 4.1.5. DC MOTOR



Figure 8 12V DC MOTOR

A DC Motor is machine which converts direct current into mechanical energy. This is done by the principal that when a current flows a coil a magnetic field is produced which helps in rotating the rotor the DC Motor is an inductive load which is used in the application of motion.

#### 4.1.6. MICRO SD BOARD READER

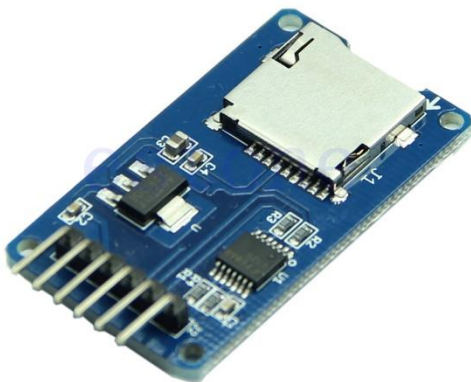


Figure 9 MICRO SD BOARD READER

A micro SD card board is an add-on peripheral for the Arduino as it helps it in data logging. The data obtained by the sensors can be read and written on to a SD card using this board.

#### 4.1.7. RF TX/RX

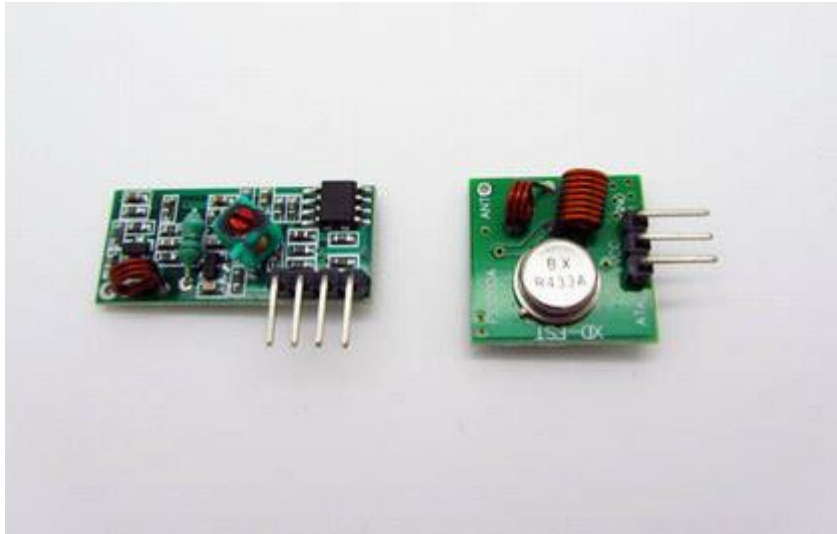


Figure 10 RF TX/RX

The RF Tx/Rx is radio frequency transmitter and receiver which are used to communicate with each other. These two work on radio frequency technology and at the range of 433MHz. The one on the left is the RF receiver and the one on the right is the RF transmitter.

## 4.2. SOFTWARE REQUIREMENTS

- 1 Languages : C, C++ Compiler for Arduino

The Arduino coding language is based on C, C++ language

- 2 Tools : Arduino IDE

This is the tool or interface used to code the Arduino.

- 3 Simulation & Data analysis : MATLAB

MATLAB is a simulation tool which can be used to model a control system, do complex math operations and simulate a real time system to troubleshoot it virtually before it is done in real time.

4 Modelling tool : Modelio

Modelio is a system modelling tool. We would be using sysml to model certain aspects of the inspection robot. Sysml helps us understand the control flow, who can access what and the internal blocks by the use of diagrams.

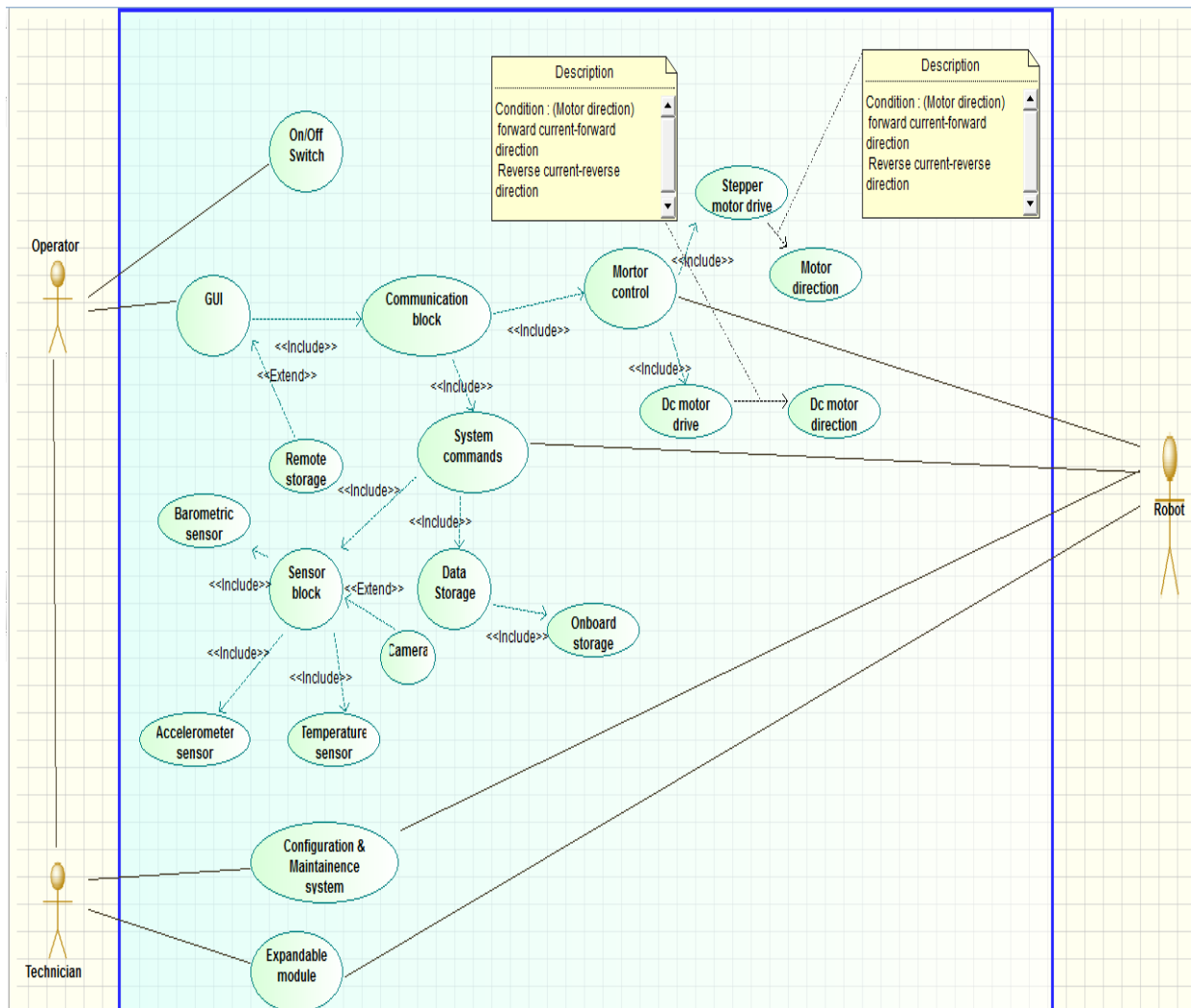
5 Operation system : Microsoft XP, vista, 8, IOS, LINUX

Our robotic control system is compatible with any OS. As the software's required would run on all OS.



## 5. USE CASE DIAGRAM:

This diagram is used to enable us to entrap the entire functionality of the system in a pictorial representation. It helps us depict the system goals. The use case diagram uses subject, actors, use cases and relationships to depict that what is necessary. Figure 3 depicts the use case diagram of our system.



**Figure 11 USE CASE DIAGRAM**

The use case diagram which was modelled in modileo illustrates the functionality of the in pipe inspection robot. Here the user is illustrated as an actor and is denote as the operator. The main aim of this project is to design an in-pipe inspection robot which would help us monitor the internal conditions of the pipe such as temperature pressure etc. These data's are monitored by the operator from a remote control hub or station and he uses this data to control the process or system.

Here we see that the user can activate the inspection robot by switching on the On/Off switch once it is activated he or she has control over the robotic system via a GUI system. The commands given in the GUI is transmitted through a communication block. In our case it would mean RF communication. The user has access to the system commands through the communication block. The system commands block consists of a sensor block which has a wide range of sensors ranging from temperature, pressure, and accelerometer and can be further expanded. These sensors transmit data back to the user. The system command block is also fitted with an on board data storage system which helps store the data obtained from the sensors for backup. The control hub also has a data storage unit for future analysis. The motor control block is connected to the user via the communication block. Here the user gives the command to be executed by the motor driver and the motor driver either the stepper or DC motor in the required direction.

The technician is the only person who can reconfigure the system and expand its modules. He also has access to the other accepts of the system which the operator can access.



## 6. REQUIRMENT DIAGRAM:

This diagram helps to describe the requirements of the system. It also describes the performance, parameters and the interfaces used in the system. Here figure 4 gives the requirement diagram. The Requirement diagram specifies the requirement in hierarchy order includes Satisfy, verify, Refine and Trace. The requirement diagram depict the relationship each sub-system in the overall system.

The requirements of the in pipe inspection robot as per figure 4 is as followed

- Protective housing
- Programmable module
- Expandable module
- Motion control
- Power supply

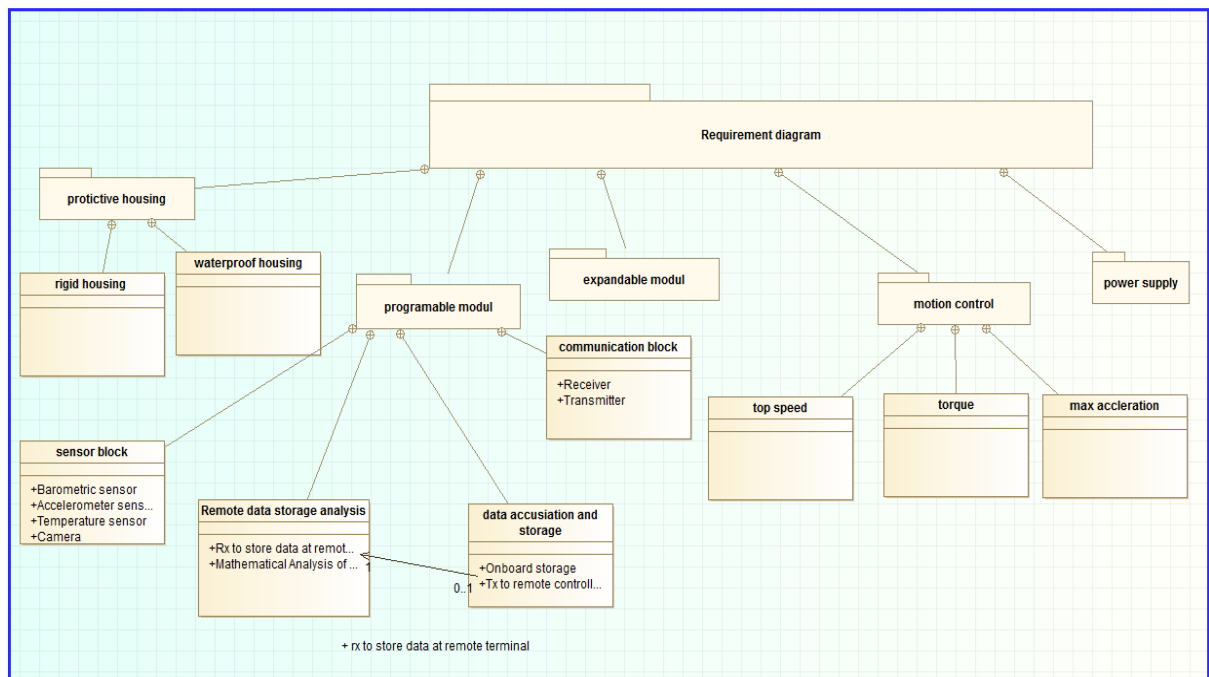


Figure 12 REQUIRMENT DIAGRAM

The top most level of the diagram is the in-pipe inspection robot. After which on the next level comes all its key features.

### **Protective housing:**

Protective housing is required so that the entire system is rigid and that the internal system of the inspection robot doesn't get compromised due to external conditions. The two main

requirements of this system I that the housing should be rigid and that it should be water proof so that it doesn't affect the internal circuitry if it is travelling through pipes with fluids.

### **Programmable module:**

The programing module is the main part of the data accusation and distribution of the system. It is divided into the following key components.

Sensor block: This block consists of the required sensors such as barometric sensor temperature sensors etc. These sensors are used to collect the required data from the environment.

Data accusation and storage: This is the data storage block which is present on the robot this helps to log the data which is obtained from the environment for future analysis.

Remote data storage and analysis: This requirement is that the data is needed to be stored at the remote controller end so that it can be analysed and saved for future reference.

Communication Block: This is a very important requirement for the robotic system, because without communication between the operator and the robot we would not be able to control the inspection robot. Here the medium used to communicate between the robot and the control hub is RF (Radio Frequency) communication.

### **Expandable module:**

This helps or rather enables us to add on further modules on to the system if required in the future.

### **Motion control:**

This is another very important block and it deals with locomotion of the system. It consists of motor driver circuits and motors and the main requirements of this block would be the torque required, top speed and max acceleration.

### **Power supply:**

The next key requirement is that required amount of power should be supplied for the circuitry for it to function properly. Each Arduino takes us to 5v max and the dc motors are 12v dc motors which take about 0.2 amps of current.

## 7. BLOCK DIAGRAM:

The block diagram describes the main features which make up the system and gives the basic outline their classification and composition. Figure 5 is the block diagram for the pipe inspection robot with its key components explaining which sub system is linked or grouped with what and which components are inter related. From figure 5 we can see that the top most level represents the entire system on the whole and as we go downwards the hierarchy, we see in the next level the main features of the system which are the protective housing, programming module, motor control, gui and the expandable module. When we go down one more level we can see the different sub systems grouped under each feature and how they are related.

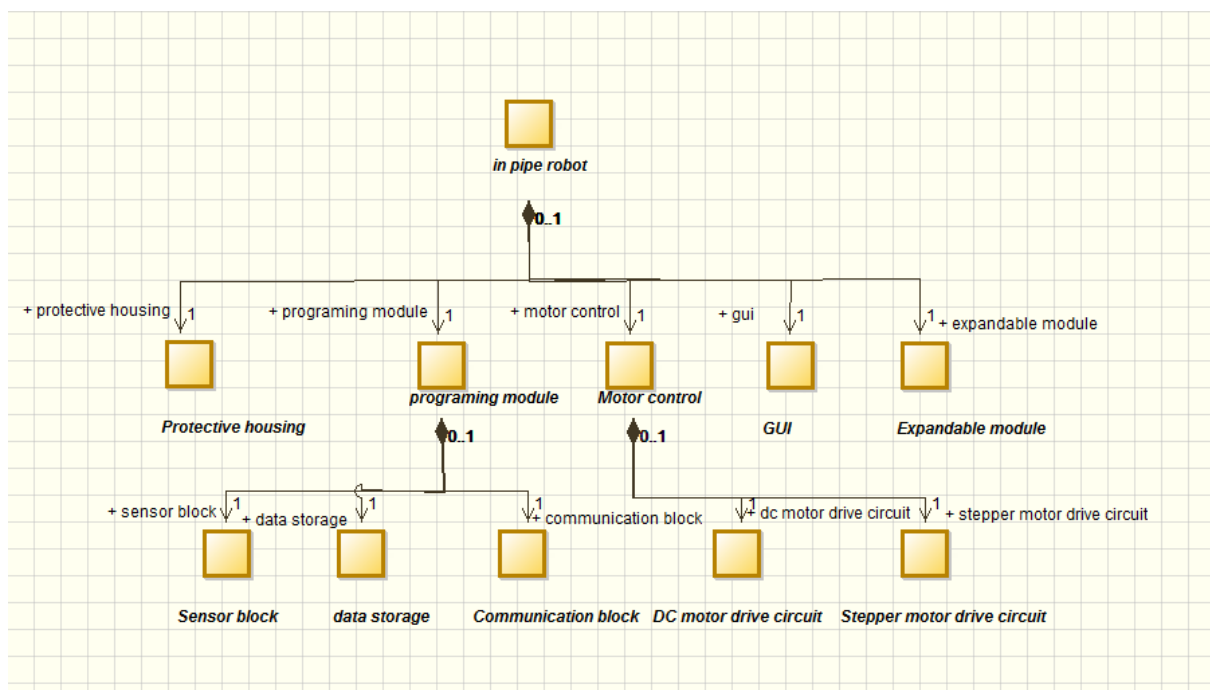


Figure 13 SYSML BLOCK DIAGRAM

We can see that under the programming block we have the sensor block which houses all the sensors and their operations. The data storage block which is used to store data which is recorded by the sensor block and the communication block which is used to communicate between the tele operator or remote operator and the robot. Here in this project the technology used for communication is RF technology. Under the motor control block we have two main sub systems which are the DC motor driver circuit and Stepper motor drive circuit.

## 8. BAROMETRIC SENSOR [APPENDIX 5]

### 8.1. INTRODUCTION

In our project sensor data such as pressure and temperature data is obtained from the IMU board which is connected to the arduino. The chip on the IMU board which is responsible for attaining these sensor data values is the BMP085 barometric sensor. This sensor is mounted on to the IMU which is in turn connected to the Arduino.

A BMP085 sensor is a high precision and low powered sensor. The BMP08 is a barometric sensor used in mobile applications.

This sensor has an accuracy of 2.5hPa. The level of noise present in this sensor is 0.03hPa this is similar to 0.25m change in altitude. This sensor is a high performance sensor with good accuracy level. As said before this sensor uses very low amount of power which is about 3μA. These are the prime features of the sensor that makes it apt for mobile application.

The BMP085 sensor works on piezo-resistive MEMS technology. It is an LCC otherwise known as lead less chip. It is very slim and has 8 pins and made of ceramic. It is a strong and robust chip. This sensor can be mounted on to the board or controller and connects to the controller via  $I^2C$  bus.

### 8.2. APPLICATIONS

- It can be used for weather forecasting.
- It can be used for vertical velocity indicator to identify the altitude.
- It can act as a GPS enhancement.
- It can also help to control the power supplied to fans as per the change in temperature.

### 8.3. SPECIFICATIONS AND RANGE

| Specifications                 | range  |
|--------------------------------|--|
| Pressure                       | 300 – 1100hPa                                    |
| Noise present in pressure data | 0.06hPa – 0.03hPa (from low power to high power) |

|   |   |
|---|---|
| Noise present in altitude data          | 0.5m – 0.25m (from low power to high power) |
| Accuracy of pressure reading            | ±2.5hPa                                     |
| Accuracy of temperature reading         | ±2 °C                                       |
| Average current (1Hz data refresh rate) | 3µA - 12µA (from low power to high power)   |
| Supply voltage VDD                      | 1.62V – 3.6V                                |
| Operational temperature range           | -40 °C - +85 °C                             |

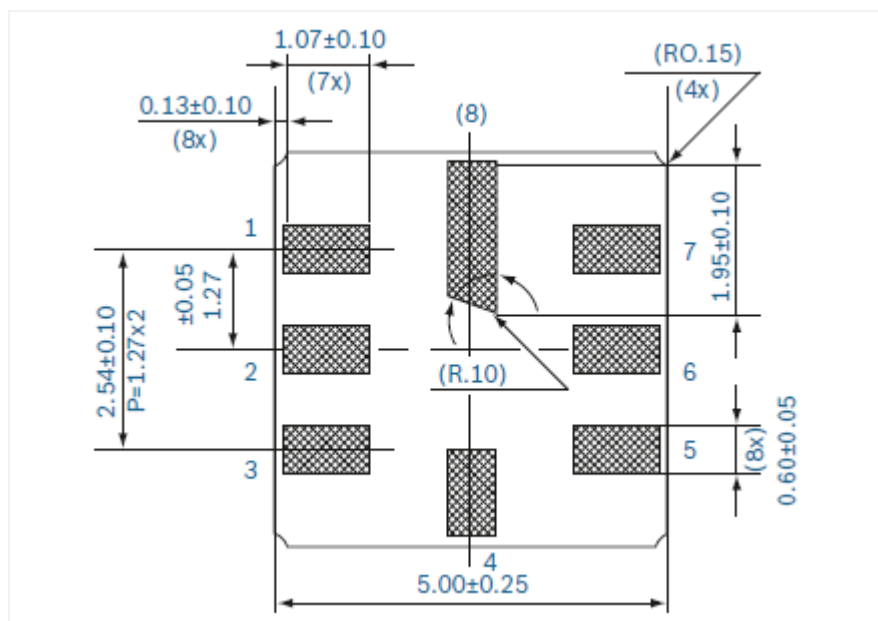


Table 1 PIN CONFIGURATON BMP085

| PIN | NAME      | FUNCTION                      |
|-----|-----------|-------------------------------|
| 1   | GND       | Ground                        |
| 2   | EOC       | End of conversion output      |
| 3   | $V_{DDA}$ | Power supplied to analog      |
| 4   | $V_{DDD}$ | Power supplied to DIGITAL     |
| 5   | NC        | Not connected                 |
| 6   | SCL       | $I^2C$ serial bus clock input |
| 7   | SDA       | $I^2C$ serial bus data        |
| 8   | XCLR      | Master clear input            |

## 8.5. WIRING TO ARDUINO :

As we know the BMP085 is a ultra thin cemic LC which is mounted on to the imu or indicator sensor module. So the indicator sensor has an intergation of different sensors and these have to be mounted on to the Ardunio in order to read from the BMP085 barometric sensor.

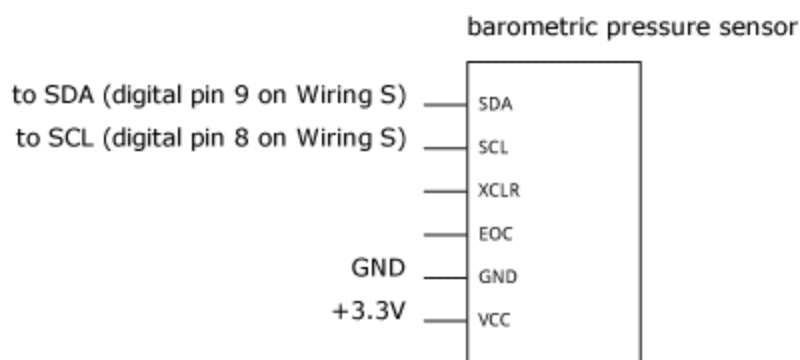


Figure 15 WIRING FOR BAROMETRIC SENSOR

From figure 14 we see that the SDA pin of the sensor is connected to the digital pin 9 of the Arduino and the SCL pin on thee sensor is connected to the digital pin 8 on the Arduino.

GND is connected to the GND on the Arduino and VCC is connected to the +3.3V pin on the Arduino.

## 8.6. SAMPLE CODE

```
#include <Wire.h>
#include <Adafruit_BMP085.h>

Adafruit_BMP085 bmp;

void setup()
{
  Serial.begin(9600);
  if (!bmp.begin())
  {
    Serial.println("Could not find a valid BMP085 sensor, check wiring!");
    while (1)
    {
    }
  }
}

void loop()
{
  Serial.print("Pressure = ");
  Serial.print(bmp.readPressure());
  Serial.println(" Pa");

  Serial.print("Temperature = ");
  Serial.print(bmp.readTemperature());
  Serial.println(" *C");

  // Calculate altitude assuming 'standard' barometric
  // pressure of 1013.25 millibar = 101325 Pascal
  Serial.print("Altitude = ");
  Serial.print(bmp.readAltitude());
  Serial.println(" meters");
```

```
Serial.println();  
delay(500);  
}
```



## 8.7. OUTPUT:

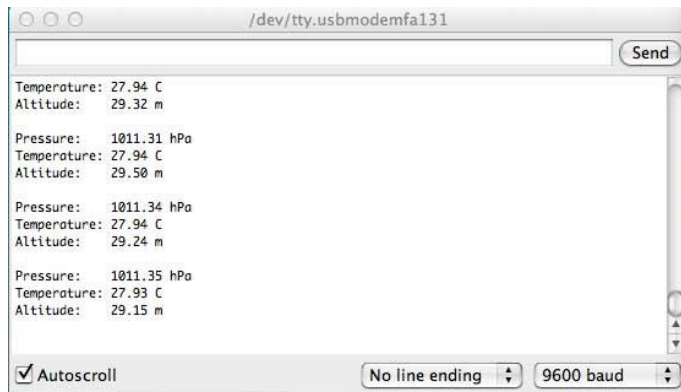


Figure 16 OUTPUT FOR BMP085 SENSOR

## 8.8. CONCLUSION

Thus based upon the above given reasons and specs the BMP085 sensor has been chosen for its low power consumption, rigidity and accuracy.

## 9. IMU (Inertial Measurement Unit)

### 9.1. INTRODUCTION:

IMU sensor is a sensor used to measure the velocity, orientation and gravitational force of any particular object. An IMU consists of a combination of three sensors which are accelerometer, gyroscope and compass. Here in this IMU sensor the accelerometer used is Gyroscope used is L3G4200D, Accelerometer is ADXL345 and compass is HMC5883L. If we analyse all these data together we can obtain the orientation and heading of the robot.

### 9.2. ACCELEROMETER ADXL345 [APPENDIX 4]

An ADXL345 is an ultra slim and small 3 axis accelerometer with 14 leads, which consumes very low power. It is a digital accelerometer with very high resolution measurement of about  $\pm 16$  g. Output is digital data of 16 bits. It communicates to the microcontroller through  $I^2C$  digital interface.

ADXL345 accelerometer sensor is well suited for remote operation or mobile projects. It can be used to measure static acceleration and dynamic acceleration. Here static acceleration such as tilt sensing and measurement and dynamic such as motion of object all with respect to gravity. Due to the high resolution available we can take measurements of inclination change with accuracy up to  $1.0^\circ$ .

There are several added features in this accelerometer. It can be used to find the activity or inactivity of the mobile robot; it can be used to detect presence and absence or lack of motion. In addition we can also find or denote single tap and double tap. Or use it to limit the motion of the mobile robot along an axis within a fixed limit.

#### 9.2.1. FEATURES

- Power consumption :  
Measurement mode: 40  $\mu$ A  
Standby mode: 0.1  $\mu$ A  
Voltage: 2.5V
- User can select the required resolution.

- Tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- It requires a voltage supply of 2V – 3.6V
- SPI and I2C digital interface.
- Operating temperature: –40°C to +85°C
- It can resist a shock of 10,000g

### 9.2.2. APPLICATION:

- Mobile phones
- Medical devices
- Gaming devices
- pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive (HDD) protection
- Fitness equipment

### 9.2.3. ABSOLUTE POWER RATING

Table 2 ABSOLUTE POWER RATING ADXL345

| PARAMETERS                  | RATING          |
|-----------------------------|-----------------|
| Acceleration                | 10,000g         |
| $V_s$                       | -0.3V - +3.6V   |
| $V_{DD\ I/O}$               | -0.3V - +3.6V   |
| DIGITAL PINS                | -0.3V - +3.6V   |
| Other pins                  | -0.3V - +3.6V   |
| TEMPERATURE RANGE (POWERED) | -40°C to +105°C |
| TEMPERATURE RANGE (STORAGE) | -40°C to +105°C |

## 9.2.4. THERMAL RESISTANCE

Table 3 THERMAL RESISTANCE ADXL345

| PACKAGE TYPE       | $\theta_{JA}$ | $\theta_{JC}$ | DEVICE WEIGHT |
|--------------------|---------------|---------------|---------------|
| 14-TERMINAL<br>LGA | 150°C/W       | 85°C/W        | 20mg          |

## 9.2.5. PIN CONFIGURATION:

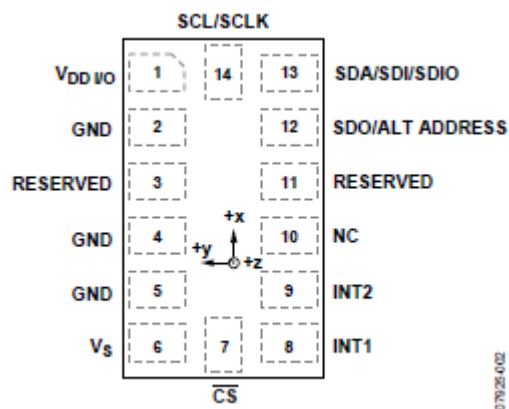


Figure 17 ADXL345 PIN CONFIGURATION

Table 4 PIN CONFIGURATION ADXL345

| Table 4. Pin Function Descriptions Pin No. | Mnemonic            | Description                                   |
|--|---------------------|---|
| 1  | V <sub>DD I/O</sub> | Digital I/O Supply Voltage.                   |
| 2  | GND                 | ground.                                       |
| 3  | Reserved            | connected to V <sub>s</sub> or left open.     |
| 4  | GND                 | ground.                                       |
| 5  | GND                 | ground.                                       |
| 6  | V <sub>s</sub>      | Supply Voltage.                               |
| 7  | CS                  | Chip Select.                                  |
| 8  | INT1                | Interrupt 1 Output.                           |
| 9  | INT2                | Interrupt 2 Output.                           |
| 10   | NC                  | Not Internally Connected.                     |
| 11   | Reserved            | connected to ground or left open.             |
| 12   | SDO/ALT ADDRESS     | Serial Data Output/Alternate I <sup>2</sup> C |

|    |              |  |
|----|--------------|--|
|    |              | Address Select.  |
| 13 | SDA/SDI/SDIO | Serial Data (I <sup>2</sup> C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire). |
| 14 | SCL/SCLK     | Serial Communications Clock.   |

### 9.2.6. ADXL345 TO ARDUNIO

This chip is fixed on a board and then connected with an Arduino the wirings for connexion is given below

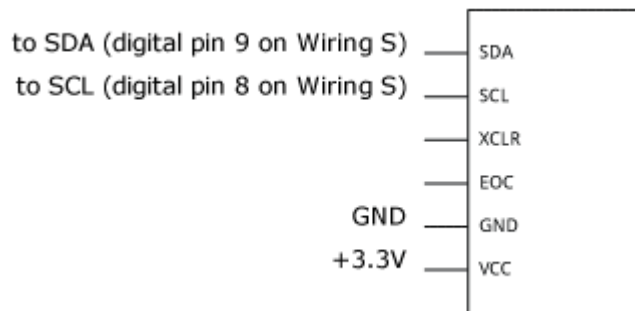


Figure 18 ADXL345 WIRING

## 9.3. GYROSCOPE L3G4200D [APPENDIX 2]

### 9.3.1. INTRODUCTION:

This is a low powered 3 axis gyroscope. This helps us view the stability of the system from a zero rate level. It communicates using I<sup>2</sup>C and SPI protocol.

### 9.3.2. FEATURES:

- Supply voltage – 2.4V to 3.6V
- Two digital output lines.
- Three selectable full scales
- Stable over a wide temp range
- It has a HPF in which user can select the bandwidth.

### 9.3.3. PIN DISCRIPTION:

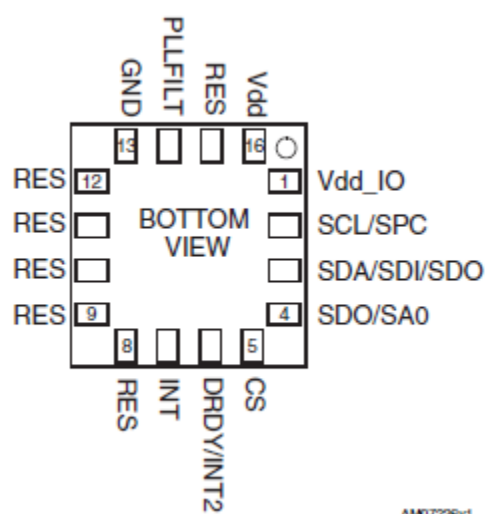


Figure 19 PIN CONFIGURATION FOR L3G4200D

Table 5 PIN CONFIGURATION FOR L3G4200D

| Pin# | Name              | Function  |
|------|-------------------|---|
| 1    | Vdd_IO            | Power supply for I/O pins   |
| 2    | SCL<br>SPC        | I <sup>2</sup> C serial clock (SCL)<br>SPI serial port clock (SPC)  |
| 3    | SDA<br>SDI<br>SDO | I <sup>2</sup> C serial data (SDA)<br>SPI serial data input (SDI)<br>3-wire interface serial data output (SDO)  |
| 4    | SDO<br>SA0        | SPI serial data output (SDO)<br>I <sup>2</sup> C least significant bit of the device address (SA0)  |
| 5    | CS                | SPI enable<br>I <sup>2</sup> C/SPI mode selection (1:SPI idle mode / I <sup>2</sup> C communication enabled; 0: SPI communication mode / I <sup>2</sup> C disabled) |
| 6    | DRDY/INT2         | Data ready/FIFO interrupt   |
| 7    | INT1              | Programmable interrupt  |
| 8    | Reserved          | Connect to GND  |
| 9    | Reserved          | Connect to GND  |
| 10   | Reserved          | Connect to GND  |
| 11   | Reserved          | Connect to GND  |
| 12   | Reserved          | Connect to GND  |
| 13   | GND               | 0 V supply  |
| 14   | PLLFILT           | Phase-locked loop filter (see <a href="#">Figure 3</a> )  |
| 15   | Reserved          | Connect to Vdd  |
| 16   | Vdd               | Power supply  |

### 9.3.4. ELECTRICAL CHARACTERISTICS

Table 6 ELECTRICAL CHARACTERISTICS L3G4200D

| Symbol | Parameter                                   | Test condition                  | Min. | Typ. <sup>(2)</sup> | Max.    | Unit |
|--------|---|---------------------------------|------|---------------------|---------|------|
| Vdd    | Supply voltage                              |                                 | 2.4  | 3.0                 | 3.6     | V    |
| Vdd_IO | I/O pins supply voltage <sup>(3)</sup>      |                                 | 1.71 |                     | Vdd+0.1 | V    |
| Idd    | Supply current                              |                                 |      | 6.1                 |         | mA   |
| IddSL  | Supply current in sleep mode <sup>(4)</sup> | Selectable by digital interface |      | 1.5                 |         | mA   |
| IddPdn | Supply current in power-down mode           | Selectable by digital interface |      | 5                   |         | μA   |
| Top    | Operating temperature range                 |                                 | -40  |                     | +85     | °C   |

### 9.3.5. TEMPERATURE CHARECTERSTICS

Table 7 TEMPERATURE CHARECTERSTICS L3G4200D

| Symbol | Parameter  | Test condition | Min. | Typ. <sup>(2)</sup> | Max. | Unit     |
|--------|--|----------------|------|---------------------|------|----------|
| TSDr   | Temperature sensor output change vs. temperature |                |      | -1                  |      | °C/digit |
| TODR   | Temperature refresh rate                         |                |      | 1                   |      | Hz       |
| Top    | Operating temperature range                      |                | -40  |                     | +85  | °C       |

## 9.4. Compass HMC5883L [APPENDIX 3]

### 9.4.1. INTRODUCTION

The compass is made by Honeywell it is ultra-thin small and light weight surface mount 16 pins LLC designed for low field magnetic sensing. This sensor module uses I2C serial bus for communication.

### 9.4.2. Features

- 3 Axis Magnetoresistive Sensor with ASCII

- It has a 12 bit ADC and a low noise AMR sensor.
- It uses I2C Digital interface to communicate.
- It is a leadless surface mounted package.
- It uses 2.16 to 3.6v as supply voltage.
- It has a wide magnetic range of  $\pm 80\text{e}$ .

#### 9.4.3. PIN CONFIGURATION.

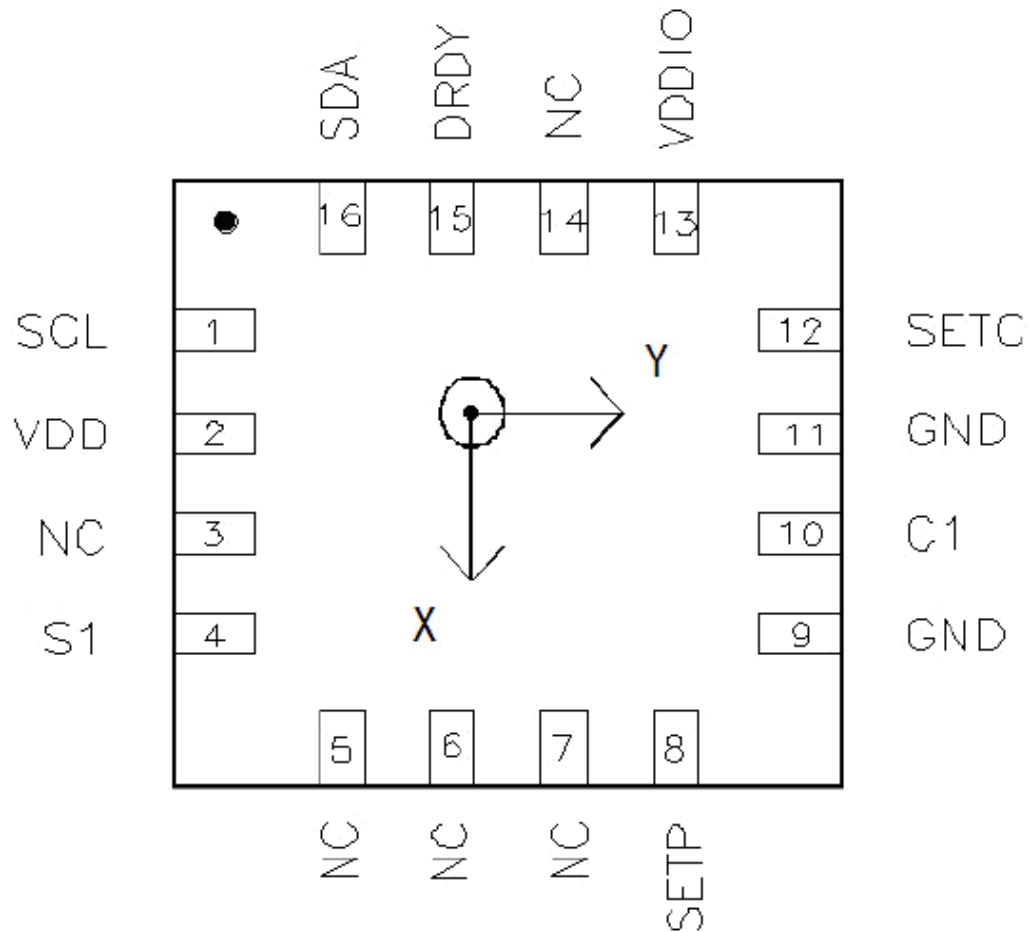


Figure 20 HMC5883L PIN CONFIGURATION



**Table 8 HMC5883L PIN CONFIGURATION**

| Pin | Name  | Description   |
|-----|-------|---|
| 1   | SCL   | Serial Clock – I <sup>2</sup> C Master/Slave Clock  |
| 2   | VDD   | Power Supply (2.16V to 3.6V)  |
| 3   | NC    | Not to be Connected   |
| 4   | S1    | Tie to VDDIO  |
| 5   | NC    | Not to be Connected   |
| 6   | NC    | Not to be Connected   |
| 7   | NC    | Not to be Connected   |
| 8   | SETP  | Set/Reset Strap Positive – S/R Capacitor (C2) Connection  |
| 9   | GND   | Supply Ground   |
| 10  | C1    | Reservoir Capacitor (C1) Connection   |
| 11  | GND   | Supply Ground   |
| 12  | SETC  | S/R Capacitor (C2) Connection – Driver Side   |
| 13  | VDDIO | IO Power Supply (1.71V to VDD)  |
| 14  | NC    | Not to be Connected   |
| 15  | DRDY  | Data Ready, Interrupt Pin. Internally pulled high. Optional connection. Low for 250 $\mu$ sec when data is placed in the data output registers. |
| 16  | SDA   | Serial Data – I <sup>2</sup> C Master/Slave Data  |

#### 9.4.4. SPECIFICATIONS

**Table 9 SPECIFICATIONS HMC5883L**

| Characteristics      | Conditions*                            | Min  | Typ | Max     | Units   |
|----------------------|--|------|-----|---------|---------|
| <b>Power Supply</b>  |  |      |     |         |         |
| Supply Voltage       | VDD Referenced to AGND                 | 2.16 | 2.5 | 3.6     | Volts   |
|                      | VDDIO Referenced to DGND               | 1.71 | 1.8 | VDD+0.1 | Volts   |
| Average Current Draw | Idle Mode                              | -    | 2   | -       | $\mu$ A |
|                      | Measurement Mode (7.5 Hz ODR;          | -    | 100 | -       | $\mu$ A |
|                      | No measurement average, MA1:MA0 = 00)  |      |     |         |         |
|                      | VDD = 2.5V, VDDIO = 1.8V (Dual Supply) |      |     |         |         |
|                      | VDD = VDDIO = 2.5V (Single Supply)     |      |     |         |         |

|                       |                                 |     |  |      |       |
|-----------------------|---------------------------------|-----|--|------|-------|
| ESD Voltage           | Human Body Model (all pins)     |     |  | 2000 | Volts |
|                       | Charged Device Model (all pins) |     |  | 750  |       |
| Operating Temperature | Ambient                         | -30 |  | 85   | °C    |
| Storage Temperature   | Ambient, unbiased               | -40 |  | 125  | °C    |

#### **Absolute Maximum Ratings** (\* Tested at 25°C except stated otherwise.)

| Characteristics      | Min  | Max | Units |
|----------------------|------|-----|-------|
| Supply Voltage VDD   | -0.3 | 4.8 | Volts |
| Supply Voltage VDDIO | -0.3 | 4.8 | Volts |

#### 9.4.5. SAMPLE CODE

```
#include <Wire.h>
```

```
#include <I2Cdev.h>

#include <Narcoleptic.h>

#include <Adafruit_BMP085.h>

#include <Adafruit_Sensor.h>

#include <Adafruit_ADXL345_U.h>

#include <L3G4200D.h>

#include "HMC5883L.h"


Adafruit_BMP085 bmp;

Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

L3G4200D gyro;

HMC5883L mag;


int16_t avx, avy, avz;

int16_t mx, my, mz;


#define LED_PIN 13

bool blinkState = false;


void setup()
{
  Wire.begin();

  Serial.begin(9600);

  if (!bmp.begin())
  {
    Serial.println("Could not find a valid BMP085 sensor, check wiring!");

    while (1)
    {

```

```

    }
}

Serial.println("Accelerometer Test");
Serial.println("");
if(!accel.begin())
{
    /* There was a problem detecting the ADXL345 ... check your connections */
    Serial.println("Ooops, no ADXL345 detected ... Check your wiring!");
    while(1);
}

/* Set the range to whatever is appropriate for your project */
accel.setRange(ADXL345_RANGE_16_G);
// displaySetRange(ADXL345_RANGE_8_G);
// displaySetRange(ADXL345_RANGE_4_G);
// displaySetRange(ADXL345_RANGE_2_G);

/* Display some basic information on this sensor */
displaySensorDetails();

/* Display additional settings (outside the scope of sensor_t) */
displayDataRate();
displayRange();
Serial.println("");

// initialize device
Serial.println("Initializing I2C devices...");

```

```

gyro.initialize();

// verify connection
Serial.println("Testing device connections...");

Serial.println(gyro.testConnection() ? "L3G4200D connection successful" : "L3G4200D
connection failed");

// configure LED for output
pinMode(LED_PIN, OUTPUT);

// data seems to be best when full scale is 2000
gyro.setFullScale(2000);

Serial.println("Initializing I2C devices...");
mag.initialize();

// verify connection
Serial.println("Testing device connections...");

Serial.println(mag.testConnection() ? "HMC5883L connection successful" : "HMC5883L
connection failed");

// configure Arduino LED for
pinMode(LED_PIN, OUTPUT);

}

void loop()
{
    //Narcoleptic.delay(500); // During this time power consumption is minimised

```

```

char temp = (bmp.readTemperature()) ;
char Pa = (bmp.readPressure()*0.000145037738);

Serial.print("Temperature = ");
Serial.print(temp);
Serial.println(" *C");
Serial.print("Pressure = ");
Serial.print(Pa);
Serial.println(" Psi");


Serial.println();
delay(500);


sensors_event_t event;
accel.getEvent(&event);


char acc_x = event.acceleration.x;
char acc_y = event.acceleration.y;
char acc_z = event.acceleration.z;


char pitch = atan(acc_x/sqrt(pow(acc_y,2) + pow(acc_z,2)));
char roll = atan(acc_y/sqrt(pow(acc_x,2) + pow(acc_z,2)));


pitch = pitch*(180.0/PI);
roll = roll*(180.0/PI);


/* Display the results (acceleration is measured in m/s^2) */

```

```
Serial.print("X: ");  
Serial.print(acc_x);  
Serial.print(" ");  
Serial.print("Y: ");  
Serial.print(acc_y);  
Serial.print(" ");  
Serial.print("Z: ");  
Serial.print(acc_z);  
Serial.print(" ");  
Serial.println("m/s^2 ");  
Serial.println(" ");
```

```
Serial.print("Pitch :\\t");  
Serial.print(pitch);  
Serial.print("\\t");  
Serial.print("Roll :\\t");  
Serial.println(roll);  
delay(500);
```

```
gyro.getAngularVelocity(&avx, &avy, &avz);
```

```
Serial.print("angular velocity:\\t");  
Serial.print(avx);  
Serial.print("\\t");  
Serial.print(avy);  
Serial.print("\\t");  
Serial.println(avz);  
Serial.println(" ");
```

```

// blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED_PIN, blinkState);


// read raw heading measurements from device
mag.getHeading(&mx, &my, &mz);


// display tab-separated gyro x/y/z values
Serial.print("mag:\t");

Serial.print("x: ");
Serial.print(mx);
Serial.print("\t");
Serial.print("\t");
Serial.print("y: ");
Serial.print(my);
Serial.print("\t");
Serial.print("\t");
Serial.print("z: ");
Serial.println(mz);
Serial.println(" ");
// Serial.print("\t");


// To calculate heading in degrees. 0 degree indicates North
int heading = atan2(my, mx);

if(heading < 0)

    heading += 2 * M_PI;

```

```

Serial.print("heading:\t");

Serial.println(heading * 180/M_PI);

Serial.println(" ");


// blink LED to indicate activity
blinkState = !blinkState;
digitalWrite(LED_PIN, blinkState);
}


void displaySensorDetails(void)
{
    sensor_t sensor;
    accel.getSensor(&sensor);
    Serial.println("-----");
    Serial.print ("Sensor:   ");
    Serial.println(sensor.name);
    Serial.print ("Driver Ver: ");
    Serial.println(sensor.version);
    Serial.print ("Unique ID:  ");
    Serial.println(sensor.sensor_id);
    Serial.print ("Max Value:  ");
    Serial.print(sensor.max_value);
    Serial.println(" m/s^2");
    Serial.print ("Min Value:  ");
    Serial.print(sensor.min_value);
    Serial.println(" m/s^2");
    Serial.print ("Resolution: ");
    Serial.print(sensor.resolution);

```



```
Serial.println(" m/s^2");  
Serial.println("-----");  
Serial.println("");  
delay(500);  
}
```

```
void displayDataRate(void)  
{  
    Serial.print ("Data Rate:  ");  
  
    switch(accel.getDataRate())  
    {  
        case ADXL345_DATARATE_3200_HZ:  
            Serial.print ("3200 ");  
            break;  
        case ADXL345_DATARATE_1600_HZ:  
            Serial.print ("1600 ");  
            break;  
        case ADXL345_DATARATE_800_HZ:  
            Serial.print ("800 ");  
            break;  
        case ADXL345_DATARATE_400_HZ:  
            Serial.print ("400 ");  
            break;  
        case ADXL345_DATARATE_200_HZ:  
            Serial.print ("200 ");  
            break;  
        case ADXL345_DATARATE_100_HZ:
```

```
Serial.print ("100 ");  
break;  
case ADXL345_DATARATE_50_HZ:  
Serial.print ("50 ");  
break;  
case ADXL345_DATARATE_25_HZ:  
Serial.print ("25 ");  
break;  
case ADXL345_DATARATE_12_5_HZ:  
Serial.print ("12.5 ");  
break;  
case ADXL345_DATARATE_6_25HZ:  
Serial.print ("6.25 ");  
break;  
case ADXL345_DATARATE_3_13_HZ:  
Serial.print ("3.13 ");  
break;  
case ADXL345_DATARATE_1_56_HZ:  
Serial.print ("1.56 ");  
break;  
case ADXL345_DATARATE_0_78_HZ:  
Serial.print ("0.78 ");  
break;  
case ADXL345_DATARATE_0_39_HZ:  
Serial.print ("0.39 ");  
break;  
case ADXL345_DATARATE_0_20_HZ:  
Serial.print ("0.20 ");
```

```
        break;

    case ADXL345_DATARATE_0_10_HZ:

        Serial.print ("0.10 ");

        break;

    default:

        Serial.print ("???? ");

        break;

    }

    Serial.println(" Hz");
}
```

```
void displayRange(void)
{
    Serial.print ("Range:      +/- ");

    switch(accel.getRange())
    {
    case ADXL345_RANGE_16_G:

        Serial.print ("16 ");

        break;

    case ADXL345_RANGE_8_G:

        Serial.print ("8 ");

        break;

    case ADXL345_RANGE_4_G:

        Serial.print ("4 ");

        break;

    case ADXL345_RANGE_2_G:

        Serial.print ("2 ");
```

```

    break;

default:

    Serial.print ("?? ");

    break;

}

Serial.println(" g");

}

```

#### 9.4.6. OUTPUT

```

set up
Accelerometer Test

Initializing I2C devices...
Testing device connections...
L3G4200D connection successful
Initializing I2C devices...
Testing device connections...
HMC5883L connection successful
Initializing SD Card Data Logger
Card INITIALIZED
ID, Temp, Pa
-----
Temperature = 23 *C
Pressure = 14 Psi

X: 10  Y: 0  Z: 0  m/s^2

Pitch : 57      Roll : 0
angular velocity:    16      0      -3

mag:    x: -409      y: 265      z: -442

      heading:      114

1,23,14

```

Figure 21 OUTPUT FOR THE IMU

### **9.5. CONCLUSION:**

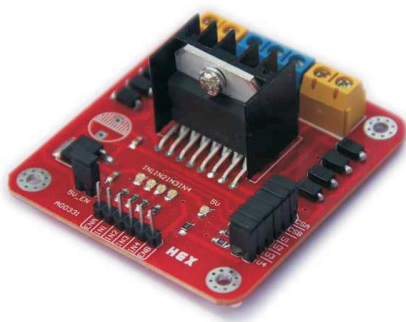
Since we require all these sensors it would not seem economical to use each of them separately as it would occupy a lot of pins. So in this case to be efficient we use the IMU sensor which has all these sensors integrated on to one single board.

## 10. Motion control unit

### 10.1. L298n

#### 10.1.1. INTRODUCTION:

For the motion control we connect an L298n dual h bridge board to an Arduino. This board is used to control the direction and speed of the motor. The board consists of a full dual h bridge this is used to drive any inductive load such as a dc motor or stepper motor.



#### 10.1.2. FEATURES:

Operating voltage – 4volts to 35 volts

It can drive two DC Motor or one stepper motor

Max current 4A – 4A

Voltage supply – 5V

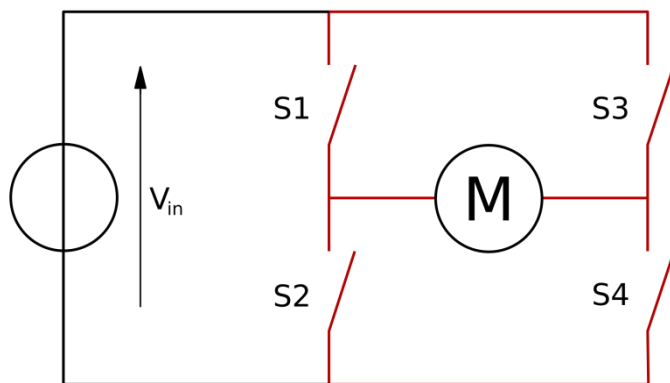
Max power: 25w

#### 10.1.3. WORKING

The H Bridge is an electronic circuit which helps drive an inductive load in the preferred direction. As shown in figure 20 a h bridge consists of a voltage source and four switches and a motor connected in between the pair of switches. According to the value or status of the switch the motor is driven in the required direction.

**Table 10 H BRIDGE SWITCH CHARESTERISTICS**

| <b>S1</b> | <b>S2</b> | <b>S3</b> | <b>S4</b> | <b>Motor</b>               |
|-----------|-----------|-----------|-----------|----------------------------|
| 0 (off)   | 0 (off)   | 0 (off)   | 0 (off)   | OFF                        |
| 0 (off)   | 1 (on)    | 1(on)     | 0(off)    | Moves in one direction     |
| 1(on)     | 0(off)    | 0(off)    | 1(on)     | Moves in another direction |
| 1(on)     | 1(on)     | 0(off)    | 0(off)    | Stop                       |
| 0(off)    | 0(off)    | 1(on)     | 1(on)     | Stop                       |



**Figure 22 SCHEMATIV REPRESENTATION OF H BRIDGE**

## 10.2. SIMULINK MODELING

### 10.2.1. SIMULINK MODELLING OF DC MOTOR.

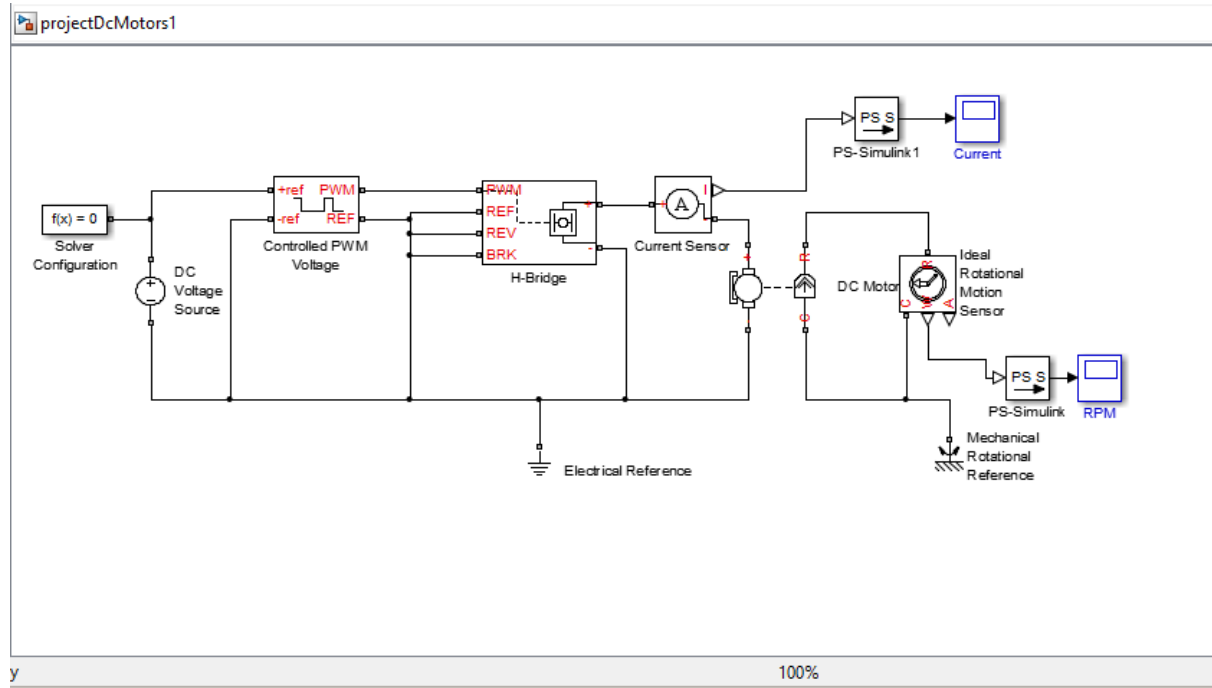
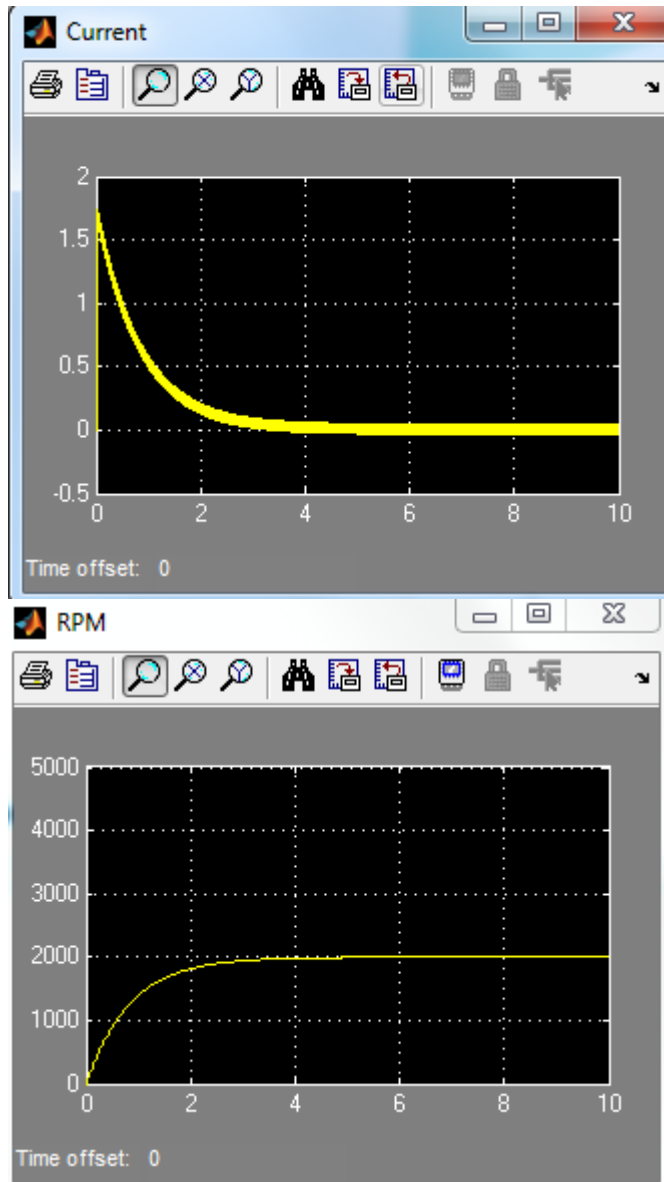


Figure 23 SIMULINK MODEL FOR DC MOTOR

From the above figure we can see that the voltage source is feed to a PWM generator which is sent to an H bridge. The PWM controls the H Bridge and One H Bridge connection is electrical grounded. And another connection is sent to a current sensor and the value is seen on the scope ant it is also connected to the electrical ports or sides of the DC motor. The mechanical port of the motor is connected to an ideal motion sensor which senses the rotational motion of the motor and it is viewed on a scope.



### 10.2.2. OUTPUT



**Figure 24 DC MOTOR I AND RPM PLOT SIMULINK**

From the output viewed on the scope we can see that when the motor starts it draws the max amount of current and slowly as the rpm reaches a steady state the current reduces. When the rpm is at a steady state the current also reaches steady state.

### 10.2.3. SIMULINK MODELING OF STEPPER MOTOR

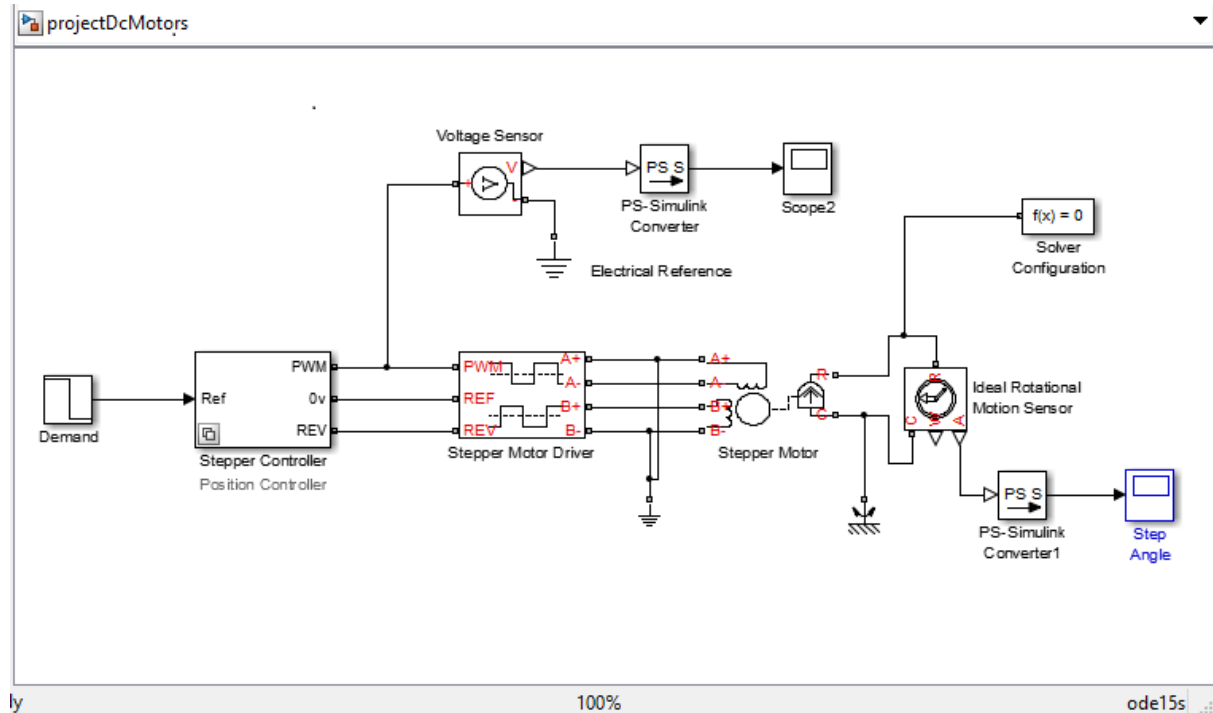


Figure 25 SIMULINK MODELING OF STEPPER MOTOR

Here the stepper controller is connected to a stepper motor drive which is used to drive the stepper motor. The stepper motor driver has four outputs which are A+, A-, B+, B- connected to the coils in the motor. The mechanical side is connected to an ideal motion sensor which senses the rotational motion of the motor and it is viewed on a scope.

## 11. System internal block diagram:

The entire robotic system they explained is split into various functional blocks. Each of these blocks has their own functionalities and properties and therefore it is good to give a block diagram representation for the simple analysis and understanding of the different blocks in the system. The block diagram of each of the blocks of the system is called an internal block diagram.

Here to denote the internal block diagram of the system we use SysML. As explained before, SysML is a modelling tool which helps us model various systems with ease. It helps in understanding and showing the flow in a system and the various functional units or blocks in that sub block or internal block system by representing it in the form of a block diagram.

The two important internal blocks involved in this system are given as follows.

- Motor control
- Sensor block

### 11.1. MOTOR CONTROL:

This diagram involves all the integral components required for motor control. The diagram shows as to what is needed at the controller end and what is needed on the robot. The internal block diagram of the motor control system of the robot is given below.

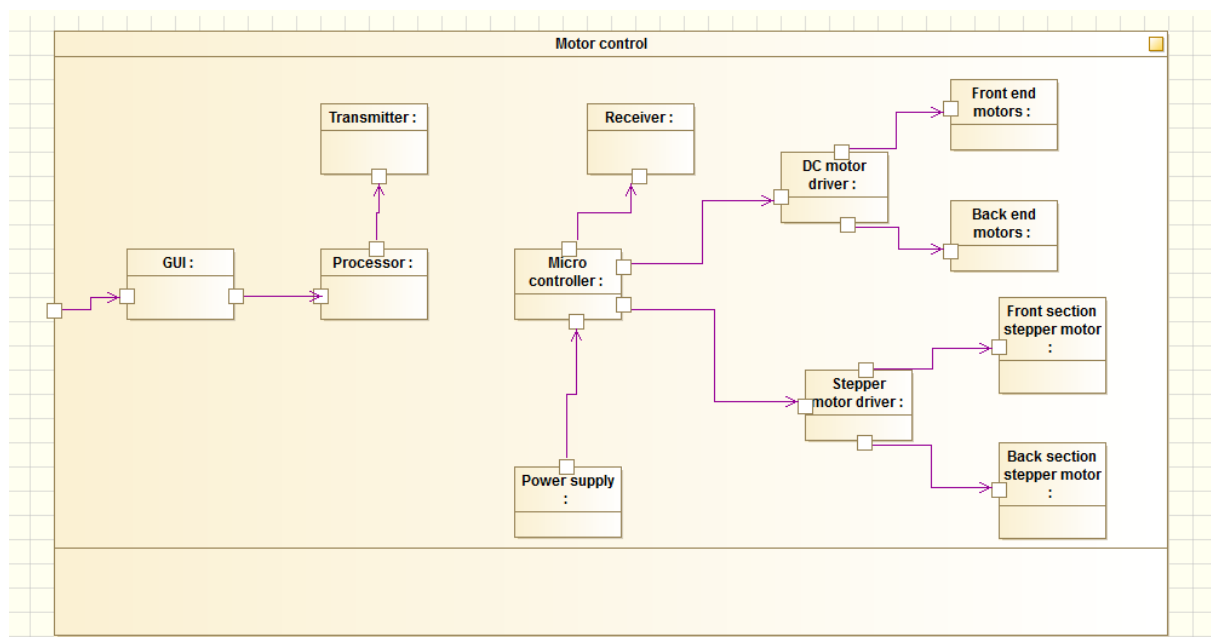


Figure 26 INTERNAL BLOCK DIAGRAM - MOTOR CONTROL

On the remote controller end of the system there is a microprocessor which is the heart of the system. A GUI (Graphic User Interface) is fitted in with the microprocessor which is the input side of the microprocessor. On the other side which is the output part of the remote controller system is attached a RF Tx which is used to transmit commands from the user to the robot.

On the robot we see that for the motion control of the system again at the heart of the system we have a microcontroller. The micro controller is fitted with a power supply which helps run the controller. The power supply required is minimum and mostly ranges below 5 -3V. A RF receiver is connected to the microcontroller which gives the input to the controller, which are the commands sent by the controller. Based on the commands sent the microcontroller either drives the DC motor drive circuit or the Stepper motor drive circuit. Each of these driver circuits are fitted with H-Bridges. These H-Bridges are switching devices which help in changing the direction of the flow of current. This in turn helps change the direction of the motor and either makes it rotate forward or backward.

In this system the DC motor circuits are used to drive the entire robot in the required direction and the stepper motor drive circuit is used to expand and contract the limbs of the robot which is used to grip itself against the walls of the pipe.

## 11.2. SENSOR BLOCK SYSTEM

This diagram shows all the integral components required for the sensor block. It covers both the parts which is data acquisition and data storage. As in motor control block diagram this too shows as to what is required on both ends. The internal block diagram is given as follow.

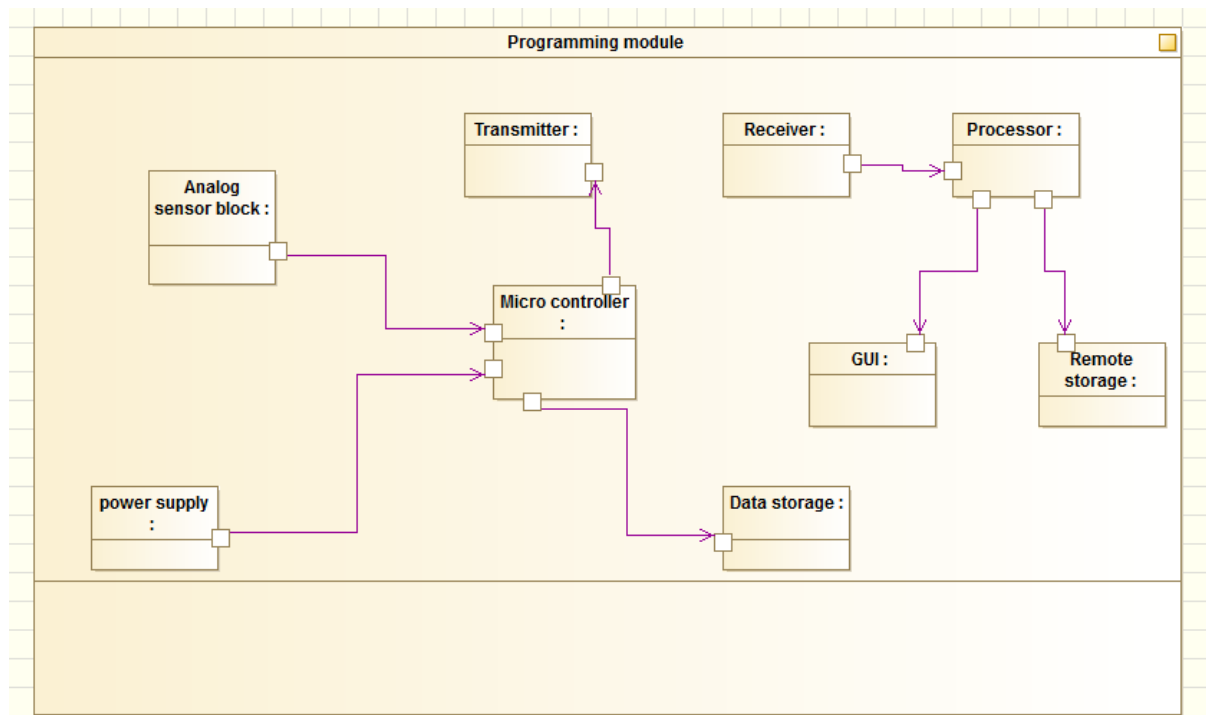


Figure 27 INTERNAL BLOCK DIAGRAM - SENSOR CONTROL

Here we see that the transmitter end of the system is the robot as it has to sense its environment and send the gained data to the operator.

On the robot we see that there is a microcontroller which is at the heart of the system. It receives its data through its digital input pins to which the required sensor is connected. The microcontroller on the robot is powered using a power supply to run it. The sensor data is collected or read by the sensors and then they are stored on to a micro sd card which is attached to the output pin of the system. The data stored here is used for data logging and for future analysis.

A RF Tx is also attached to the output pins of the microcontroller. This is used to transmit the data collect or read by the sensor to the remote terminal so that the user can view it and analyse the data.

On the remote terminal or the controller hub we can see that a microprocessor is at the heart of the system. At the input terminals of the microprocessor is connected a RF Rx which receives the data transmitted from the sensor block on the robot. The processor is connected to a GUI via serial communication in order to display the received data to the user. It can also be future analysed through the GUI and the data is stored on to another micro SD card which is connected to the microcontroller. The data which is received can be stored and plotted using different software's.

### **11.3. CONCLUSION:**

Thus from the above diagrams we can say as to what are the integral parts that comprise the motor control and sensor block and as to what are its functionalities. It's also helps us understand as to how the commands flow in the system.

## 12. Experimental results:

### 12.1. INTRODUCTION

This chapter described the working of the entire system as a whole. It will explain the working sequence flow of commands in the system and also describe the hardware and software information. In this chapter we shall see in detail about how the Arduino is connected to the various components their pin configuration etc.

In this chapter we shall discuss the construction and programming for the in-pipe inspection robot. It is concerned with the monitoring of the temperature and pressure within the pipes and at the end of the chapter we shall describe as to how to control the system by modelling the system in Simulink so that the system would maintain its self in stable state.

This system consists of two integral parts which is the control hub which is the remote operating unit which helps us to control the inspection robot and receive and view the information transmitted by it regarding the conditions within the pipe. The control hub consists of a transmitter and receiver module which transmits motor commands to the robot to drive it in the required direction and the other is used to receive sensor values from the inspection robot respectively. Likewise on the robot it is fitted with a transmitter and receiver module which transmits the sensor values which indicates the conditions within the pipe and the receiver receives the required commands from the operator to drive the motors respectively. Both the control hub and the in-pipe inspection are fitted with a micro SD card board in order to store the data attained by the sensors in the inspection robot. The inspection robot is also fitted with a motor drive circuit which consists of a microcontroller and a motor drive circuit which is an L298N dual H-Bridge circuit.

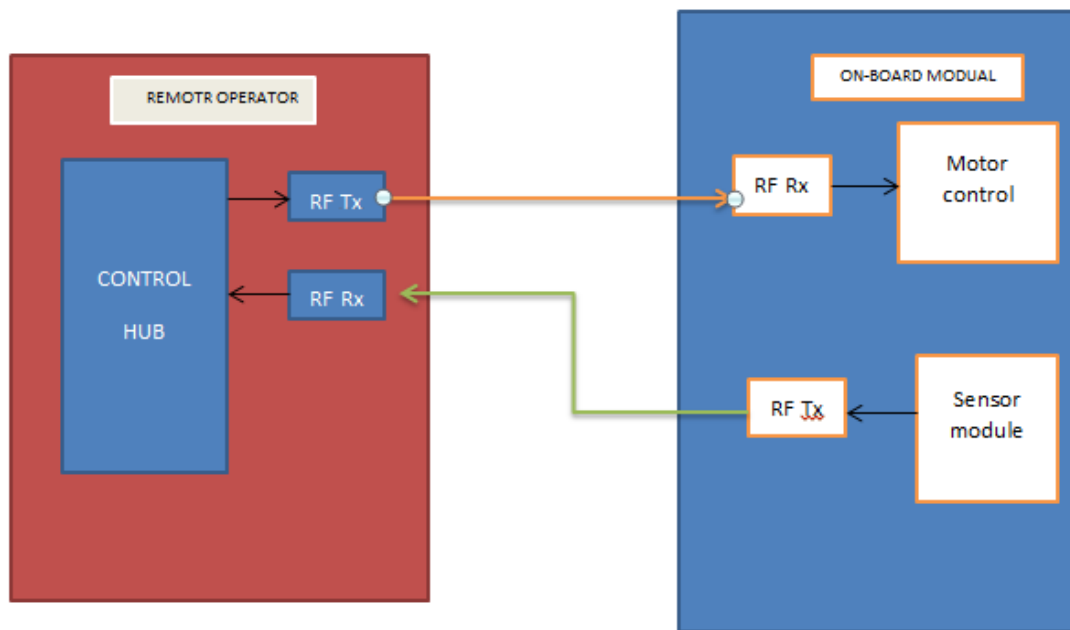


Figure 28 OVERALL SYSTEM BLOCK DIAGRAM

## 12.2. ARDUINO UNO SPECIFICATIONS [APENDIX 1]:

Table 11 ARDUINO UNI SPECIFICATIONS

|                             |                                    |
|-----------------------------|------------------------------------|
| Microcontroller             | ATmega328                          |
| Operating Voltage           | 5V                                 |
| Input Voltage (recommended) | 7-12V                              |
| Input Voltage (limits)      | 6-20V                              |
| Digital I/O Pins            | 14 (of which 6 provide PWM output) |
| Analog Input Pins           | 6                                  |
| DC Current per I/O Pin      | 40 mA                              |
| DC Current for 3.3V Pin     | 50 mA                              |
| Flash Memory                | 32 KB (ATmega328)                  |
| SRAM                        | 2 KB (ATmega328)                   |
| EEPROM                      | 1 KB (ATmega328)                   |
| Clock Speed                 | 16 MHz                             |



## 12.3. CONTROL HUB

### 12.3.1. CONTROL HUB PIN CONNECTIONS:

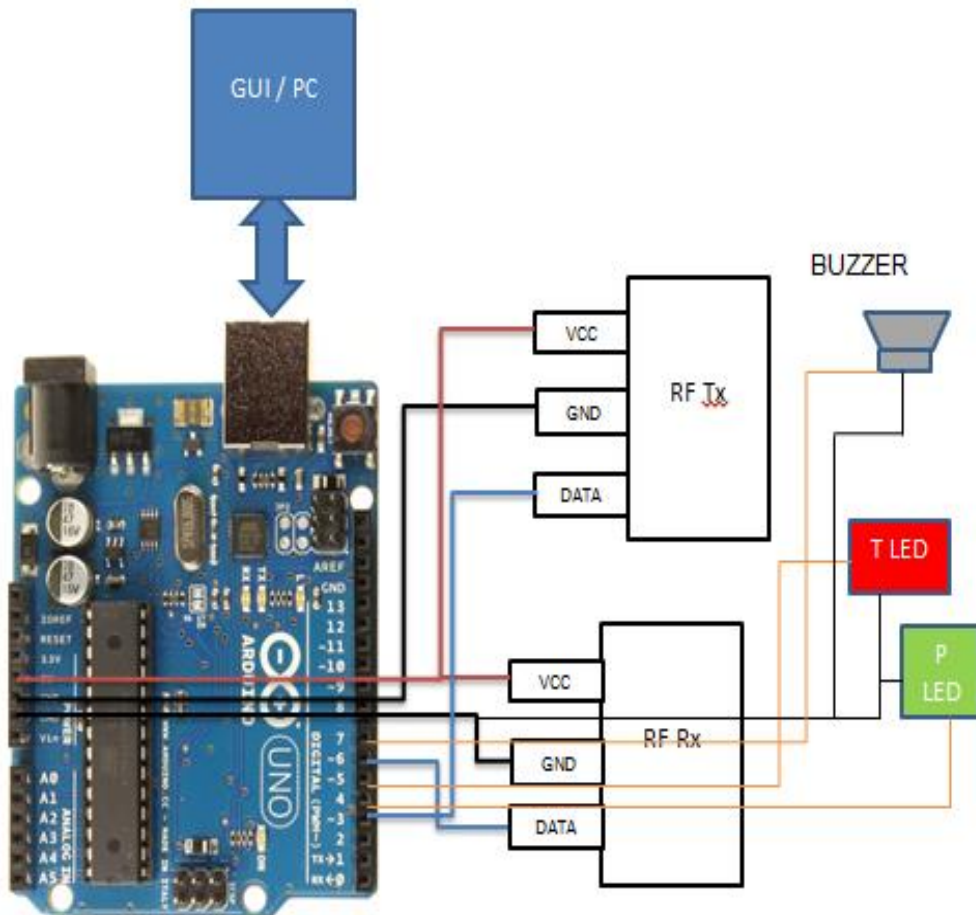


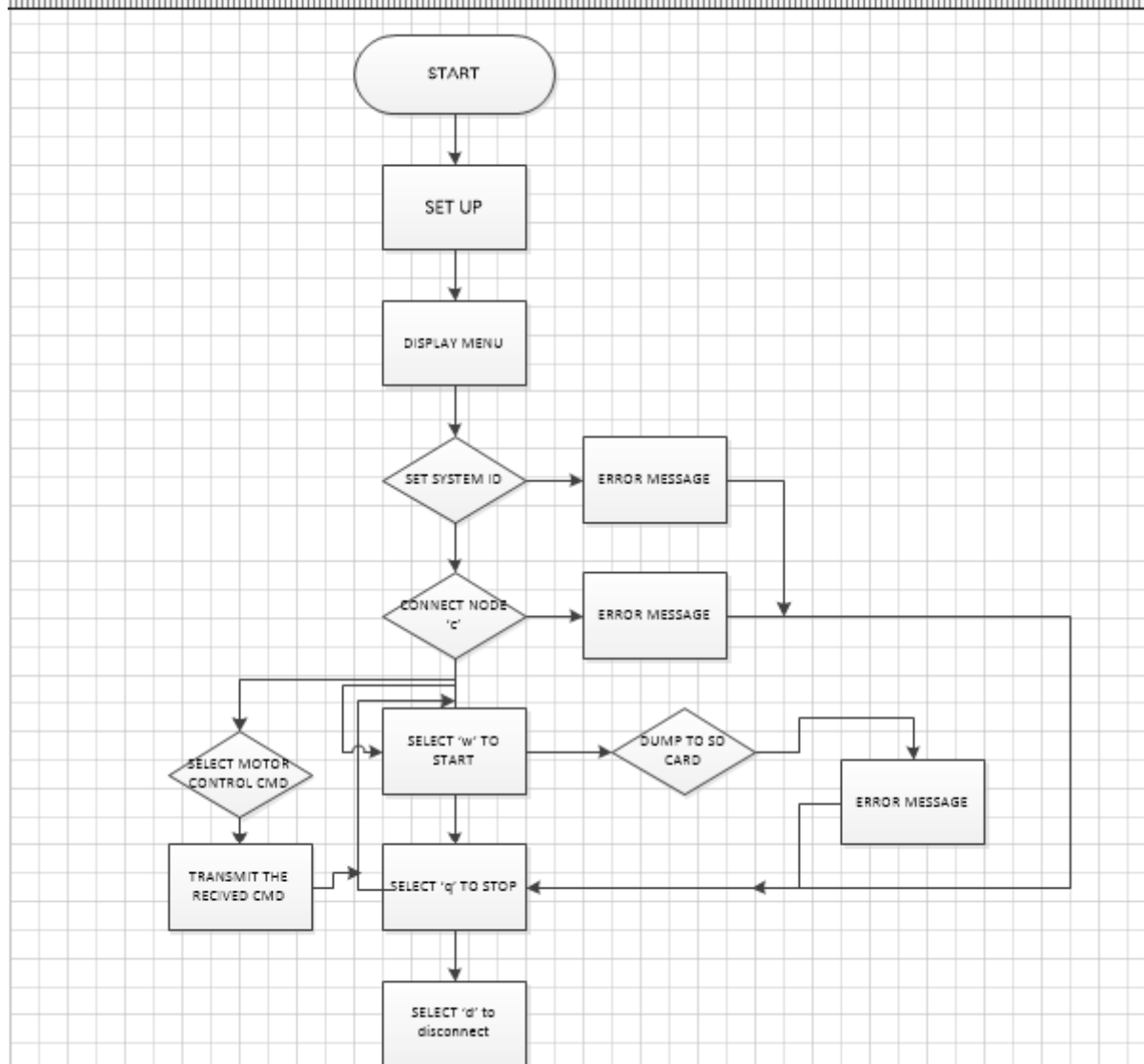
Figure 29 PIN CONNECTIONS FOR CONTROL HUB

Here the Arduino connections for the control hub are shown in a clear manner.

- Since there is only one +5V DC supply present on the Arduino UNO both the VCC of the RF Tx and the RF Rx is connected parallelly to it.
- The grounds of the RF TX /RX are both connected to the GND PINS (GROUND) present on the Arduino UNO.
- The DATA PIN of the RF Tx is connected to PIN 3 and the DATA PIN of the RF RX is connected to PIN 6 on the Arduino UNO.
- The LED's which are red and green are connected to PIN 5 and PIN 4 respectively on the Arduino UNO.
- These LED's are then connected to a common ground.
- The pin connections for the Buzzer are PIN 7 which is high and GND.

- The Arduino is then connected to the PC to display the GUI using the serial port which consists of a UART. The serial port is in sync with the PC at a baud rate of 9600.

### 12.3.2. SEQUENCE OF THE CODE



The process recommend to be followed is:

1. Start the program
2. Initialise the time using unix time.
3. Set system (hub) id.
4. Connect the inspection robot to be used by selecting <c><node id>. example: c2
5. Select 's' to start reading and displaying on the serial monitor.
6. To dump these values to a csv file select 'w'.

7. Select 'q' to quit from reading and displaying the values. When 'q' is selected it stops displaying the values and the menu is displayed again.
8. To control the motor control select quit from the sensor value viewing mode and select the required motor operation such as
  - Forward – f
  - Backward – b
  - Stop – x
  - Expand limbs – l
  - Contract limbs – r
9. The command is transmitted.
10. Start the sensor value viewing mode by selecting 's'.
11. Select 'q' to quit from reading and displaying the values. When 'q' is selected it stops displaying the values and the menu is displayed again.
12. Select 'd' to delete the current node from the data base.
13. To add a new node now all we have to do is to repeat from step 4.

### **Start:**

In the start we need to initialize all required variables and libraries.

```
#include <SD.h>
#include <Time.h>
#include <VirtualWire.h>
#include <stdio.h>
```

```
/*#define VCC 3
#define Data 4
#define GND 2*/
```

```
int chipSelect = 10;
int pow_pin = 8;
```

```
long id = 1; //Serial No. in the SD Card File
```

```
#define TIME_HEADER "T" // Header tag for serial
time sync message
#define TIME_REQUEST 7 // ASCII bell character
requests a time sync message
```

```
int menu_item = 0; // defining the integers data type
int newNode[15];
int system_id;
```

```
const int BUZZER = 7;
```

```
String room_names[15];
```

```
int temp_led = 5;
int p_led = 4;
```

```
//char *msg2 ;
```

- First the required libraries are initialised
- The pins for the RF receiver are defined next
- The header tag and ASCII bell character are defined in order to set unix time for time.h library.
- The required variables and string to be defined in the global scope is defined.

```
int readings [60][5] = {
  0, 0, 0, 0, 0,
  1, 0, 0, 0, 0,
  2, 0, 0, 0, 0,
  3, 0, 0, 0, 0,
  4, 0, 0, 0, 0,
  5, 0, 0, 0, 0,
  6, 0, 0, 0, 0,
  7, 0, 0, 0, 0,
  8, 0, 0, 0, 0,
  9, 0, 0, 0, 0,
  10, 0, 0, 0, 0,
  11, 0, 0, 0, 0,
  12, 0, 0, 0, 0,
  13, 0, 0, 0, 0,
  14, 0, 0, 0, 0,
  15, 0, 0, 0, 0,
  16, 0, 0, 0, 0,
  17, 0, 0, 0, 0,
  18, 0, 0, 0, 0,
  19, 0, 0, 0, 0,
  20, 0, 0, 0, 0,
  21, 0, 0, 0, 0,
  22, 0, 0, 0, 0,
  23, 0, 0, 0, 0};
```

### **Set time:**

This is used to initialise the device to unix time.

```
int setTime()
{
  if (Serial.available())
  {
    processSyncMessage();
  }
  if (timeStatus() != timeNotSet)
  {
    digitalClockDisplay();
  }

  delay(1000);
  return 1;
}
```

```
void digitalClockDisplay(){
  // digital clock display of the time
  Serial.print(hour());
  printDigits(minute());
  printDigits(second());
  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
}
```

```
void printDigits(int digits){
  // utility function for digital clock display: prints
  preceding colon and leading 0
  Serial.print(":");
  if(digits < 10)
```

- int setTime() which is declared in the scope. it is used to set the time. To set time we should enter in the serial monitor <T><UNIX TIME> example: T1401415800
- The first function declares the structure for a digital clock display for time.
- The second function prints the time in correct format.
- The next function is used to check if the input value is a valid time and.
- It sync the Arduino to the time received.

```

    Serial.print('0');
    Serial.print(digits);
}

void processSyncMessage() {
    unsigned long pctime;
    const unsigned long DEFAULT_TIME =
    1357041600; // Jan 1 2013
    int daylight = 3600;

    if(Serial.find(TIME_HEADER)) {
        pctime = Serial.parseInt();
        pctime += daylight;
        if( pctime >= DEFAULT_TIME) { // check the integer
        is a valid time (greater than Jan 1 2013)
            setTime(pctime); // Sync Arduino clock to the time
            received on the serial port
        }
    }
}

time_t requestSync()
{
    Serial.write(TIME_REQUEST);
    return 0; // the time will be sent later in response to
    serial mesg
}

```

### **Display menu:**

```

void displaymenu()
{
    //digitalClockDisplay();
    Serial.println(F("In-Pipe Inspection Robot"));
    Serial.println(F("Menu options"));
    Serial.println(F("-----"));
    Serial.println(F("T - set time"));
    Serial.println(F("i - set system id"));
    Serial.println(F("c - Connect to robot"));
    Serial.println(F("d - Disconnect"));
    Serial.println(F("w - dump to CSV"));
    Serial.println(F("-----"));
    Serial.println(F("motor control"));
    Serial.println(F("-----"));
    Serial.println(F("l - expand limbs"));
    Serial.println(F("r - contract limbs"));
    Serial.println(F("f - forward"));
    Serial.println(F("b - backward"));
    Serial.println(F("x - stop"));
    Serial.println(F("-----"));
    Serial.println(F("s - start"));
    Serial.println(F("q - quit"));
    Serial.println(F("-----"));
    return;
}

```

- This menu is used to print the menu on the screen.
- displaymenu() function is called it display the menu.

```

if (Serial.available() > 0)
{
    int sendByte = Serial.read();//Starts reading data
    from the serial monitor once condition is met

    switch (sendByte)
    {

```

- This shows the case statement used for selection in the menu.
- The case statement is present in the loop whereas each option functions

```

case 'f': //if the controller type f
{
    Serial.println(F("forward"));
    char *msg2 = "f";
    digitalWrite(13, true); // Flash a light to show
transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send
byte to the receiver
    vw_wait_tx(); // Wait until the whole message is
gone
    digitalWrite(13, false);
    delay(500);
    break;
}

case 'b': //if controller types b
{
    Serial.println(F("backward"));
    char *msg2 = "b";
    digitalWrite(13, true); // Flash a light to show
transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send
byte to the receiver
    vw_wait_tx(); // Wait until the whole message is
gone
    digitalWrite(13, false);
    delay(500);
    break;
}

case 'x': //if controller types s
{
    Serial.println(F("stop"));
    char *msg2 = "x";
    digitalWrite(13, true); // Flash a light to show
transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send
byte to the receiver
    vw_wait_tx(); // Wait until the whole message is
gone
    digitalWrite(13, false);
    delay(500);
    break;
}

case 'l': //if controller types l
{
    char *msg2 = "l";
    digitalWrite(13, true); // Flash a light to show
transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send
byte to the receiver
    vw_wait_tx(); // Wait until the whole message is
gone
    digitalWrite(13, false);
    break;
}

case 'r': //if controller types r
{
    char *msg2 = "r";
    digitalWrite(13, true); // Flash a light to show
transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send
byte to the receiver
    vw_wait_tx(); // Wait until the whole message is
gone

```

is present outside the loop in the scope.

- At first an if statement checks if any values have been entered.
- It then cross-references the entered value with the values of each case by using serial read and obtaining the ASCII value and checking them with the option for each case.
- When the required case is identified it enters that case and calls the function in it and processes only that case and returns when it is done.

```

        digitalWrite(13, false);
        break;
    }
    case 'T':        // if user enters 'T' then print the unix
time
    {
        settime();
        break;
    }
    case 'i':
    {
        setSystemID();
        break;
    }
    case 'c':
    {
        Connect();
        break;
    }
    case 'd':
    {
        Disconnect(); // if user enters 'x' then erase
database
        break;
    }
    case 'w':
    {
        dumpSD();
        break;
    }
    case 's':
    {
        Serial.println(F("string scan"));
        msgscan();
        Serial.println(F("stopped *****string scan"));
        displaymenu();
        break;
    }
    default://if any other value is entered
    {

        Serial.println(F("wrong entry"));//print wrong entry
in the serial monitor
    }
} // close switch-case
} //close if
} //close loop

```

### **Select system ID:**

This is used to define the hub id.

```

int setSystemID()
{
    system_id= 15;
    Serial.println(F("setSystemID"));
    Serial.println(system_id);
    return 1;
}

```

- This function is found in the scope it is called from the case statement in the loop by selecting 's'.
- The system Id is given as 15 and when selected it is displayed on the

|  |         |
|--|---------|
|  | screen. |
|--|---------|

### **CONNECT NODE:**

Here both the node id and the node name have been declared and done in a single function by selecting <a><node id><node name> example: a2bedroom

|   |  |
|---|--|
| <pre> int Connect() {   Serial.println(F("Connecting....."));   delay(500);   boolean gotit = false;   int option;   while (!gotit)   {     if (Serial.available() &gt; 0)     {       // read the incoming byte:       option = Serial.parseInt();       Serial.println(F(" "));       Serial.print(F("Node= "));       Serial.println(option);       delay(100);       if (option &gt; 0 &amp;&amp; option &lt; 16)       {         gotit = true;         newNode[option] = 1;         //delay(500);        }       Serial.print(F("Connected to node "));       delay(500);      }     else     {       Serial.print(option);       Serial.print(F(" ** out of range - enter a number between 1 and 15 **"));     }   } } </pre> | <ul style="list-style-type: none"> <li>• It first checks if it has got a value and then it reads the incoming byte.</li> <li>• It checks if the input value is from 1 to 15. If so it adds the node and if not it gives an error message that it is out of range.</li> <li>• To add node name it is entered along with the node id.</li> <li>• The node name is first stored as an array which is of type string.</li> <li>• Each character in the node name takes a place in the array.</li> <li>• It is then added to the node id and corresponds to it and is displayed on the screen.</li> </ul> |
|---|--|

### **Select 's' to start and q to quit:**

To start printing the values we enter 's' which is one of the cases in the loop. This intern calls a function which is declared in the scope outside the loop.

|   |  |
|---|--|
| <pre> case 's': {   Serial.println(F("string scan"));   msgscan();   Serial.println(F("stopped *****string scan"));   displaymenu(); } </pre> | <ul style="list-style-type: none"> <li>• This is the case for starting to print the values on the screen.</li> <li>• When it is selected it calls the</li> </ul> |
|---|--|



|   |   |
|---|---|
| <pre>break; }</pre>   | <p>function msgscan().</p>  |
| <pre>void msgscan() {   //Serial.read();   uint8_t buflen = VW_MAX_MESSAGE_LEN;   uint8_t buf[buflen];    int exitnow = 0;    while(!exitnow)   {     if (Serial.available() &gt; 0) // reading the data from the     serial portn which is already stored in the serial     reciever buffer     {       if(Serial.read() == 'q')         exitnow = 1;     }      if (vw_get_message(buf, &amp;buflen)) // check to see if     anything has been received     {       int i;       // Message with a good checksum received.       for (i = 0; i &lt; buflen; i++)       {         //Serial.println(i);         Serial.println(buf[i]);         // char* message = (char*)buf;         // int value = atoi(message);         // Serial.println(value);         // the received data is stored in buffer       }       // Serial.println("");     }     // Serial.println(buf[0]);      int temp = buf[0];      int nid = temp - 240;     int hid = temp - nid;      int err = 0 ;     if(hid == 240)      {        if( newNode[nid] == 1)       {         Serial.println(F("-----         -----"));         Serial.println(F("-----         -----"));          Serial.print(F("System ID: \t"));         Serial.println(hid,DEC); // printing the value of the         system ID in decimal          Serial.print(F("NODE ID: "));         Serial.println(nid,DEC); // printing the value of         Node ID in decimal          Serial.print(F("Temperature = "));</pre> | <ul style="list-style-type: none"> <li>• This function first reads the incoming buffer messages in the receiver section.</li> <li>• It checks to see if anything is received.</li> <li>• It checks the buffer length and the values received from the buffer.</li> <li>• It then stores the received data in another buffer.</li> <li>• It splits the higher buffer values and lower buffer values as hid and nid respectively to display the system id and node id.</li> <li>• It then prints out all the values received from the buffer on to the screen.</li> <li>• If it is receiving data the LEDs would turn on.</li> <li>• If it is not receiving any data it would be off.</li> <li>• If the temperature or pressure value is above or below critical value, the LEDS would blink and the buzzer would go off and display a warning message on the screen indicating the system is no longer in stable state.</li> <li>• If the system does not recognise the node id or system id of if there is an error it would print not recognized system id.</li> <li>• To stop displaying the values on the screen and to exit this function we are required to select 'q'.</li> </ul> |

```

Serial.print(buf[2],DEC); // printing the value of
humidity in decimal
Serial.println(" *C");
//Serial.print("\t");
Serial.println(F(""));
if(buf[2]>25)
{
  makenoise();
  Serial.println(F("System temperature is
Exceeded Critical Limit"));
  Serial.println(F(""));
  digitalWrite(temp_led,HIGH);
  delay(1000);
  digitalWrite(temp_led,LOW);
  delay(500);
}
else if(buf[2]<=20)
{
  makenoise();
  Serial.println(F("System temperature is fallen
below Minimum Limit"));
  Serial.println(F(""));
  digitalWrite(temp_led,HIGH);
  delay(1000);
  digitalWrite(temp_led,LOW);
  delay(500);
}
else
{
  Serial.println(F("System temperature is in
control"));
  Serial.println(F(""));
  digitalWrite(temp_led,HIGH);
}

Serial.print(F("Pressure = "));
Serial.print(buf[4],DEC); // printing the value of
temperature in decimal
Serial.println(" Psi");
Serial.println(F(""));
if(buf[4]>20)
{
  makenoise();
  Serial.println(F("System preassure is Exceeded
Critical Limit"));
  Serial.println(F(""));
  digitalWrite(p_led,HIGH);
  delay(1000);
  digitalWrite(p_led,LOW);
  delay(500);
}
else if(buf[4]<=12)
{
  makenoise();
  Serial.println(F("System preassure is fallen
below Minimum Limit"));
  Serial.println(F(""));
  digitalWrite(p_led,HIGH);
  delay(1000);
  digitalWrite(p_led,LOW);
  delay(500);
}
else
{
  Serial.println(F("System preassure is in
control"));
  Serial.println(F(""));
}

```

- This would exit from the function and display the menu.

```

        digitalWrite(p_led,HIGH);
    }

    Serial.println(F("-----"));
    Serial.println(F("-----"));

    //update the array with details read
    readings[nid][0] = nid;    //node id
    readings[nid][1] = buf[2]; //temp
    readings[nid][2] = buf[4]; //pressure

    delay(1000);

}
else
{
    err = 1;
}

}

else
{
    err = 1;
}
if(err)
{
    Serial.println("not recognized System ID: ");
}
}
}
}

```

### **Dump to SD:**

This is used to log all the data received to a csv file. The option is selected by selecting 'c' in the case statement of the menu which in turn executes the dumpCSV() function.

```

int dumpSD()
{
    Serial.println(F("Initializing SD Card Data Logger"));

    pinMode(chipSelect, OUTPUT);

    pinMode(pow_pin,OUTPUT);
    digitalWrite(pow_pin, HIGH);

    //Checking if card is ready

    if(!SD.begin(chipSelect))
    {
        Serial.println(F("Card Initialization Failed"));
    }
    // return;
}

```

- To dump to CSV we first store the data in a SD card.
- If there is no card present an error message will appear.
- The data is saved into a text file using string format.
- Each column is stored as an array in the format of a string.

```

}
else
{
  Serial.println(F("Card INITIALIZED"));
}

File logFile = SD.open("logger2.txt", FILE_WRITE);
if(logFile)
{
  // logFile.println(", , , , , , , , , ,");
  // String header = "ID, Temp, Pa, acc_x, acc_y,
  acc_z, avx, avy, avz, Pitch, Roll, mx, my, mz,
  heading";
  logFile.println(", ,");
  String header = "ID, Temp, Pa";
  logFile.println(header);
  logFile.close();
  Serial.println(header);
}
else
{
  Serial.println(F("Couldn't Open File in setup"));
}
for(int ll = 0 ; ll<23 ;ll++)
{

  // String dataString = String(id) + "," + String(temp) +
  "," + String(Pa) + "," + String(acc_x) + "," +
  String(acc_y) + "," + String(acc_z) + "," + String(avx) +
  "," + String(avy) + "," + String(avz) + "," + String(pitch)
  + "," + String(roll) + "," + String(mx) + "," + String(my)
  + "," + String(mz) + "," + String(head);
  String dataString = String(id) + "," +
  String(readings[ll][1]) + "," + String(readings[ll][2]);

  //Open a file to write to
  //Only one file can be opened at a time
  File logFile = SD.open("logger2.txt", FILE_WRITE);
  if(logFile)
  {
    logFile.println(dataString);
    logFile.close();
    Serial.println(dataString);
  }
  else
  {
    Serial.println(F("Couldn't access file"));
  }

  id++; //Increment id file

  delay(1000);
}
}

```

- The file name is denoted in the program if the file is available it opens it and writes the data to it.
- If the file isn't open an error pops up.
- Once it is done copying it exits the function and displays the menu.

### **Disconnect node:**

When a node is to be deleted we select x from the case statement. This in turn executes the deleteNode() function.

```

int Disconnect()
{
  boolean gotit = false;

```

- It first checks if it has got a value and then it reads the incoming byte.

```

int option;
while (!gotit)
{
  if (Serial.available() > 0)
  {
    // read the incoming byte:
    option = Serial.parseInt();
    Serial.println(F(" "));
    if (option > 0 && option < 16)
    {
      gotit = true;
      newNode[option] = 0;
      Serial.print(F("Disconnect from node"));
      //delay(1000);

    }
    else
    {
      Serial.print(option);
      Serial.print(F(" ** out of range - enter a number
between 1 and 15 **"));
    }
  }
}
return option;
}

```

- It checks if the input value is from 1 to 15.
- If so it deletes the node and prints node deleted.
- If not it gives an error message that it is out of range.

### 12.3.3. PROGRAM:

```
#include <SD.h>
#include <Time.h>
#include <VirtualWire.h>
#include <stdio.h>

int chipSelect = 10;
int pow_pin = 8;

long id = 1; //Serial No. in the SD Card File

#define TIME_HEADER "T" // Header tag for serial time sync message
#define TIME_REQUEST 7 // ASCII bell character requests a time sync message

int menu_item = 0; // defining the integers data type
int newNode[15];
int system_id;

const int BUZZER = 7;

String room_names[15];

int temp_led = 5;
int p_led = 4;

int readings [60][5] = {
    0, 0, 0, 0,0,
    1, 0, 0, 0,0,
    2, 0, 0, 0,0,
    3, 0, 0, 0,0,
```

```
4, 0, 0, 0,0,  
5, 0, 0, 0,0,  
6, 0, 0, 0,0,  
7, 0, 0, 0,0,  
8, 0, 0, 0,0,  
9, 0, 0, 0,0,  
10, 0, 0, 0,0,  
11, 0, 0, 0,0,  
12, 0, 0, 0,0,  
13, 0, 0, 0,0,  
14, 0, 0, 0,0,  
15, 0, 0, 0,0,  
16, 0, 0, 0,0,  
17, 0, 0, 0,0,  
18, 0, 0, 0,0,  
19, 0, 0, 0,0,  
20, 0, 0, 0,0,  
21, 0, 0, 0,0,  
22, 0, 0, 0,0,  
23, 0, 0, 0,0 };
```

```
void setup()  
{  
  Serial.begin(9600);  // Debugging only  
  Serial.println(F("setup"));  
  
  setSyncProvider(requestSync); //set function to call when sync required  
  Serial.println(F("Waiting for sync message"));  
  delay(500);  
  settime();  
  delay(500);
```

```

vw_setup(2000);    // Bits per sec
vw_set_tx_pin(3); //Transmitter Data Pin to Digital Pin 3


vw_setup(2000);
vw_set_rx_pin(6);
vw_rx_start();


pinMode(temp_led,OUTPUT);
pinMode(p_led,OUTPUT);


Serial.println(F("-----"));


do
{
    displaymenu();
    menu_item = 0;
}
while (menu_item > 1);


} //close setup


void loop()
{

    if (Serial.available() > 0)
    {
        int sendByte = Serial.read();//Starts reading data from the serial monitor once condition
        is met
    }
}

```



```

switch (sendByte)
{

case 'f': //if the controller type f
{
    Serial.println(F("forward"));
    char *msg2 = "f";
    digitalWrite(13, true); // Flash a light to show transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send byte to the receiver
    vw_wait_tx(); // Wait until the whole message is gone
    digitalWrite(13, false);
    delay(500);
    break;

}

case 'b': //if controller types b
{
    Serial.println(F("backward"));
    char *msg2 = "b";
    digitalWrite(13, true); // Flash a light to show transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); //send byte to the receiver
    vw_wait_tx(); // Wait until the whole message is gone
    digitalWrite(13, false);
    delay(500);
    break;

}

case 'x': //if controller types s
{
    Serial.println(F("stop"));
    char *msg2 = "x";

```

```

digitalWrite(13, true); // Flash a light to show transmitting
vw_send((uint8_t *)msg2, strlen(msg2)); // send byte to the receiver
vw_wait_tx(); // Wait until the whole message is gone
digitalWrite(13, false);
delay(500);
break;
}

case 'l': // if controller types l
{
    char *msg2 = "l";
    digitalWrite(13, true); // Flash a light to show transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); // send byte to the receiver
    vw_wait_tx(); // Wait until the whole message is gone
    digitalWrite(13, false);
    break;
}

case 'r': // if controller types r
{
    char *msg2 = "r";
    digitalWrite(13, true); // Flash a light to show transmitting
    vw_send((uint8_t *)msg2, strlen(msg2)); // send byte to the receiver
    vw_wait_tx(); // Wait until the whole message is gone
    digitalWrite(13, false);
    break;
}

case 'T': // if user enters 'T' then print the unix time
{
    settime();
    break;
}

```

```
case 'i':
{
    setSystemID();
    break;
}
case 'c':
{
    Connect();
    break;
}
case 'd':
{
    Disconnect(); // if user enters 'x' then erase database
    break;
}
case 'w':
{
    dumpSD();
    break;

}
case 's':
{
    Serial.println(F("string scan"));
    msgscan();
    Serial.println(F("stopped *****string scan"));
    displaymenu();
    break;
}
default://if any other value is entered
{
```

```

        Serial.println(F("wrong entry")); //print wrong entry in the serial monitor
    }
    } // close switch-case
} //close if
} //close loop

void displaymenu()
{
    //digitalClockDisplay();
    Serial.println(F("In-Pipe Inspection Robot"));
    Serial.println(F("Menu options"));
    Serial.println(F("-----"));
    Serial.println(F("T - set time"));
    Serial.println(F("i - set system id"));
    Serial.println(F("c - Connect to robot"));
    Serial.println(F("d - Disconnect"));
    Serial.println(F("w - dump to CSV"));
    Serial.println(F("-----"));
    Serial.println(F("motor control"));
    Serial.println(F("-----"));
    Serial.println(F("l - expand limbs"));
    Serial.println(F("r - contract limbs"));
    Serial.println(F("f - forward"));
    Serial.println(F("b - backward"));
    Serial.println(F("x - stop"));
    Serial.println(F("-----"));
    Serial.println(F("s - start"));
    Serial.println(F("q - quit"));
    Serial.println(F("-----"));
    return;
}

```

```

}

int settime()
{
    if (Serial.available())
    {
        processSyncMessage();
    }
    if (timeStatus() != timeNotSet)
    {
        digitalClockDisplay();
    }

    delay(1000);
    return 1;
}

void digitalClockDisplay(){
    // digital clock display of the time
    Serial.print(hour());
    printDigits(minute());
    printDigits(second());
    Serial.print(" ");
    Serial.print(day());
    Serial.print(" ");
    Serial.print(month());
    Serial.print(" ");
    Serial.print(year());
    Serial.println();
}

```

```

void printDigits(int digits){
    // utility function for digital clock display: prints preceding colon and leading 0
    Serial.print(":");
    if(digits < 10)
        Serial.print('0');
    Serial.print(digits);
}

void processSyncMessage() {
    unsigned long pctime;
    const unsigned long DEFAULT_TIME = 1357041600; // Jan 1 2013
    int daylight = 3600;

    if(Serial.find(TIME_HEADER)) {
        pctime = Serial.parseInt();
        pctime += daylight;
        if( pctime >= DEFAULT_TIME) { // check the integer is a valid time (greater than Jan 1
            2013)
            setTime(pctime); // Sync Arduino clock to the time received on the serial port
        }
    }
}

time_t requestSync()
{
    Serial.write(TIME_REQUEST);
    return 0; // the time will be sent later in response to serial mesg
}

```

```
int setSystemID()
{
    system_id= 15;
    Serial.println(F("setSystemID"));
    Serial.println(system_id);
    return 1;
}
```

```
int Connect()
{
    Serial.println(F("Connecting....."));
    delay(500);
    boolean gotit = false;
    int option;
    while (!gotit)
    {
        if (Serial.available() > 0)
        {
            // read the incoming byte:
            option = Serial.parseInt();
            Serial.println(F(" "));
            Serial.print(F("Node= "));
            Serial.println(option);
            delay(100);
            if (option > 0 && option < 16)
            {
                gotit = true;
                newNode[option] = 1;
                //delay(500);
            }
        }
    }
}
```

```

    }
    Serial.print(F("Connected to node "));
    delay(500);

}
else
{
    Serial.print(option);
    Serial.print(F("  ** out of range - enter a number between 1 and 15 **"));
}
}
}

int Disconnect()
{

    boolean gotit = false;
    int option;
    while (!gotit)
    {
        if (Serial.available() > 0)
        {
            // read the incoming byte:
            option = Serial.parseInt();
            Serial.println(F(" "));
            if (option > 0 && option < 16)
            {
                gotit = true;
                newNode[option] = 0;
                Serial.print(F("Disconnect from node"));
            }
        }
    }
}

```



```

        //delay(1000);

    }
    else
    {
        Serial.print(option);
        Serial.print(F("  ** out of range - enter a number between 1 and 15 **"));
    }
}
}
return option;
}

int dumpSD()
{
    Serial.println(F("Initializing SD Card Data Logger"));

    pinMode(chipSelect, OUTPUT);

    pinMode(pow_pin, OUTPUT);
    digitalWrite(pow_pin, HIGH);

    //Checking if card is ready

    if(!SD.begin(chipSelect))
    {
        Serial.println(F("Card Initialization Failed"));
    }
    else
    {
        Serial.println(F("Card INITIALIZED"));
    }
}

```

```

}

File logFile = SD.open("logger2.txt", FILE_WRITE);
if(logFile)
{
    logFile.println(" , ");
    String header = "ID, Temp, Pa";
    logFile.println(header);
    logFile.close();
    Serial.println(header);
}
else
{
    Serial.println(F("Couldn't Open File in setup"));
}

for(int ll = 0 ; ll<23 ;ll++)
{
    String dataString = String(id) + "," + String(readings[ll][1]) + "," + String(readings[ll][2]);

    //Open a file to write to
    //Only one file can be opened at a time
    File logFile = SD.open("logger2.txt", FILE_WRITE);
    if(logFile)
    {
        logFile.println(dataString);
        logFile.close();
        Serial.println(dataString);
    }
    else
    {
        Serial.println(F("Couldn't access file"));
    }
}

```

```

}

id++;    //Increment id file

delay(1000);
}
}
void msgscan()
{
    //Serial.read();
    uint8_t buflen = VW_MAX_MESSAGE_LEN;
    uint8_t buf[buflen];

    int exitnow = 0;

    while(!exitnow)
    {
        if (Serial.available() > 0) // reading the data from the serial portn which is already stored
        in the serial reciever buffer
        {
            if(Serial.read() == 'q')
                exitnow = 1;

        }

        if (vw_get_message(buf, &buflen)) // check to see if anything has been received
        {
            int i;

            // Message with a good checksum received.
            for (i = 0; i < buflen; i++)
            {

```

```

    //Serial.println(i);
    Serial.println(buf[i]);
    //    char* message = (char*)buf;
    //    int value = atoi(message);
    //    Serial.println(value);
    // the received data is stored in buffer
}
//    Serial.println("");
}
//    Serial.println(buf[0]);

int temp = buf[0];

int nid = temp - 240;
int hid = temp - nid;

int err = 0 ;
if(hid == 240)

{

    if( newNode[nid] == 1)
    {
        Serial.println(F("-----
        -----"));
        Serial.println(F("-----
        -----"));

        Serial.print(F("System ID: \t"));
        Serial.println(hid,DEC); // printing the value of the system ID in decimal
    }
}

```

```

Serial.print(F("NODE ID: "));
Serial.println(nid,DEC); // printing the value of Node ID in decimal


Serial.print(F("Temperature = "));
Serial.print(buf[2],DEC); // printing the value of humidity in decimal
Serial.println(" *C");
//Serial.print("\t");
Serial.println(F(""));
if(buf[2]>25)
{
    makenoise();
    Serial.println(F("System temperature is Exceeded Critical Limit"));
    Serial.println(F(""));
    digitalWrite(temp_led,HIGH);
    delay(1000);
    digitalWrite(temp_led,LOW);
    delay(500);
}
else if(buf[2]<=20)
{
    makenoise();
    Serial.println(F("System temperature is fallen below Minimum Limit"));
    Serial.println(F(""));
    digitalWrite(temp_led,HIGH);
    delay(1000);
    digitalWrite(temp_led,LOW);
    delay(500);
}
else
{
    Serial.println(F("System temperature is in control"));

```

```

Serial.println(F(""));
digitalWrite(temp_led,HIGH);
}

Serial.print(F("Pressure = "));
Serial.print(buf[4],DEC); // printing the value of temperature in decimal
Serial.println(" Psi");
Serial.println(F(""));
if(buf[4]>20)
{
    makenoise();
    Serial.println(F("System preassure is Exceeded Critical Limit"));
    Serial.println(F(""));
    digitalWrite(p_led,HIGH);
    delay(1000);
    digitalWrite(p_led,LOW);
    delay(500);
}
else if(buf[4]<=12)
{
    makenoise();
    Serial.println(F("System preassure is fallen below Minimum Limit"));
    Serial.println(F(""));
    digitalWrite(p_led,HIGH);
    delay(1000);
    digitalWrite(p_led,LOW);
    delay(500);
}
else
{
    Serial.println(F("System preassure is in control"));

```

```

        Serial.println(F(""));
        digitalWrite(p_led,HIGH);
    }

    Serial.println(F("-----
-----"));

    Serial.println(F("-----
-----"));

    //update the array with details read
    readings[nid][0] = nid;    //node id
    readings[nid][1] = buf[2];    //temp
    readings[nid][2] = buf[4];    //pressure

    delay(1000);

}
else
{
    err = 1;
}
}
else
{
    err = 1;
}
if(err)
{
    Serial.println("not recognized System ID: ");
}
}
}
}

```

```

void makenoise(void)

{    // This function is basically made because we are asked to make the buzzer sound
three short beeps

    // These three different noises will tell us that the reaction timer has ended

tone(BUZZER,2000,400); // tone function is used on 1500 frequency and 400 milliseconds

delay(300);          // delay function is used to cause delay for 200 milliseconds

tone(BUZZER,2000,400);

delay(300);          // delay function is used to cause delay for 200 milliseconds

tone(BUZZER,2000,400);

}

```

#### 12.3.4. OUTPUT

When we on the remote controller we get the menu which is shown in figure 28.

```

setup
Waiting for sync message
-----
0:00:02 1 1 1970
In-Pipe Inspection Robot
Menu options
-----
T - set time
i - set system id
c - Connect to robot
d - Disconnect
w - dump to CSV
h - dump status to html
-----
motor control
-----
l - expand limbs
r - contract limbs
f - forward
b - backward
x - stop
-----
s - start
q - quit
-----

```

**Figure 30 REMOTE CONTROLLER MENU**

After which we select the note to connect with and connect to it it would display this on the screen as shown in figure 29.



```
Connecting.....  
  
Node= 3  
Connected to node string scan
```

**Figure 31 OUTPUT SHOWING IT'S CONNECTED**

Now it starts receiving data from the sensor hub and it is shown on the screen. The output is shown in figure 30.

```
-----  
-----  
System ID:      240  
NODE ID: 3  
Temperature = 23 *C  
  
System temperature is in control  
  
Pressure = 14 Psi  
  
System preassure is in control  
  
-----  
-----
```

**Figure 32 RECEIVER MODULE ON THE CONTROL HUB**

## 12.4. ON BOARD MODULE

As explained the on board module has two parts

- Motor control
- Sensor block

### 12.4.1. MOTOR CONTROL PIN CONFIGURATION

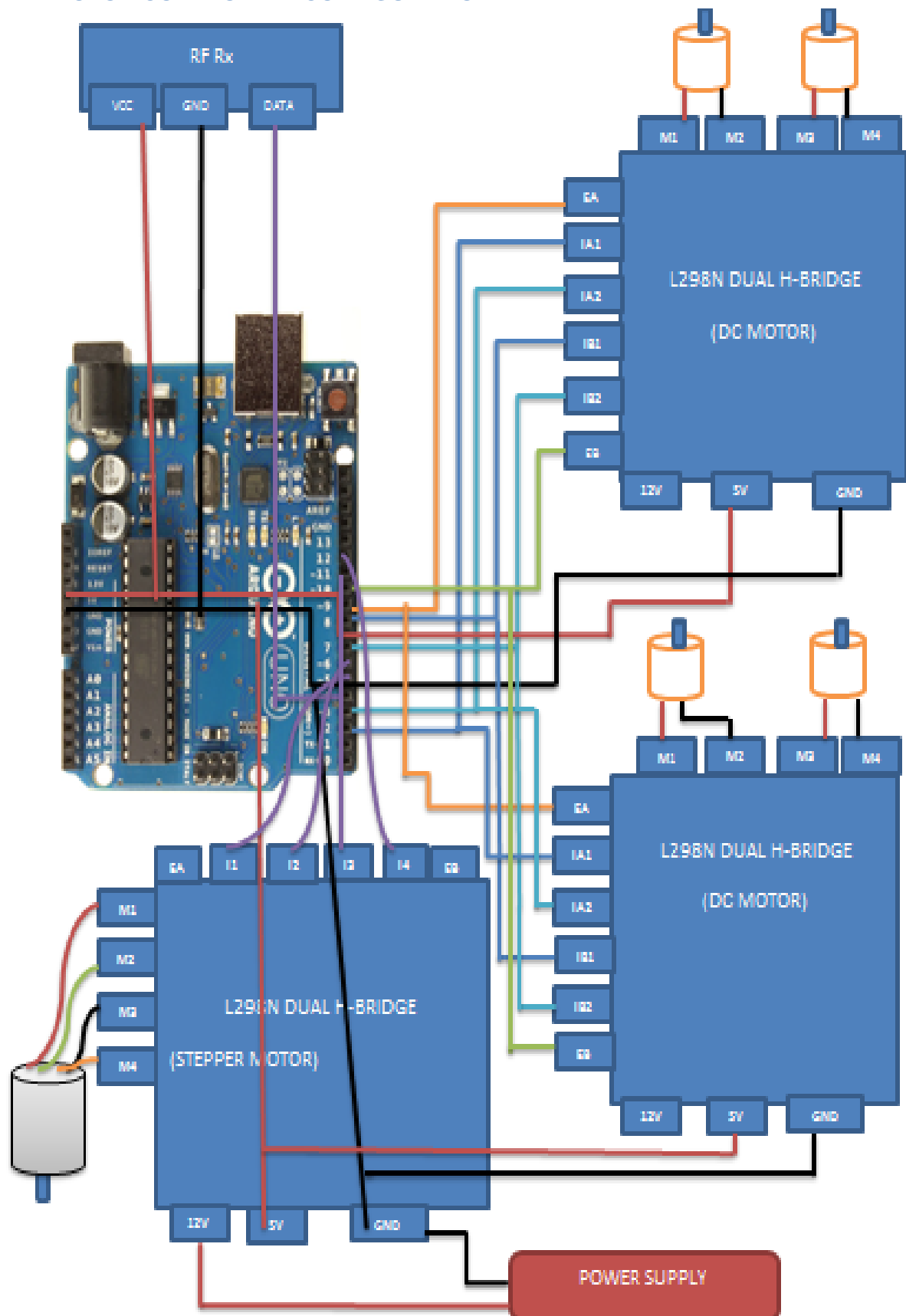


Figure 33 MOTOR PIN CONNECTION

- Here the EA, IA1, IA2, IB1, IB2 and EB are connected to the PINS 9, 2, 3, 10, 8, 7 respectively.
- Due to the lack of pines on the Arduino another L298N board is connected to the same pins in parallel.
- The L298n board is supplied with 5V from 5V PIN in the Arduino and GND is connected to the GND of the Arduino.
- Each of these boards can control two DC motors. So two DC motors are connected to the pins M1, M2 and M3, M3 respectively.
- For the stepper motor the pins IA1, IA2, IB1 and IB2 are connected to 5,6,11,12 pins in the Arduino
- The L298n board is supplied with 5V from 5V PIN in the Arduino and +12v from an external power supply and GND is a common ground connected to the GND of the Arduino and to the external power supply.
- All the pins M1, M2, M3 and M4 are all connected to one stepper motor.

#### 12.4.2. SEQUENCE OF THE CODE

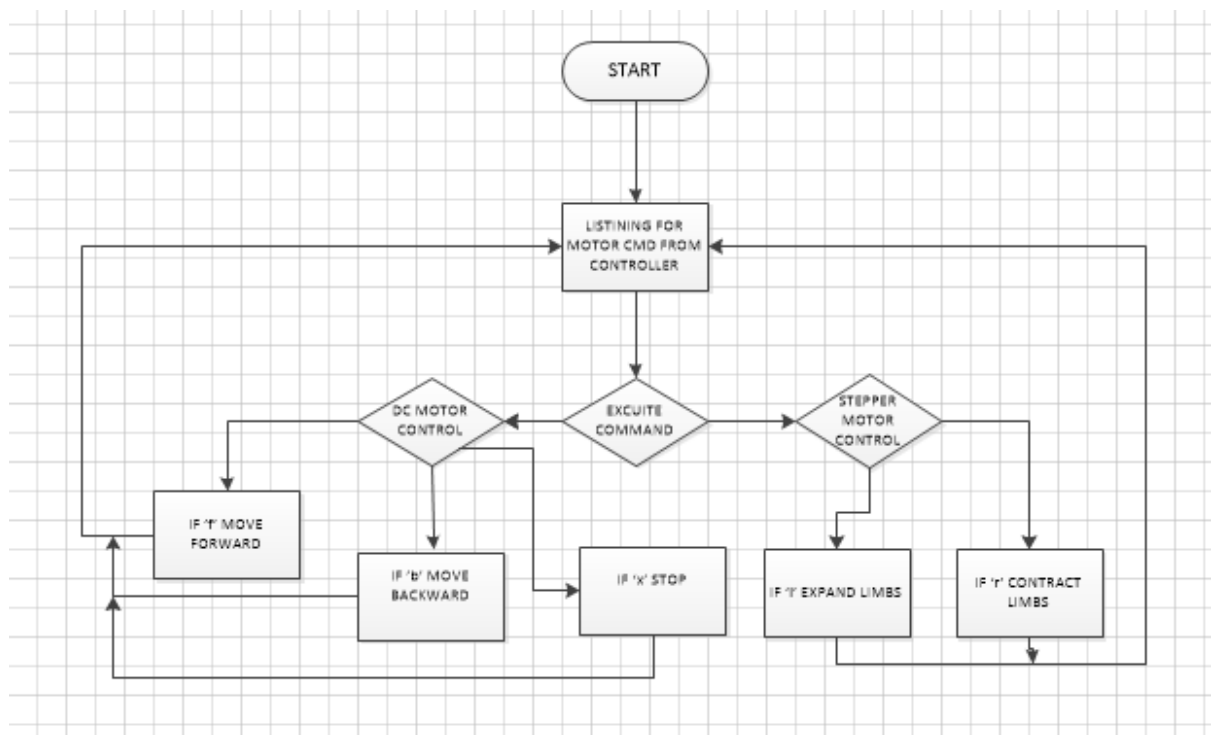


Figure 34 SEQUENCE FLOW OF MOTOR CONTROL

- Here at the start all the libraries and variables are initialized and the set up of all the pins so initialised.
- Then the receiver keeps listening for commands from the control hub.

- When it receives its commands it decides if it is for the DC Motor or the Stepper Motor.
- If it is 'f', 'b' or 'x' it would drive the switch the DC motor forward, backward and stop respectively.
- If it receives commands like 'l' or 'r' it would drive the stepper motor either to expand or contract respectively.
- All this is then while it is constantly looking for new commands.

### 12.4.3. SEQUENCE FLOW DIAGRAM

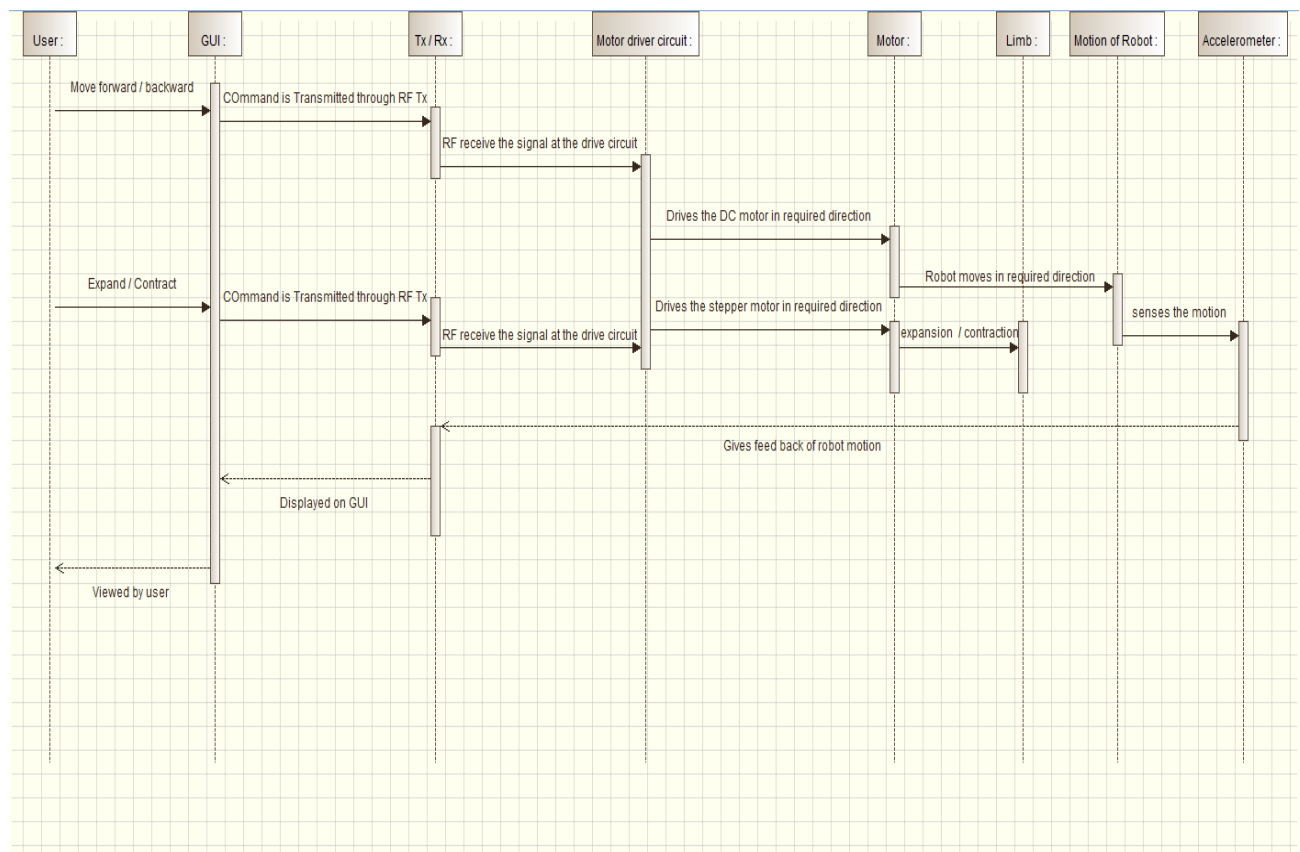


Figure 35 SEQUENCE DIAGRAM FOR MOTOR CONTROL

- When the user gives a command to control the DC motor through the GUI, this commands sent to the pipe inspection robot through the RF transmitter which is present in the remote control hub.
- The RF receiver on the pipe inspection robot receives this command and performs the required task of driving the motor driver circuit either forward or backward.

- The same is don for the stepper motor. When the user gives the command through the GUI it is transmitted to the pipe inspection robot through the transmitter.
- The receiver on the robot receives this command and performs the required task.
- The orientation and position of the robot can be got by the analysis of the accelerometer. Gyro and compass readings present on the robot.
- These values are sent to the user stored and they can be interpreted to tell the robots position.

#### 12.4.4. PROGRAM

```
#include <VirtualWire.h>
```

```
#include <Stepper.h>
```

```
const int stepsPerRevolution = 200; // change this to fit the number of steps per revolution
// for your motor
```

```
// initialize the stepper library on pins 8 through 11:
```

```
Stepper myStepper(stepsPerRevolution, 5,6,11,12);
```

```
// Motor 1
```

```
int dir1PinA = 2;
```

```
int dir2PinA = 3;
```

```
int speedPinA = 9; // Needs to be a PWM pin to be able to control motor speed
```

```
// Motor 2
```

```
int dir1PinB = 8;
```

```
int dir2PinB = 7;
```

```
int speedPinB = 10; // Needs to be a PWM pin to be able to control motor speed
```

```
int motorPin[] = {
```

```
  2, 3, 8, 7, 9, 10}; //array for storing pin nos
```

```

void setup()
{
    Serial.begin(9600); // Debugging only
    Serial.println("setup");

    // pinMode(VCC,OUTPUT);
    // pinMode(GND,OUTPUT);
    // digitalWrite(VCC,HIGH);
    // digitalWrite(GND,LOW);

    // Initialise the IO and ISR
    vw_setup(2000);    // Bits per sec
    vw_set_rx_pin(4); //Receiver at Digital Pin 2
    vw_rx_start(); // Start the receiver PLL running
    // Serial.println("setup");
    /*
    for (int i = 0; i < 6; i++)
    {
        pinMode(motorPin[i], OUTPUT);
        //Serial.println("pinmode");
    }*/

    //This is basically what the for loop does :-
    pinMode(dir1PinA,OUTPUT);
    pinMode(dir2PinA,OUTPUT);
    pinMode(speedPinA,OUTPUT);
    pinMode(dir1PinB,OUTPUT);
    pinMode(dir2PinB,OUTPUT);

```

```

pinMode(speedPinB,OUTPUT);

// set the stepper motor speed :
myStepper.setSpeed(100);

}

void loop()
{
  //Serial.println("loop");
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;
  //Serial.println("-----");
  //delay(500);

  if (vw_get_message(buf, &buflen)) // Non-blocking
  {
    Serial.println("vw_msg");
    int i;

    digitalWrite(13, true); // Flash a light to show received good message
    // Message with a good checksum received, dump it.

    for (i = 0; i < buflen; i++)
    {
      Serial.print(buf[i]);

      if(buf[i] == 'f')

```

```

    {
        forward();//go forward when f is pressed
    }
    if(buf[i] == 'b')
    {
        backward();//go backward when b is pressed
    }
    if(buf[i] == 'x')
    {
        stopMotor();//stop/brake when s is pressed
    }
    if(buf[i] == 'l')
    {
        stepperXpand();//go left when l is pressed
    }
    if(buf[i] == 'r')
    {
        stepperContract();//go right when r is pressed
    }

} //close for loop

digitalWrite(13, false);

} //close main if
} //close loop

//you can print the data entered when debugging by adding Serial.println

```



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//set of functions
```

```
void forward()
```

```
{
```

```
    Serial.println("forward");
```

```
    analogWrite(speedPinA, 255); //Sets speed variable via PWM
```

```
    digitalWrite(dir1PinA, LOW);
```

```
    digitalWrite(dir2PinA, HIGH);
```

```
    Serial.println("Motor 1 Forward"); // Prints out "Motor 1 Forward" on the serial monitor
```

```
    Serial.println(" "); // Creates a blank line printed on the serial monitor
```

```
    analogWrite(speedPinB, 255);
```

```
    digitalWrite(dir1PinB, LOW);
```

```
    digitalWrite(dir2PinB, HIGH);
```

```
    Serial.println("Motor 2 Forward");
```

```
    Serial.println(" ");
```

```
}
```

```
void backward()
```

```
{
```

```
    Serial.println("backward");
```

```
    analogWrite(speedPinA, 255);
```

```
    digitalWrite(dir1PinA, HIGH);
```

```
    digitalWrite(dir2PinA, LOW);
```

```
    Serial.println("Motor 1 Reverse");
```

```
    Serial.println(" ");
```

```
    analogWrite(speedPinB, 255);
```

```
digitalWrite(dir1PinB, HIGH);  
digitalWrite(dir2PinB, LOW);  
Serial.println("Motor 2 Reverse");  
Serial.println(" ");  
}
```

```
void stepperXpand()  
{  
    // step one revolution in one direction:  
    Serial.println("stepper clockwise");  
    myStepper.step(500);  
    delay(500);  
}
```

```
void stepperContract()  
{  
    Serial.println("stepper counterclockwise");  
    myStepper.step(-200);  
    delay(500);  
}
```

```
void stopMotor()  
{  
    Serial.println("stop");  
    analogWrite(speedPinA, 0);  
    digitalWrite(dir1PinA, LOW);  
    digitalWrite(dir2PinA, LOW);  
    Serial.println("Motor 1 Stop");  
}
```

```
Serial.println(" ");  
analogWrite(speedPinB, 0);  
digitalWrite(dir1PinB, LOW);  
digitalWrite(dir2PinB, LOW);  
Serial.println("Motor 2 Stop");  
Serial.println(" ");  
}  
//End Of Code
```

## 12.4.5. SENSOR MODULE

### 12.4.5.1. SENSOR MODULE PIN CONFIGURATION

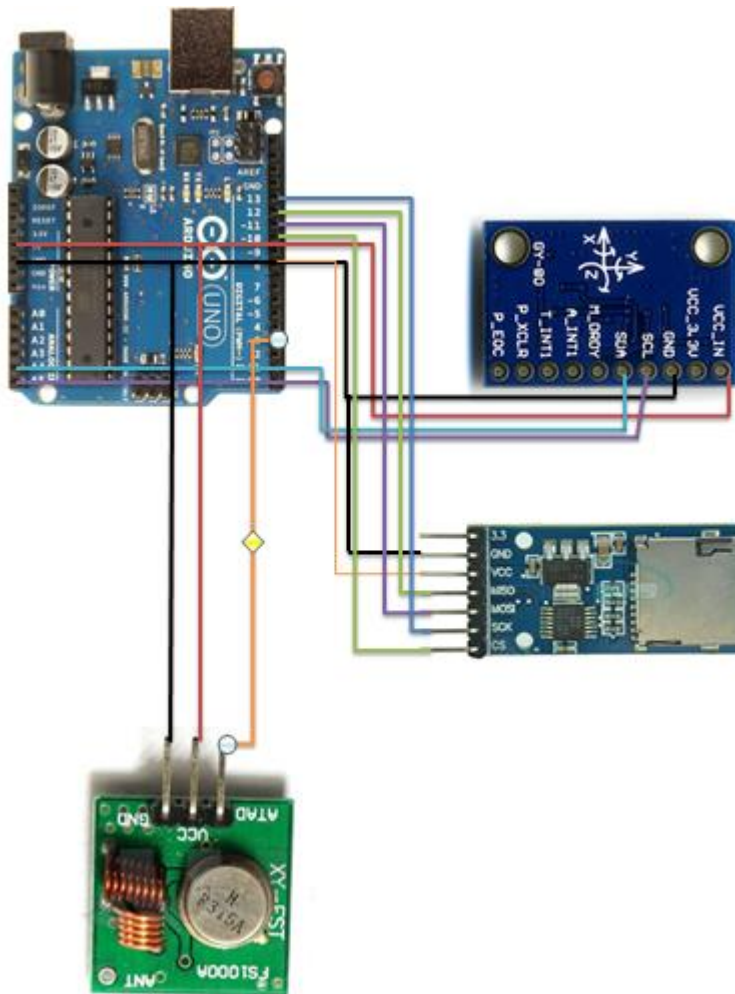


Figure 36 PIN CONNECTIONS SENSOR MODULE

From the above diagram we can see that

- The VCC of the sensor is connected to the 3.3v supply of the Arduino.
- The GND of the sensor is connected to the GND of the Arduino.
- SDA and SCL are connected to A5 and A4 respectively.
- The GND and the VCC of the SD card is connected to GND and PIN 8 of the Arduino respectively.
- MISO to PIN 12 and MOSI to PIN 11
- SCK to PIN 13
- CS (chip select) PIN 10

#### 12.4.5.2. SEQUENCE DIAGRAM FOR SENSOR MODULE

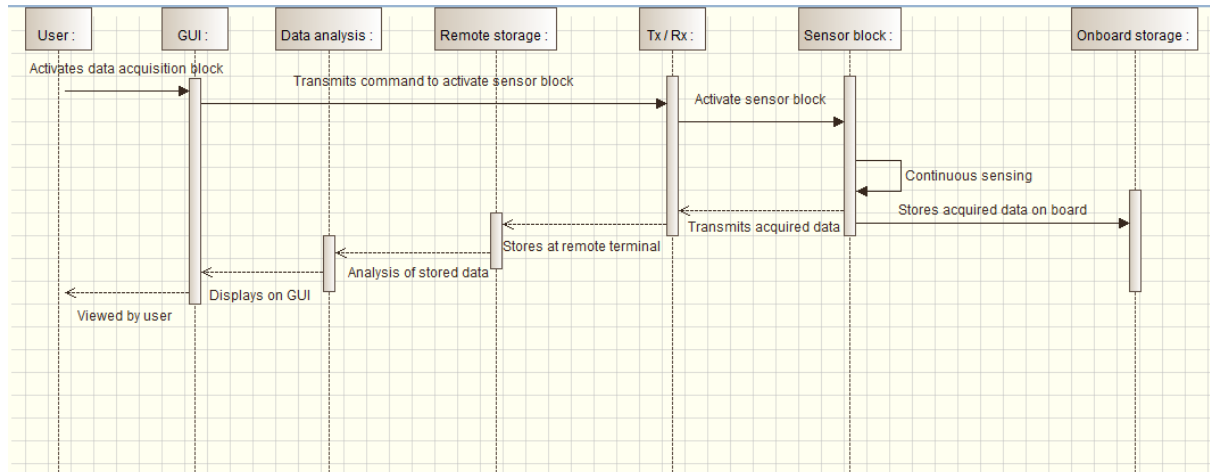


Figure 37 SEQUENCE FLOW DIAGRAM FOR SENSOR BLOCK

- From the above sequence diagram we can see that the user connects to the on board sensor block on the pipe inspection robot through the GUI that is present on the remote control hub.
- The sensor block is on'ed and is active and is constantly sensing the environment and transmitting the sensor values through the RF transmitter to the remote control hub.
- Simultaneously as it is transmitting the data, that data is being stored on an SD card fitted on the robot.
- The control hub receives the transmitted data through the RF receiver and stores this data and analyses it and displays it on the GUI for the user to view.

#### 12.4.5.3. SEQUENCE FLOW OF SENSOR MODULE CODE

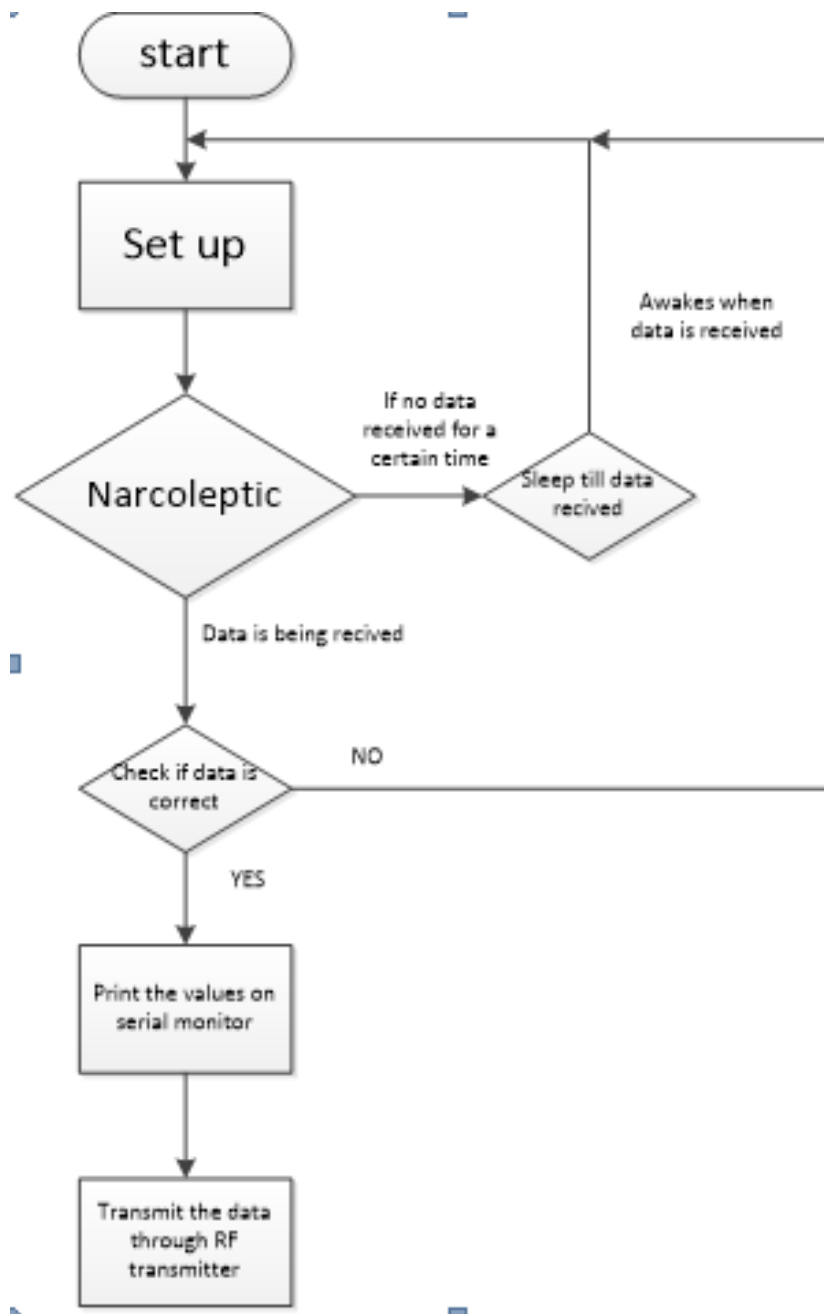


Figure 38 SEQUENCE FLOW OF SENSOR BLOCK

- Here at the start all the variables and libraries are initialized and then the required pins are initialised in the set up.
- We then go to the narcoleptic library which puts it to sleep if there is no activity from the robot then it would go to sleep to save power.

- It then checks the sensor reading and prints it on the screen if it's correct for debugging purposes.
- This data is then transmitted to the other end.

#### **12.4.5.4. PROGRAM**

```
#include <VirtualWire.h>
```

```
#include <Wire.h>
```

```
#include <I2Cdev.h>
```

```
#include <Narcoleptic.h>
```

```
#include <Adafruit_BMP085.h>
```

```
#include <Adafruit_Sensor.h>
```

```
#include <Adafruit_ADXL345_U.h>
```

```
#include <L3G4200D.h>
```

```
#include "HMC5883L.h"
```

```
#include <SD.h>
```

```
Adafruit_BMP085 bmp;
```

```
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
```

```
L3G4200D gyro;
```

```
HMC5883L mag;
```

```
int16_t avx, avy, avz;
```

```
int16_t mx, my, mz;
```

```
#define LED_PIN 13
```

```
bool blinkState = false;
```

```
int chipSelect = 10;
```

```
int pow_pin = 8;
```

```
long id = 1; //Serial No. in the SD Card File
```

```

void setup()
{
  Serial.begin(9600);
  Serial.println("set up");
  Wire.begin();

  // Initialise the IO and ISR
  vw_set_tx_pin(3);
  vw_setup(2000);

  if (!bmp.begin())
  {
    Serial.println(F("Could not find a valid BMP085 sensor, check wiring!"));
    while (1)
    {
    }
  }

  Serial.println(F("Accelerometer Test"));
  Serial.println("");
  if (!accel.begin())
  {
    /* There was a problem detecting the ADXL345 ... check your connections */
    Serial.println(F("Ooops, no ADXL345 detected ... Check your wiring!"));
    while(1);
  }

  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  gyro.initialize();

```



```

// verify connection

Serial.println(F("Testing device connections..."));

Serial.println(gyro.testConnection() ? "L3G4200D connection successful" : "L3G4200D
connection failed");


// configure LED for output
pinMode(LED_PIN, OUTPUT);


// data seems to be best when full scale is 2000
gyro.setFullScale(2000);


Serial.println(F("Initializing I2C devices..."));
mag.initialize();


// verify connection

Serial.println(F("Testing device connections..."));

Serial.println(mag.testConnection() ? "HMC5883L connection successful" : "HMC5883L
connection failed");


// configure Arduino LED for
pinMode(LED_PIN, OUTPUT);


Serial.println(F("Initializing SD Card Data Logger"));


pinMode(chipSelect, OUTPUT);


pinMode(pow_pin, OUTPUT);
digitalWrite(pow_pin, HIGH);


//Checking if card is ready

if(!SD.begin(chipSelect))

```

```

{
    Serial.println(F("Card Initialization Failed"));
    return;
}
else
{
    Serial.println(F("Card INITIALIZED"));
}

File logFile = SD.open("logger.txt", FILE_WRITE);
if(logFile)
{
    logFile.println(", , ");
    String header = "ID, Temp, Pa";
    logFile.println(header);
    logFile.close();
    Serial.println(header);
}
else
{
    Serial.println(F("Couldn't Open File in setup"));
}
Serial.println(F("-----"));
}

void loop()
{
    Narcoleptic.delay(500); // During this time power consumption is minimised

    int temp = (bmp.readTemperature()) ;

```

```

int Pa = (bmp.readPressure()*0.000145037738);

Serial.print(F("Temperature = "));
Serial.print(temp);
Serial.println(F(" *C"));
Serial.print(F("Pressure = "));
Serial.print(Pa);
Serial.println(F(" Psi"));

Serial.println();
delay(500);

sensors_event_t event;
accel.getEvent(&event);

int acc_x = event.acceleration.x;
int acc_y = event.acceleration.y;
int acc_z = event.acceleration.z;

int acc_pitch = atan(acc_x/sqrt(pow(acc_y,2) + pow(acc_z,2)));
int acc_roll = atan(acc_y/sqrt(pow(acc_x,2) + pow(acc_z,2)));

int pitch = acc_pitch*(180.0/PI);
int roll = acc_roll*(180.0/PI);

/* Display the results (acceleration is measured in m/s^2) */
Serial.print(F("X: "));
Serial.print(acc_x);
Serial.print(F(" "));
Serial.print(F("Y: "));

```

```

Serial.print(acc_y);
Serial.print(F(" "));
Serial.print(F("Z: "));
Serial.print(acc_z);
Serial.print(F(" "));
Serial.println(F("m/s^2 "));
Serial.println(F(" "));

Serial.print(F("Pitch :t"));
Serial.print(pitch);
Serial.print(F("\t"));
Serial.print(F("Roll :t"));
Serial.println(roll);
delay(500);

gyro.getAngularVelocity(&avx, &avy, &avz);

Serial.print(F("angular velocity:t"));
Serial.print(avx);
Serial.print(F("\t"));
Serial.print(avy);
Serial.print(F("\t"));
Serial.println(avz);
Serial.println(F(" "));
// blink LED to indicate activity
blinkState = !blinkState;
digitalWrite(LED_PIN, blinkState);

// read raw heading measurements from device
mag.getHeading(&mx, &my, &mz);

```

```

// display tab-separated gyro x/y/z values
Serial.print(F("mag:\t"));
Serial.print(F("x: "));
Serial.print(mx);
Serial.print(F("\t"));
Serial.print(F("\t"));
Serial.print(F("y: "));
Serial.print(my);
Serial.print(F("\t"));
Serial.print(F("\t"));
Serial.print(F("z: "));
Serial.println(mz);
Serial.println(F(" "));
Serial.print(F("\t"));

// To calculate heading in degrees. 0 degree indicates North
int heading = atan2(my, mx);
if(heading < 0)
    heading += 2 * M_PI;

Serial.print(F("heading:\t"));
int head = heading * 180/M_PI;
Serial.println(head);
Serial.println(F(" "));

// blink LED to indicate activity
blinkState = !blinkState;
digitalWrite(LED_PIN, blinkState);

// const int bytes = 30;

```

```

// uint8_t name[bytes];

const int bytes = 3 * sizeof(int);

int name[bytes];


//Serial.print("Size of buffer: ");

//Serial.println(sizeof(name));

name[0]=243;

name[1]=temp;

name[2]=Pa;

// name[3]=acc_x;

// name[4]=acc_y;

// name[5]=acc_z;

// name[6]=avx;

// name[7]=avy;

// name[8]=avz;

// name[9]=pitch;

// name[10]=roll;

// name[11]=mx;

// name[12]=my;

// name[13]=mz;

// name[14]=heading * 180/M_PI;

```

```

send((byte*)name,bytes);

delay(2000);

```

```

// }

```

```

// String dataString = String(id) + "," + String(temp) + "," + String(Pa) + "," + String(acc_x)
+ "," + String(acc_y) + "," + String(acc_z) + "," + String(avx) + "," + String(avy) + "," +
String(avz) + "," + String(pitch) + "," + String(roll) + "," + String(mx) + "," + String(my) + "," +
String(mz) + "," + String(head);

```

```

String dataString = String(id) + "," + String(temp) + "," + String(Pa);

//Open a file to write to
//Only one file can be opened at a time
File logFile = SD.open("logger.txt", FILE_WRITE);
if(logFile)
{
    logFile.println(dataString);
    logFile.close();
    Serial.println(dataString);
}
else
{
    Serial.println(F("Couldn't access file"));
}

id++;    //Increment id file

delay(1000);
}

```

#### **12.4.4.5. OUTPUT**

Here the sensor block reads the environment and sends that data to the control hub. For debugging purpose we print it out on the screen it is shown as follows.

```

set up
Accelerometer Test

Initializing I2C devices...
Testing device connections...
L3G4200D connection successful
Initializing I2C devices...
Testing device connections...
HMC5883L connection successful
Initializing SD Card Data Logger
Card INITIALIZED
ID, Temp, Pa
-----
Temperature = 23 *C
Pressure = 14 Psi

X: 10 Y: 0 Z: 0 m/s^2

Pitch : 57      Roll : 0
angular velocity: 16      0      -3

mag:    x: -409      y: 265      z: -442

      heading:      114

1,23,14

```

**Figure 39 SENSOR MODULE ON ROBOT**



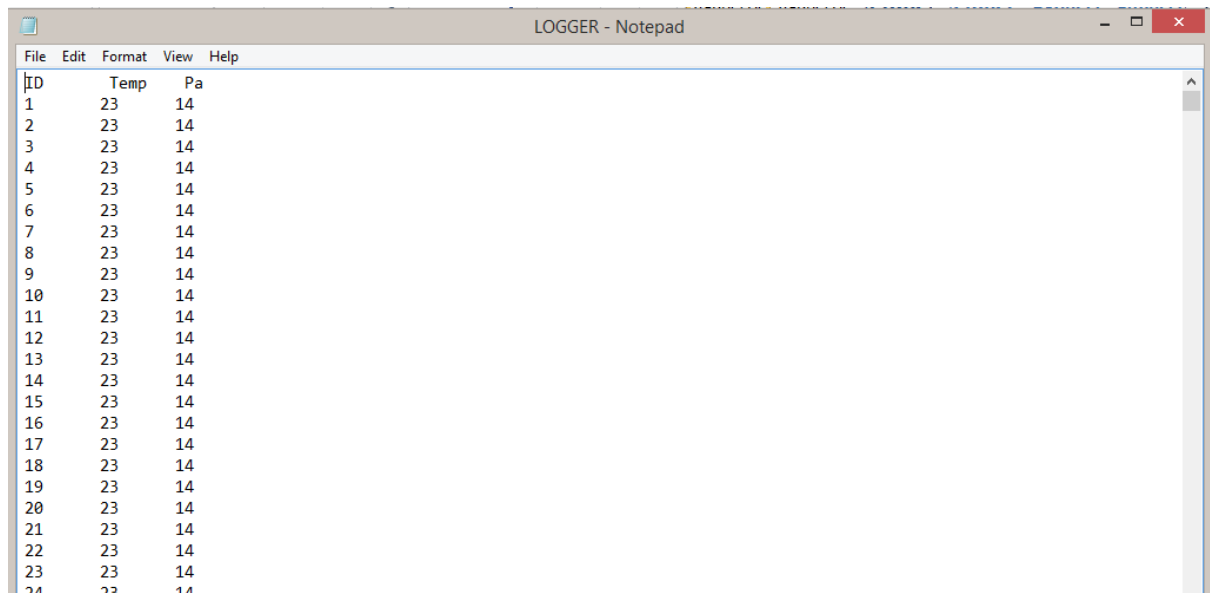
## 12.5. DATA ANALYSIS AND PLOTING

After the data is saved and save on to the SD card as a .txt file or a .csv file it is then read by the pc through MATLAB or other suitable software such as excel in order to analyse the data and plot it graphically for further analysis. The code for the matlab plot is as follows:

```
clc
clear all
sensor_data=xlsread('LOGGER.xlsx');
sample_data=sensor_data(:,1);
temp=sensor_data(:,2);
pressure=sensor_data(:,3);
figure;
subplot(1,2,1)
plot(sample_data,temp);
axis([0 150 0 50]);
grid on; %turn grid on
title('temperature data'); %adds a title to figure
xlabel('sample'); %adds label on the x axis
ylabel('temp(*C)');
%%figure;
subplot(1,2,2)
plot(sample_data,pressure);
axis([0 150 0 50]);
grid on; %turn grid on
title('pressure data'); %adds a title to figure
xlabel('sample'); %adds label on the x axis
ylabel('pressure(psi)');
```

### 12.4.4. OUTPUT

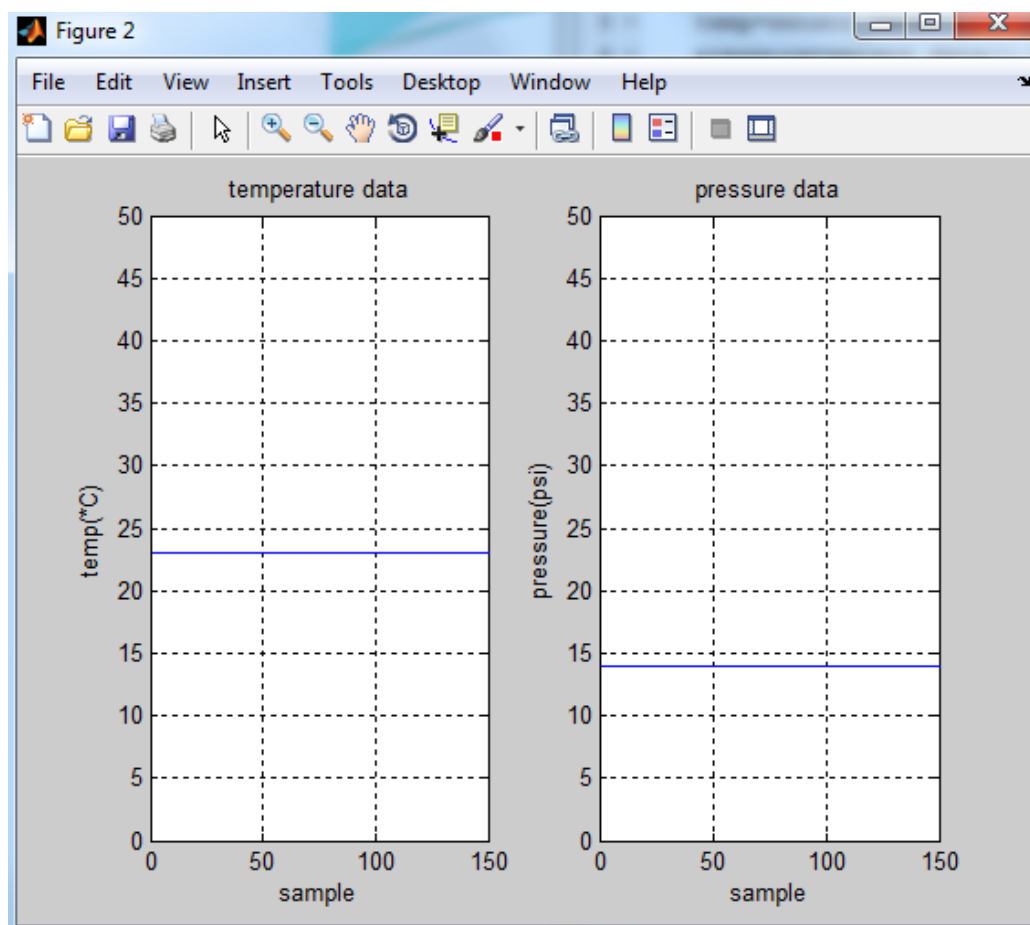
After the data is saved on the SD Card it is then taken and analysed by MATLAB and it is plotted on the screen.



| ID | Temp | Pa |
|----|------|----|
| 1  | 23   | 14 |
| 2  | 23   | 14 |
| 3  | 23   | 14 |
| 4  | 23   | 14 |
| 5  | 23   | 14 |
| 6  | 23   | 14 |
| 7  | 23   | 14 |
| 8  | 23   | 14 |
| 9  | 23   | 14 |
| 10 | 23   | 14 |
| 11 | 23   | 14 |
| 12 | 23   | 14 |
| 13 | 23   | 14 |
| 14 | 23   | 14 |
| 15 | 23   | 14 |
| 16 | 23   | 14 |
| 17 | 23   | 14 |
| 18 | 23   | 14 |
| 19 | 23   | 14 |
| 20 | 23   | 14 |
| 21 | 23   | 14 |
| 22 | 23   | 14 |
| 23 | 23   | 14 |
| 24 | 23   | 14 |

**Figure 40** **LOGGER FILE WHICH STORES SENSOR DATA**

These values are then sent to Matlab and with the help of the code the values are plotted in MATLAB. This is shown in figure 37.



**Figure 41** **PLOT OF TEMPERATURE AND PRESSURE DATA**

## 13. Future work:

For the future development of the project we are working on a GUI which would make navigating the various features and options of the menu system of the robot more attractive and easy. The GUI would have a log in page so that only the authorised user can log in to use the system the login page is shown in figure 38.

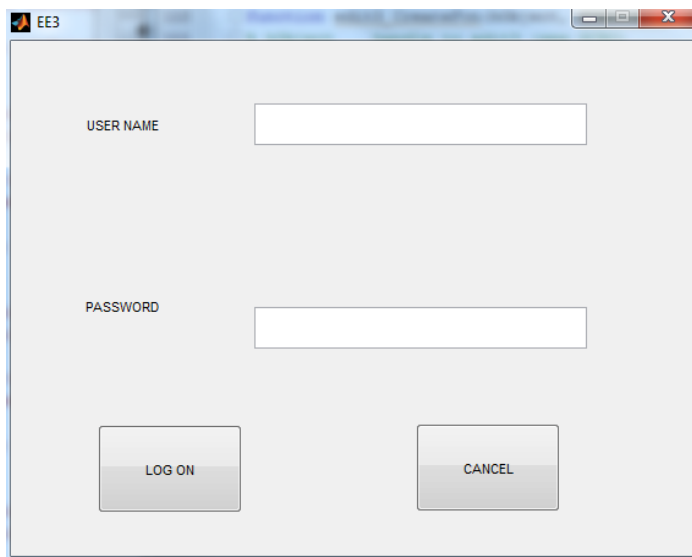


Figure 42 GUI LOG IN PAGE

Code

```
function varargout = EE3(varargin)
% EE3 MATLAB code for EE3.fig
%   EE3, by itself, creates a new EE3 or raises the existing
%   singleton*.
%
%   H = EE3 returns the handle to a new EE3 or the handle to
%   the existing singleton*.
%
%   EE3('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in EE3.M with the given input arguments.
%
%   EE3('Property','Value',...) creates a new EE3 or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before EE3_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property application
```

```

% stop. All inputs are passed to EE3_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help EE3

% Last Modified by GUIDE v2.5 20-May-2015 13:10:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @EE3_OpeningFcn, ...
    'gui_OutputFcn', @EE3_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before EE3 is made visible.
function EE3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to EE3 (see VARARGIN)

```

```

% Choose default command line output for EE3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes EE3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = EE3_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
% str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

```

```
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit3_Callback(hObject, eventdata, handles)
% hObject handle to edit3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
% str2double(get(hObject,'String')) returns contents of edit3 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
```

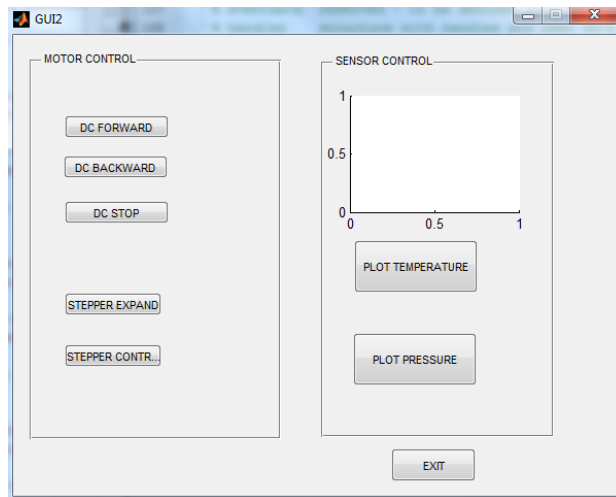
```
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

The next page will show us the controls for the motor control and also a real time screen for plotting the sensor values.



**Figure 43 GUI CONTROL PAGE**

Code

```
function varargout = GUI2(varargin)
% GUI2 MATLAB code for GUI2.fig
%   GUI2, by itself, creates a new GUI2 or raises the existing
%   singleton*.
%
%   H = GUI2 returns the handle to a new GUI2 or the handle to
%   the existing singleton*.
%
%   GUI2('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in GUI2.M with the given input arguments.
%
%   GUI2('Property','Value',...) creates a new GUI2 or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before GUI2_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to GUI2_OpeningFcn via varargin.
%
```

```
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help GUI2
```

```
% Last Modified by GUIDE v2.5 20-May-2015 13:15:36
```

```
% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @GUI2_OpeningFcn, ...
    'gui_OutputFcn', @GUI2_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before GUI2 is made visible.
```

```
function GUI2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI2 (see VARARGIN)
```

```
% Choose default command line output for GUI2
```

```
handles.output = hObject;
```



% Update handles structure

```
guidata(hObject, handles);
```

% UIWAIT makes GUI2 wait for user response (see UIRESUME)

```
% uiwait(handles.figure1);
```

% --- Outputs from this function are returned to the command line.

```
function varargout = GUI2_OutputFcn(hObject, eventdata, handles)
```

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

```
varargout{1} = handles.output;
```

% --- Executes on button press in pushbutton6.

```
function pushbutton6_Callback(hObject, eventdata, handles)
```

% hObject handle to pushbutton6 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton7.

```
function pushbutton7_Callback(hObject, eventdata, handles)
```

% hObject handle to pushbutton7 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton8.

```
function pushbutton8_Callback(hObject, eventdata, handles)
```

% hObject handle to pushbutton8 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

```
% --- Executes on button press in pushbutton1.  
function pushbutton1_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton1 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton2.  
function pushbutton2_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton2 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton3.  
function pushbutton3_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton3 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton4.  
function pushbutton4_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton4 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton5.  
function pushbutton5_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton5 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

The other development would be to add a wheel encoder to tell the distance travelled and integrate it with the IMU data to plot the trajectory of the robot.

## 14. CONCLUSION:

Here we have presented a report which deals with the entire controls construction and working of the basic requirement for an in-pipe inspection robot. This will help us monitor and control the temperature and pressure in pipes and we have devices a flexible system which is controlled using RF technology.

# BIBLIOGRAPHY

## REFERENCE

[1] Iszmir Nazmi Ismail, Development of In-pipe Inspection Robot: a Review, Centre for Advanced Mechatronics and Robotics Universiti Tenaga Nasional, 2012 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT)

[2] Junghu Min, Development and Controller Design of Wheeled- Type Pipe Inspection Robot, Department of Mechanical Design Engineering Pukyong National University, 2014 *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*

[3] E. Omerdic, D. Toal, G. Dooly, A. Kaknjo, Remote Presence: Long Endurance Robotic Systems for Routine Inspection of Offshore Subsea Oil & Gas Installations and Marine Renewable Energy Devices

[4] Joffin George, *Manipulator Robot for Crack Detection and Welding in Underground Process Pipes*, International Conference on Magnetics, Machines & Drives

[5] Dongmei Wu, Drain Pipe Inspection Robot using Wireless Communication System, August 18-21, 2009, Fukuoka International Congress Center, Japan

## INTERNET

<http://www.arduino.cc/en/Reference/HomePage> - Various Pages Accessed

<http://www.mathworks.co.uk> – Various Pages Accessed

<http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SimulinkModeling> - Various Pages Accessed

<http://sysml.org/> - Various Pages Accessed

computergeek,” Simple 2-way motor control for the arduino”, computergeek [online]

<http://www.instructables.com/id/Simple-2-way-motor-control-for-the-arduino/> [accessed: 25/04/2015]

# APPENDIX

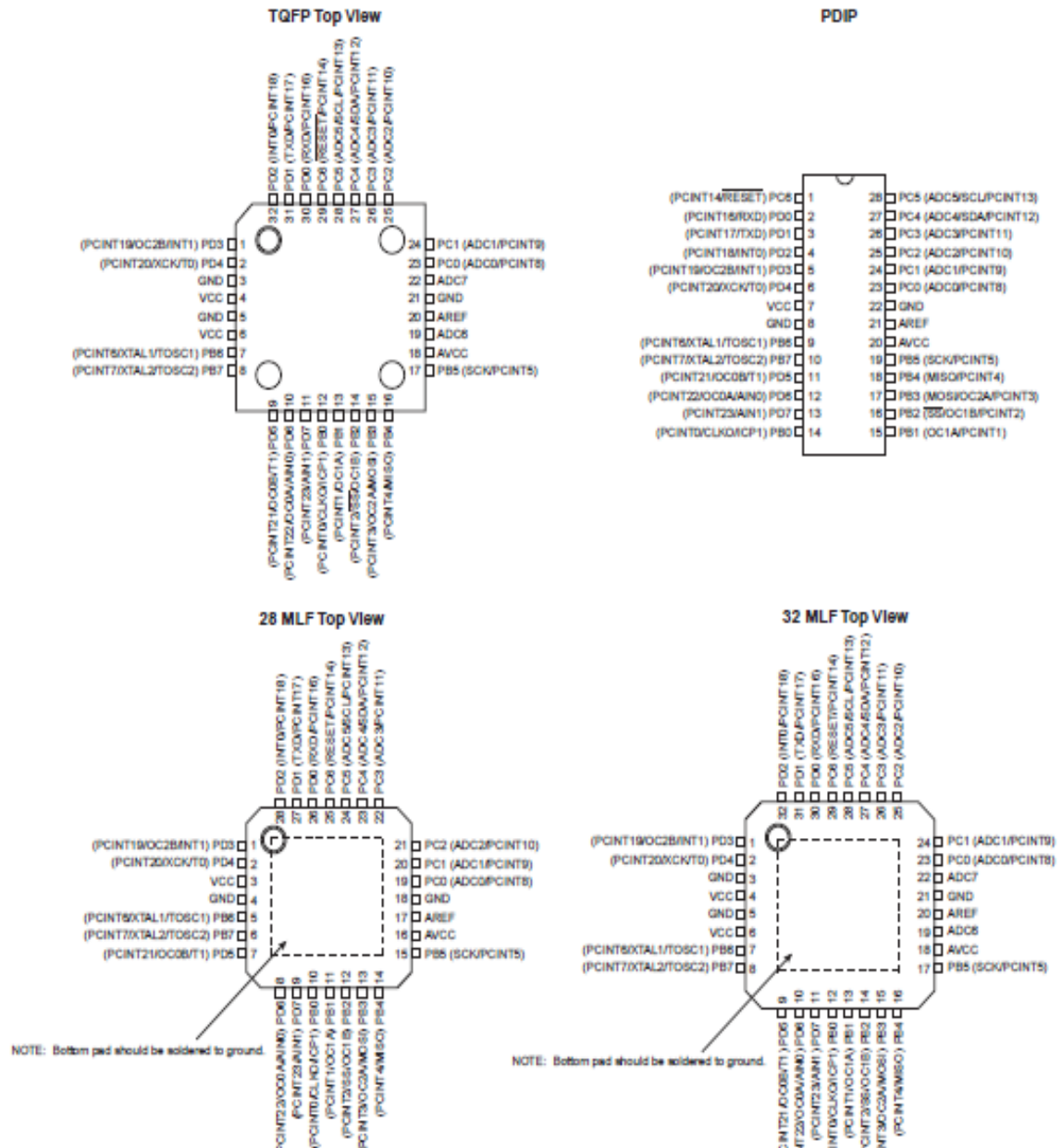
## APPENDIX1:

### Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4K/8K/16K/32K Bytes of In-System Self-Programmable Flash program memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/1K/2K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C<sup>(1)</sup>
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I<sup>2</sup>C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V

# 1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P



## 1.1 Pin Descriptions

### 1.1.1 VCC

Digital supply voltage.

### 1.1.2 GND

Ground.

### 1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in ["Alternate Functions of Port B" on page 82](#) and ["System Clock and Clock Options" on page 28](#).

### 1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

### 1.1.5 PC6/RESET

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in [Table 28-3 on page 318](#). Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ["Alternate Functions of Port C" on page 85](#).

### 1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.





## L3G4200D

### MEMS motion sensor: ultra-stable three-axis digital output gyroscope

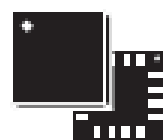
Preliminary data

#### Features

- Three selectable full scales (250/500/2000 dps)
- I<sup>2</sup>C/SPI digital output interface
- 16 bit-rate value data output
- 8-bit temperature data output
- Two digital output lines (Interrupt and data ready)
- Integrated low- and high-pass filters with user-selectable bandwidth
- Ultra-stable over temperature and time
- Wide supply voltage: 2.4 V to 3.6 V
- Low voltage-compatible I/Os (1.8 V)
- Embedded power-down and sleep mode
- Embedded temperature sensor
- Embedded FIFO
- High shock survivability
- Extended operating temperature range (-40 °C to +85 °C)
- ECOPACK® RoHS and "Green" compliant

#### Applications

- Gaming and virtual reality input devices
- Motion control with MMI (man-machine interface)
- GPS navigation systems
- Appliances and robotics



LGA-16 (4x4x1.1 mm)

#### Description

The L3G4200D is a low-power three-axis angular rate sensor able to provide unprecedented stability of zero rate level and sensitivity over temperature and time. It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I<sup>2</sup>C/SPI).

The sensing element is manufactured using a dedicated micro-machining process developed by STMicroelectronics to produce inertial sensors and actuators on silicon wafers.

The IC interface is manufactured using a CMOS process that allows a high level of integration to design a dedicated circuit which is trimmed to better match the sensing element characteristics.

The L3G4200D has a full scale of  $\pm 250/\pm 500/\pm 2000$  dps and is capable of measuring rates with a user-selectable bandwidth.

The L3G4200D is available in a plastic land grid array (LGA) package and can operate within a temperature range of -40 °C to +85 °C.

Table 1. Device summary

| Order code | Temperature range (°C) | Package             | Packing       |
|------------|------------------------|---------------------|---------------|
| L3G4200D   | -40 to +85             | LGA-16 (4x4x1.1 mm) | Tray          |
| L3G4200DTR | -40 to +85             | LGA-16 (4x4x1.1 mm) | Tape and reel |



## 1.1 Pin description

Figure 2. Pin connection

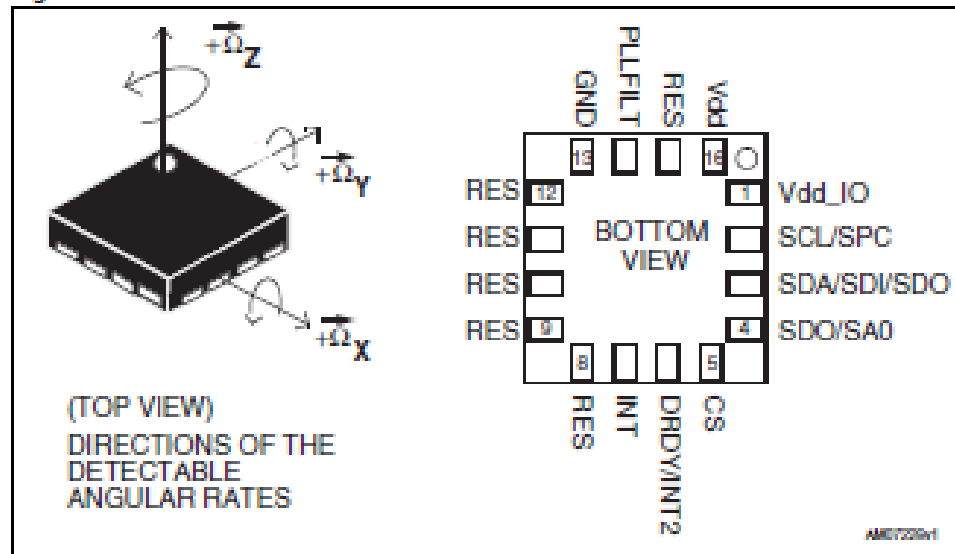


Table 2. Pin description

| Pin# | Name              | Function   |
|------|-------------------|--|
| 1    | Vdd_IO            | Power supply for I/O pins  |
| 2    | SCL<br>SPC        | I <sup>2</sup> C serial clock (SCL)<br>SPI serial port clock (SPC)   |
| 3    | SDA<br>SDI<br>SDO | I <sup>2</sup> C serial data (SDA)<br>SPI serial data input (SDI)<br>3-wire interface serial data output (SDO)   |
| 4    | SDO<br>SA0        | SPI serial data output (SDO)<br>I <sup>2</sup> C least significant bit of the device address (SA0)   |
| 5    | CS                | SPI enable<br>I <sup>2</sup> C/SPI mode selection (1: SPI idle mode / I <sup>2</sup> C communication enabled; 0: SPI communication mode / I <sup>2</sup> C disabled) |
| 6    | DRDY/INT2         | Data ready/FIFO interrupt  |
| 7    | INT1              | Programmable interrupt   |
| 8    | Reserved          | Connect to GND   |
| 9    | Reserved          | Connect to GND   |
| 10   | Reserved          | Connect to GND   |
| 11   | Reserved          | Connect to GND   |
| 12   | Reserved          | Connect to GND   |
| 13   | GND               | 0 V supply   |
| 14   | PLLFLT            | Phase-locked loop filter (see <a href="#">Figure 3</a> )   |
| 15   | Reserved          | Connect to Vdd   |
| 16   | Vdd               | Power supply   |

## 2 Mechanical and electrical characteristics

### 2.1 Mechanical characteristics

Table 4. Mechanical characteristics @ Vdd = 3.0 V, T = 25 °C, unless otherwise noted<sup>(1)</sup>

| Symbol | Parameter   | Test condition         | Min. | Typ. <sup>(2)</sup> | Max. | Unit             |
|--------|---|------------------------|------|---------------------|------|------------------|
| FS     | Measurement range                                     | User-selectable        |      | ±250                |      | dps              |
|        |   |                        |      | ±500                |      |                  |
|        |   |                        |      | ±2000               |      |                  |
| So     | Sensitivity   | FS = 250 dps           |      | 8.75                |      | mdps/digit       |
|        |   | FS = 500 dps           |      | 17.50               |      |                  |
|        |   | FS = 2000 dps          |      | 70                  |      |                  |
| SoDr   | Sensitivity change vs. temperature                    | From -40 °C to +85 °C  |      | ±2                  |      | %                |
| DVoff  | Digital zero-rate level                               | FS = 250 dps           |      | ±10                 |      | dps              |
|        |   | FS = 500 dps           |      | ±15                 |      |                  |
|        |   | FS = 2000 dps          |      | ±75                 |      |                  |
| OoDr   | Zero-rate level change vs. temperature <sup>(3)</sup> | FS = 250 dps           |      | ±0.03               |      | dps/°C           |
|        |   | FS = 2000 dps          |      | ±0.04               |      | dps/°C           |
| NL     | Non linearity <sup>(4)</sup>                          | Best fit straight line |      | 0.2                 |      | % FS             |
| DST    | Self-test output change                               | FS = 250 dps           |      | 130                 |      | dps              |
|        |   | FS = 500 dps           |      | 200                 |      |                  |
|        |   | FS = 2000 dps          |      | 530                 |      |                  |
| Rn     | Rate noise density                                    | BW = 50 Hz             |      | 0.03                |      | dps/<br>sqrt(Hz) |
| ODR    | Digital output data rate                              |                        |      | 100/200/<br>400/800 |      | Hz               |
| Top    | Operating temperature range                           |                        | -40  |                     | +85  | °C               |

1. The product is factory calibrated at 3.0 V. The operational power supply range is specified in [Table 5](#).

2. Typical specifications are not guaranteed.

3. Min/max values have been estimated based on the measurements of the current gyros in production.

4. Guaranteed by design.

## 2.2 Electrical characteristics

Table 5. Electrical characteristics @ Vdd =3.0 V, T=25 °C, unless otherwise noted<sup>(1)</sup>

| Symbol | Parameter                                   | Test condition                  | Min. | Typ. <sup>(2)</sup> | Max.    | Unit |
|--------|---|---------------------------------|------|---------------------|---------|------|
| Vdd    | Supply voltage                              |                                 | 2.4  | 3.0                 | 3.6     | V    |
| Vdd_IO | I/O pins supply voltage <sup>(3)</sup>      |                                 | 1.71 |                     | Vdd+0.1 | V    |
| Idd    | Supply current                              |                                 |      | 6.1                 |         | mA   |
| IddSL  | Supply current in sleep mode <sup>(4)</sup> | Selectable by digital interface |      | 1.5                 |         | mA   |
| IddPdn | Supply current in power-down mode           | Selectable by digital interface |      | 5                   |         | µA   |
| Top    | Operating temperature range                 |                                 | -40  |                     | +85     | °C   |

1. The product is factory calibrated at 3.0 V.

2. Typical specifications are not guaranteed.

3. It is possible to remove Vdd maintaining Vdd\_IO without blocking the communication busses, in this condition the measurement chain is powered off.

4. Sleep mode introduces a faster turn-on time compared to power-down mode.

## 2.3 Temperature sensor characteristics

Table 6. Temp. sensor characteristics @ Vdd =3.0 V, T=25 °C, unless otherwise noted<sup>(1)</sup>

| Symbol | Parameter  | Test condition | Min. | Typ. <sup>(2)</sup> | Max. | Unit     |
|--------|--|----------------|------|---------------------|------|----------|
| TSDr   | Temperature sensor output change vs. temperature |                |      | -1                  |      | °C/digit |
| TODR   | Temperature refresh rate                         |                |      | 1                   |      | Hz       |
| Top    | Operating temperature range                      |                | -40  |                     | +85  | °C       |

1. The product is factory calibrated at 3.0 V.

2. Typical specifications are not guaranteed.

## 3-Axis Digital Compass IC HMC5883L

**Honeywell**
*Advanced Information*

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I<sup>2</sup>C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.



The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

### FEATURES

### BENEFITS

- |  |  |
|--|--|
| ▶ 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package                    | ▶ Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly |
| ▶ 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in ±8 Gauss Fields | ▶ Enables 1° to 2° Degree Compass Heading Accuracy   |
| ▶ Built-In Self Test   | ▶ Enables Low-Cost Functionality Test after Assembly in Production   |
| ▶ Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 µA)                                 | ▶ Compatible for Battery Powered Applications  |
| ▶ Built-In Strap Drive Circuits  | ▶ Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation  |
| ▶ I <sup>2</sup> C Digital Interface   | ▶ Popular Two-Wire Serial Data Interface for Consumer Electronics  |
| ▶ Lead Free Package Construction   | ▶ RoHS Compliance  |
| ▶ Wide Magnetic Field Range (±8 Oe)  | ▶ Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy  |
| ▶ Software and Algorithm Support Available   | ▶ Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available   |
| ▶ Fast 160 Hz Maximum Output Rate  | ▶ Enables Pedestrian Navigation and LBS Applications   |

## HMC5883L

### SPECIFICATIONS (\* Tested at 25°C except stated otherwise.)

| Characteristics      | Conditions*  | Min  | Typ | Max     | Units |
|----------------------|--|------|-----|---------|-------|
| <b>Power Supply</b>  |  |      |     |         |       |
| Supply Voltage       | VDD Referenced to AGND   | 2.16 | 2.5 | 3.6     | Volts |
|                      | VDDIO Referenced to DGND   | 1.71 | 1.8 | VDD+0.1 | Volts |
| Average Current Draw | Idle Mode  | -    | 2   | -       | µA    |
|                      | Measurement Mode (7.5 Hz ODR;<br>No measurement average, MA1:MA0 = 00)<br>VDD = 2.5V, VDDIO = 1.8V (Dual Supply)<br>VDD = VDDIO = 2.5V (Single Supply) | -    | 100 | -       | µA    |

### Performance

|                                   |   |             |           |             |             |
|-----------------------------------|---|-------------|-----------|-------------|-------------|
| Field Range                       | Full scale (FS)   | -8          |           | +8          | gauss       |
| Mag Dynamic Range                 | 3-bit gain control  | ±1          |           | ±8          | gauss       |
| Sensitivity (Gain)                | VDD=3.0V, GN=0 to 7, 12-bit ADC   | 230         |           | 1370        | LSb/gauss   |
| Digital Resolution                | VDD=3.0V, GN=0 to 7, 1-LSb, 12-bit ADC  | 0.73        |           | 4.35        | milli-gauss |
| Noise Floor<br>(Field Resolution) | VDD=3.0V, GN=0, No measurement<br>average, Standard Deviation 100 samples<br>(See typical performance graphs below) |             | 2         |             | milli-gauss |
| Linearity                         | ±2.0 gauss input range  |             |           | 0.1         | ±% FS       |
| Hysteresis                        | ±2.0 gauss input range  |             | ±25       |             | ppm         |
| Cross-Axis Sensitivity            | Test Conditions: Cross field = 0.5 gauss,<br>Applied = ±3 gauss   |             | ±0.2%     |             | %FS/gauss   |
| Output Rate (ODR)                 | Continuous Measurement Mode   | 0.75        |           | 75          | Hz          |
|                                   | Single Measurement Mode   |             |           | 160         | Hz          |
| Measurement Period                | From receiving command to data ready  |             | 6         |             | ms          |
| Turn-on Time                      | Ready for I2C commands  |             | 200       |             | µs          |
|                                   | Analog Circuit Ready for Measurements   |             | 50        |             | ms          |
| Gain Tolerance                    | All gain/dynamic range settings   |             | ±5        |             | %           |
| I <sup>2</sup> C Address          | 8-bit read address  |             | 0x3D      |             | hex         |
|                                   | 8-bit write address   |             | 0x3C      |             | hex         |
| I <sup>2</sup> C Rate             | Controlled by I <sup>2</sup> C Master   |             |           | 400         | kHz         |
| I <sup>2</sup> C Hysteresis       | Hysteresis of Schmitt trigger inputs on SCL<br>and SDA - Fall (VDDIO=1.8V)<br>Rise (VDDIO=1.8V)                     |             | 0.2*VDDIO |             | Volts       |
|                                   |   |             | 0.8*VDDIO |             | Volts       |
| Self Test                         | X & Y Axes  |             | ±1.16     |             | gauss       |
|                                   | Z Axis  |             | ±1.08     |             | gauss       |
|                                   | X & Y & Z Axes (GN=5) Positive Bias<br>X & Y & Z Axes (GN=5) Negative Bias  | 243<br>-575 |           | 575<br>-243 | LSb         |
| Sensitivity Tempco                | T <sub>A</sub> = -40 to 125°C, Uncompensated Output   |             | -0.3      |             | %/°C        |

### General

|                       |                                 |     |  |      |       |
|-----------------------|---------------------------------|-----|--|------|-------|
| ESD Voltage           | Human Body Model (all pins)     |     |  | 2000 | Volts |
|                       | Charged Device Model (all pins) |     |  | 750  | Volts |
| Operating Temperature | Ambient                         | -30 |  | 85   | °C    |
| Storage Temperature   | Ambient, unbiased               | -40 |  | 125  | °C    |

## HMC5883L

| Characteristics       | Conditions*                    | Min  | Typ  | Max  | Units |
|-----------------------|--------------------------------|------|------|------|-------|
| Reflow Classification | MSL 3, 260 °C Peak Temperature |      |      |      |       |
| Package Size          | Length and Width               | 2.85 | 3.00 | 3.15 | mm    |
| Package Height        |                                | 0.8  | 0.9  | 1.0  | mm    |
| Package Weight        |                                |      | 18   |      | mg    |

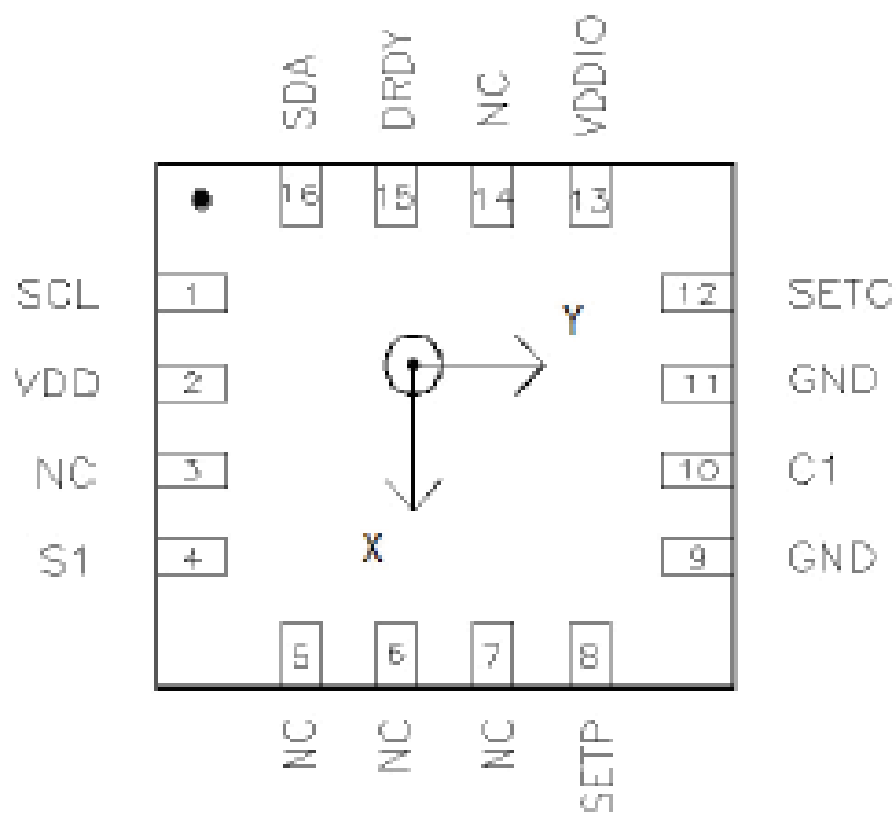
**Absolute Maximum Ratings** (\* Tested at 25°C except stated otherwise.)

| Characteristics      | Min  | Max | Units |
|----------------------|------|-----|-------|
| Supply Voltage VDD   | -0.3 | 4.8 | Volts |
| Supply Voltage VDDIO | -0.3 | 4.8 | Volts |

### PIN CONFIGURATIONS

| Pin | Name  | Description  |
|-----|-------|--|
| 1   | SCL   | Serial Clock – I <sup>2</sup> C Master/Slave Clock   |
| 2   | VDD   | Power Supply (2.16V to 3.6V)   |
| 3   | NC    | Not to be Connected  |
| 4   | S1    | Tie to VDDIO   |
| 5   | NC    | Not to be Connected  |
| 6   | NC    | Not to be Connected  |
| 7   | NC    | Not to be Connected  |
| 8   | SETP  | Set/Reset Strap Positive – S/R Capacitor (C2) Connection   |
| 9   | GND   | Supply Ground  |
| 10  | C1    | Reservoir Capacitor (C1) Connection  |
| 11  | GND   | Supply Ground  |
| 12  | SETC  | S/R Capacitor (C2) Connection – Driver Side  |
| 13  | VDDIO | IO Power Supply (1.71V to VDD)   |
| 14  | NC    | Not to be Connected  |
| 15  | DRDY  | Data Ready, Interrupt Pin. Internally pulled high. Optional connection. Low for 250 usec when data is placed in the data output registers. |
| 16  | SDA   | Serial Data – I <sup>2</sup> C Master/Slave Data   |

Table 1: Pin Configurations





## 3-Axis, $\pm 2\text{ g}/\pm 4\text{ g}/\pm 8\text{ g}/\pm 16\text{ g}$ Digital Accelerometer

### ADXL345

#### FEATURES

- Ultralow power: as low as 40  $\mu\text{A}$  in measurement mode and 0.1  $\mu\text{A}$  in standby mode at  $V_s = 2.5\text{ V}$  (typical)
- Power consumption scales automatically with bandwidth
- User-selectable resolution
- Fixed 10-bit resolution
- Full resolution, where resolution increases with  $g$  range, up to 13-bit resolution at  $\pm 16\text{ g}$  (maintaining 4 mg/LSB scale factor in all  $g$  ranges)
- Embedded, patent pending FIFO technology minimizes host processor load
- Tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to  $V_s$
- SPI (3- and 4-wire) and I<sup>2</sup>C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range ( $-40^\circ\text{C}$  to  $+85^\circ\text{C}$ )
- 10,000  $g$  shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm  $\times$  5 mm  $\times$  1 mm LGA package

#### APPLICATIONS

- Handsets
- Medical instrumentation
- Gaming and pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive (HDD) protection
- Fitness equipment

#### GENERAL DESCRIPTION

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to  $\pm 16\text{ g}$ . Digital output data is formatted as 16-bit two's complement and is accessible through either a SPI (3- or 4-wire) or I<sup>2</sup>C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than  $1.0^\circ$ .

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm  $\times$  5 mm  $\times$  1 mm, 14-lead, plastic package.

#### FUNCTIONAL BLOCK DIAGRAM



## SPECIFICATIONS

T<sub>A</sub> = 25°C, V<sub>S</sub> = 2.5 V, V<sub>DDIO</sub> = 1.8 V, acceleration = 0 g, C<sub>S</sub> = 1 µF tantalum, C<sub>IO</sub> = 0.1 µF, unless otherwise noted.

Table 1. Specifications<sup>1</sup>

| Parameter  | Test Conditions  | Min   | Typ             | Max            | Unit    |
|--|--|-------|-----------------|----------------|---------|
| <b>SENSOR INPUT</b>  | Each axis  |       |                 |                |         |
| Measurement Range  | User selectable  |       | ±2, ±4, ±8, ±16 |                | g       |
| Nonlinearity   | Percentage of full scale                               |       | ±0.5            |                | %       |
| Inter-Axis Alignment Error   |  |       | ±0.1            |                | Degrees |
| Cross-Axis Sensitivity <sup>2</sup>                                    |  |       | ±1              |                | %       |
| <b>OUTPUT RESOLUTION</b>   | Each axis  |       |                 |                |         |
| All g Ranges   | 10-bit resolution                                      |       | 10              |                | Bits    |
| ±2 g Range   | Full resolution  |       | 10              |                | Bits    |
| ±4 g Range   | Full resolution  |       | 11              |                | Bits    |
| ±8 g Range   | Full resolution  |       | 12              |                | Bits    |
| ±16 g Range  | Full resolution  |       | 13              |                | Bits    |
| <b>SENSITIVITY</b>   | Each axis  |       |                 |                |         |
| Sensitivity at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub>  | ±2 g, 10-bit or full resolution                        | 232   | 256             | 286            | LSB/g   |
| Scale Factor at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub> | ±2 g, 10-bit or full resolution                        | 3.5   | 3.9             | 4.3            | mg/LSB  |
| Sensitivity at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub>  | ±4 g, 10-bit resolution                                | 116   | 128             | 143            | LSB/g   |
| Scale Factor at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub> | ±4 g, 10-bit resolution                                | 7.0   | 7.8             | 8.6            | mg/LSB  |
| Sensitivity at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub>  | ±8 g, 10-bit resolution                                | 58    | 64              | 71             | LSB/g   |
| Scale Factor at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub> | ±8 g, 10-bit resolution                                | 14.0  | 15.6            | 17.2           | mg/LSB  |
| Sensitivity at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub>  | ±16 g, 10-bit resolution                               | 29    | 32              | 36             | LSB/g   |
| Scale Factor at X <sub>OUT</sub> , Y <sub>OUT</sub> , Z <sub>OUT</sub> | ±16 g, 10-bit resolution                               | 28.1  | 31.2            | 34.3           | mg/LSB  |
| Sensitivity Change Due to Temperature                                  |  |       | ±0.01           |                | %/°C    |
| <b>0 g BIAS LEVEL</b>  | Each axis  |       |                 |                |         |
| 0 g Output for X <sub>OUT</sub> , Y <sub>OUT</sub>                     |  | -150  | ±40             | +150           | mg      |
| 0 g Output for Z <sub>OUT</sub>  |  | -250  | ±80             | +250           | mg      |
| 0 g Offset vs. Temperature for x-, y-Axes                              |  |       | ±0.8            |                | mg/°C   |
| 0 g Offset vs. Temperature for z-Axis                                  |  |       | ±4.5            |                | mg/°C   |
| <b>NOISE PERFORMANCE</b>   |  |       |                 |                |         |
| Noise (x-, y-Axes)   | Data rate = 100 Hz for ±2 g, 10-bit or full resolution |       | <1.0            |                | LSB rms |
| Noise (z-Axis)   | Data rate = 100 Hz for ±2 g, 10-bit or full resolution |       | <1.5            |                | LSB rms |
| <b>OUTPUT DATA RATE AND BANDWIDTH</b>                                  | User selectable  |       |                 |                |         |
| Measurement Rate <sup>3</sup>  |  | 6.25  |                 | 3200           | Hz      |
| <b>SELF-TEST<sup>4</sup></b>   | Data rate ≥ 100 Hz, 2.0 V ≤ V <sub>S</sub> ≤ 3.6 V     |       |                 |                |         |
| Output Change in x-Axis  |  | 0.20  |                 | 2.10           | g       |
| Output Change in y-Axis  |  | -2.10 |                 | -0.20          | g       |
| Output Change in z-Axis  |  | 0.30  |                 | 3.40           | g       |
| <b>POWER SUPPLY</b>  |  |       |                 |                |         |
| Operating Voltage Range (V <sub>S</sub> )                              |  | 2.0   | 2.5             | 3.6            | V       |
| Interface Voltage Range (V <sub>DDIO</sub> )                           | V <sub>S</sub> ≤ 2.5 V                                 | 1.7   | 1.8             | V <sub>S</sub> | V       |
|  | V <sub>S</sub> ≥ 2.5 V                                 | 2.0   | 2.5             | V <sub>S</sub> | V       |
| Supply Current   | Data rate > 100 Hz                                     |       | 145             |                | µA      |
|  | Data rate < 10 Hz                                      |       | 40              |                | µA      |
| Standby Mode Leakage Current   |  |       | 0.1             | 2              | µA      |
| Turn-On Time <sup>5</sup>  | Data rate = 3200 Hz                                    |       | 1.4             |                | ms      |
| <b>TEMPERATURE</b>   |  |       |                 |                |         |
| Operating Temperature Range  |  | -40   |                 | +85            | °C      |
| <b>WEIGHT</b>  |  |       |                 |                |         |
| Device Weight  |  |       | 20              |                | mg      |

## ADXL345

### ABSOLUTE MAXIMUM RATINGS

Table 2.

| Parameter  | Rating  |
|--|---|
| Acceleration   |   |
| Any Axis, Unpowered                                  | 10,000 g  |
| Any Axis, Powered                                    | 10,000 g  |
| $V_s$  | -0.3 V to +3.6 V  |
| $V_{DDIO}$   | -0.3 V to +3.6 V  |
| Digital Pins   | -0.3 V to $V_{DDIO} + 0.3$ V or<br>3.6 V, whichever is less |
| All Other Pins                                       | -0.3 V to +3.6 V  |
| Output Short-Circuit Duration<br>(Any Pin to Ground) | Indefinite  |
| Temperature Range                                    |   |
| Powered  | -40°C to +105°C   |
| Storage  | -40°C to +105°C   |

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

### THERMAL RESISTANCE

Table 3. Package Characteristics

| Package Type    | $\theta_{JA}$ | $\theta_{JC}$ | Device Weight |
|-----------------|---------------|---------------|---------------|
| 14-Terminal LGA | 150°C/W       | 85°C/W        | 20 mg         |

### ESD CAUTION



**ESD (electrostatic discharge) sensitive device.** Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

## PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

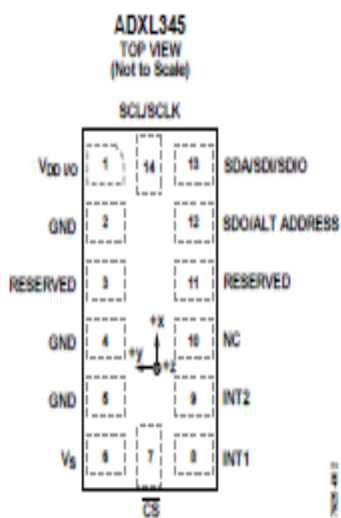


Figure 2. Pin Configuration

Table 4. Pin Function Descriptions

| Pin No. | Mnemonic        | Description  |
|---------|-----------------|--|
| 1       | VDDIO           | Digital Interface Supply Voltage.  |
| 2       | GND             | Must be connected to ground.   |
| 3       | Reserved        | Reserved. This pin must be connected to VS or left open.   |
| 4       | GND             | Must be connected to ground.   |
| 5       | GND             | Must be connected to ground.   |
| 6       | VS              | Supply Voltage.  |
| 7       | CS              | Chip Select.   |
| 8       | INT1            | Interrupt 1 Output.  |
| 9       | INT2            | Interrupt 2 Output.  |
| 10      | NC              | Not Internally Connected.  |
| 11      | Reserved        | Reserved. This pin must be connected to ground or left open.   |
| 12      | SDO/ALT ADDRESS | Serial Data Output/Alternate I <sup>2</sup> C Address Select.  |
| 13      | SDA/SDI/SDIO    | Serial Data (I <sup>2</sup> C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire). |
| 14      | SCL/SCLK        | Serial Communications Clock.   |

## I<sup>2</sup>C

With  $\overline{CS}$  tied high to  $V_{DDIO}$ , the ADXL345 is in I<sup>2</sup>C mode, requiring a simple 2-wire connection as shown in Figure 8. The ADXL345 conforms to the *UM10204 I<sup>2</sup>C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the timing parameters given in Table 11 and Figure 10 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 9. With the SDO/ALT ADDRESS pin high, the 7-bit I<sup>2</sup>C address for the device is 0x1D, followed by the R/W bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I<sup>2</sup>C address of 0x53 (followed by the R/W bit) can be chosen by grounding the SDO/ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

If other devices are connected to the same I<sup>2</sup>C bus, the nominal operating voltage level of these other devices cannot exceed  $V_{DDIO}$  by more than 0.3 V. External pull-up resistors,  $R_p$ , are necessary for proper I<sup>2</sup>C operation. Refer to the *UM10204 I<sup>2</sup>C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 10. I<sup>2</sup>C Digital Input/Output Voltage

| Parameter  | Limit <sup>1</sup>     | Unit  |
|--|------------------------|-------|
| Digital Input Voltage                              |                        |       |
| Low Level Input Voltage ( $V_{IL}$ )               | $0.25 \times V_{DDIO}$ | V max |
| High Level Input Voltage ( $V_{IH}$ )              | $0.75 \times V_{DDIO}$ | V min |
| Digital Output Voltage                             |                        |       |
| Low Level Output Voltage ( $V_{OL}$ ) <sup>2</sup> | $0.2 \times V_{DDIO}$  | V max |

<sup>1</sup> Limits based on characterization results; not production tested.

<sup>2</sup> The limit given is only for  $V_{DDIO} < 2$  V. When  $V_{DDIO} > 2$  V, the limit is 0.4 V max.

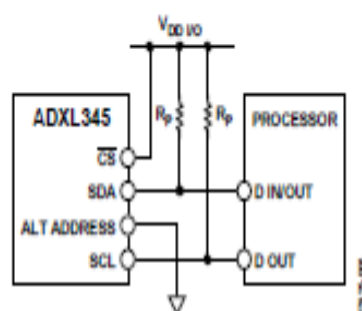
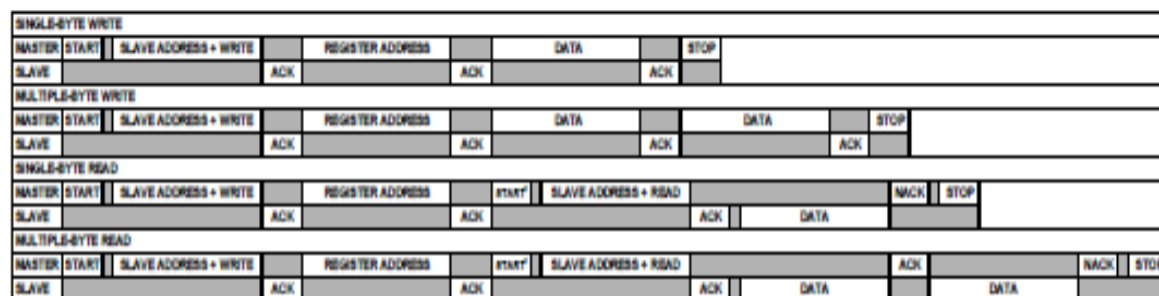


Figure 8. I<sup>2</sup>C Connection Diagram (Address 0x53)



<sup>1</sup>THIS START IS EITHER A RESTART OR A STOP FOLLOWED BY A START.

### NOTES

1. ALL DATA VALUES ARE IN DECIMAL UNLESS OTHERWISE SPECIFIED. UNLESS OTHERWISE NOTED, ALL VALUES ARE IN MICROSECONDS.

REV. 10/08

## APPENDIX 5:

### General description

The BMP085 is a high-precision, ultra-low power barometric pressure sensor for use in advanced mobile applications.

With an absolute accuracy of 2.5hPa and a noise level of down to 0.03hPa (which is equivalent to an altitude change of merely 0.25m) the BMP085 offers superior performance. At the same time the BMP085 features ultra low power consumption of down to 3µA. This and the very small, ultra-thin package make the BMP085 the sensor of choice for any mobile application requiring precise barometric pressure measurement, like for example mobile phones, PDAs, personal GPS-based navigation devices and advanced outdoor equipment.

The BMP085 sensor is based on piezo-resistive MEMS technology for EMC robustness, high accuracy and linearity as well as long term stability. It comes in an ultra-thin, but robust 8-pin ceramic lead-less chip carrier (LCC) package. The BMP085 is designed to be connected directly to a micro-controller of a mobile device via the I<sup>2</sup>C bus.

The BMP085 is the direct successor of the successful SMD500 pressure sensor, that marked a new generation of high-precision digital pressure sensors for consumer applications.

### Key features BMP085

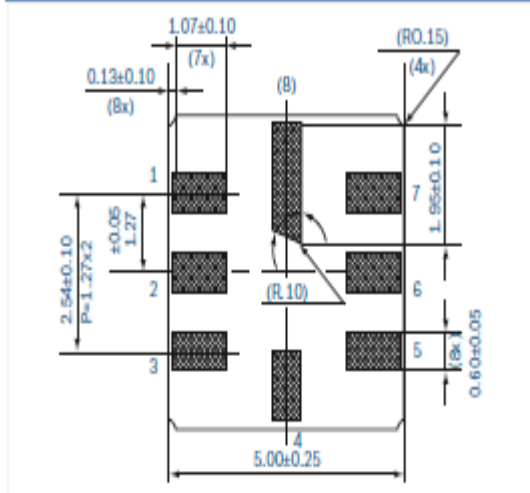
- ▶ Wide barometric pressure range
- ▶ Flexible supply voltage range
- ▶ Ultra-low power consumption
- ▶ Low noise measurement
- ▶ Fully calibrated
- ▶ Temperature measurement included
- ▶ Digital two-wire I<sup>2</sup>C interface
- ▶ Ultra-flat, small footprint ceramic package
- ▶ Pb-free and RoHS compliant

### BMP085 target applications

- ▶ GPS navigation enhancement
- ▶ Dead reckoning
- ▶ In- and outdoor navigation
- ▶ Leisure, sports and health monitoring
- ▶ Weather forecast
- ▶ Vertical velocity indication (rise/sink speed)
- ▶ Fan power control

| Technical data  | BMP085  |
|---|---|
| Pressure range  | 300 ... 1100 hPa                                      |
| RMS noise expressed in pressure   | 0.06 hPa, typ.<br>(ultra-low power mode)              |
|   | 0.03 hPa, typ.<br>(ultra-high resolution mode)        |
| RMS noise expressed in altitude   | 0.5 m, typ.<br>(ultra-low power mode)                 |
|   | 0.25 m, typ.<br>(ultra-high resolution mode)          |
| Absolute accuracy<br>p=700 ... 1100hPa<br>(T=0 ... +65°C, V <sub>DDA</sub> =3.3V) | Pressure: ± 2.5 hPa, max.<br>Temperature: ±2 °C, max. |
| Average current consumption<br>(1Hz data refresh rate)                            | 3 µA, typ.<br>(ultra-low power mode)                  |
|   | 12 µA, typ.<br>(ultra-high resolution mode)           |
| Peak current  | 600 µA, typ.  |
| Stand-by current  | 0.1 µA, typ.  |
| Supply voltage V <sub>DDO</sub>   | 1.62 ... 3.6 V  |
| Supply voltage V <sub>DDA</sub>   | 1.8 ... 3.6 V   |
| Operation temp. range   | -40 ... +85 °C  |
| full accuracy   | 0 ... +65 °C  |
| Pressure conv. time   | 7.5 msec, max.  |
| I <sup>2</sup> C data transfer rate   | 3.4 MHz, max.   |
| Package type / pin no.  | LCC / 8   |
| Package dimensions  | 5 mm x 5 mm x 1.2 mm                                  |

Pin configuration (top view, pads shown transparently)



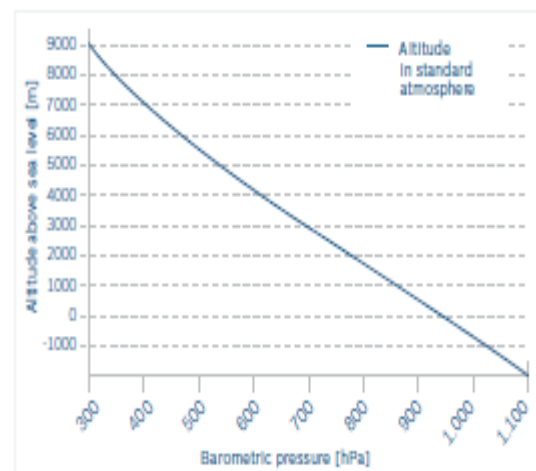
Package dimensions in mm

| Pin | Name             | Function                                |
|-----|------------------|---|
| 1   | GND              | Ground                                  |
| 2   | EOC              | End of conversion output                |
| 3   | V <sub>DDA</sub> | Power supply analog                     |
| 4   | V <sub>DD</sub>  | Power supply digital                    |
| 5   | NC               | Not connected                           |
| 6   | SCL              | I <sup>2</sup> C serial bus clock input |
| 7   | SDA              | I <sup>2</sup> C serial bus data        |
| 8   | XCLR             | Master clear input                      |

### Sensor operation

The BMP085 comes as fully calibrated, ready-to-use sensor module without the need for additional external circuitry. Pressure and temperature data are provided as 16 bit values via the I<sup>2</sup>C interface, together with calibration data for temperature compensation.

The below figure describes the so-called barometric formula by means of which the absolute altitude can be calculated from the measured barometric pressure.



Bosch is the world market leader in MEMS sensors. The BMP085 offers this high experience and reliability for consumer applications. Bosch Sensortec is a subsidiary of Bosch that focuses on micromechanical components for the non-automotive markets.

Please contact us for further details. We are happy to provide you with more information.

