A
Dissertation
on

Design & Development of Stabilised Platform for Indian Naval ships using MEMS Sensors

submitted in partial fulfilment of the requirements
for the award of degree of

Master of Technology

in

VLSI Design and Embedded System

Submitted
By

Lt Cdr Rajat Agarwal
University Roll Number- 2K12/VLS/16

Under the guidance of

Dr. S Indu
Professor,
Department of Electronics and Communication Engineering
Delhi Technological University

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY

Electronics and Communication Engineering Department
Delhi Technological University
Delhi-110042

CERTIFICATE

This is to certify that the dissertation titled "**Design & Development of Stabilised Platform for Indian Naval ships using MEMS Sensors**" is a bonafide record of work done by **Lt Cdr Rajat Agarwal, Roll No. 2K12/VLS/16** at **Delhi Technological University** for partial fulfilment of the requirements for the degree of Master of Technology in VLSI Design and Embedded System. This project was carried out under my supervision and has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

Date: _____

(Dr. S Indu)
Professor
Department of Electronics and Communication Engineering
Delhi Technological University

ACKNOWLEDGEMENTS

I would like to express my deep sense of respect and gratitude to my project supervisor **Dr. S Indu,** Professor, Electronics and Communication Engineering Department, DTU for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement she provided without which the project could not have been a success.

I am also grateful to **Prof. P R Chadha**, HOD, Electronics and Communication Engineering Department, DTU for his immense support.

I would also like to acknowledge Delhi Technological University for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my colleagues in service in helping choosing this topic and constantly encouraging me during the completion of work.

(Lt Cdr Rajat Agarwal)
University Roll no: 2K12/VLS/16
M.Tech (VLSI Design and Embedded System)
Department of Electronics & Communication Engineering
Delhi Technological University
Delhi – 110042

ABSTRACT

The use of roll and pitch stabilised platforms onboard ships, aircrafts and submarines is a common practice so as to ensure common plane of operation for all systems in a three dimensional space. It is to ensure that their performance is not affected by movement of vessel due to various undesirable forces acting on it. While weapon systems require a higher degree of accuracy and precision in stabilisers, there are other systems where accuracy and precision of stabilisation can be compromised to a certain extent. Some such systems are communication antennas, TV satellite dish, solar panel etc. installed on ships and aircrafts. These systems, if stabilised over a platform using some economic system, then accuracy can be compromised to certain extent. Present thesis deals with design and development of similar low cost MEMS sensor based roll, pitch stabilised platform. It uses ADXL345 accelerometer sensor which senses raw acceleration values in all three axes in terms of acceleration due to gravity (g) and transmits to a controller which converts the acceleration into roll and pitch values. The roll and pitch is further transformed into negative feedback and given to servo motors connected to a platform that is free to rotate in x and y axes. Thus, movement of X and Y planes of platform in the direction opposite to roll and pitch experienced by the ship helps in realising a stabilised platform which can be employed to

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

INS	Inertial Navigation System
SINS	Strap down Inertial Navigation System
MEMS	Micro Electro-Mechanical Systems
I^2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
GUI	Graphical User Interface
UART	Universal Asynchronous Receiver Transmitter

CHAPTER 1
INTRODUCTION

1.1 Motivation

Navigation has been an interesting area of Research & Development from centuries. It is a useful subject appealing to people ranging from a ship's captain to an aircraft pilot to an astronaut and to a car driver. But, the accuracy and criticality of navigation data varies from individual to individual and so do the cost and components involved in building a complete navigation system. Unlike navigation on surface, there are various problems faced when we navigate at sea or in an aircraft. One such problem is of roll, pitch and yaw which are experienced while onboard any of such platforms.

1.2 Navigation Systems

Few commonly known tools for navigation onboard ships, submarines and aircrafts are RADAR, SONAR, Gyroscopes, Magnetic Compass, Global Positioning System (GPS) and ECDIS (Electronic Chart Display System). All these systems have their own expertise and limitations. While RADAR & SONAR emit a lot of power to sense distant objects, Magnetic compass is affected by surrounding magnetic fields due to ship's or air craft's metallic body and other electrical equipments onboard.

Figure 1.1 Ship's mast containing RADAR and communication antennas

GPS relies on continuous line of sight communication with Geo stationary satellites. So, none of the navigation systems described are self sufficient in providing correct position of the travelling body. Moreover, war ships and military aircrafts have to operate in stealth mode so they cannot emit energy at all times. Submarines operate under water and therefore they cannot receive electromagnetic GPS signals. Hence, there exists a need for self sufficient Navigation system which could calculate its position using some inbuilt hardware and a software algorithm without emitting any energy. '*Necessity is the mother of invention*'; this has led to development of Inertial Navigation System (INS) which is discussed in this thesis and an application derived from sensors involved in INS is developed and explained in detail.

1.3 Inertial Navigation Systems

Inertial navigation is an arrangement designed to offer navigation information using direct measurements of acceleration, without any external measurements. Before the advent of global positioning systems, such as the United State's GPS or the EU's Galileo system, inertial navigation was relied upon to provide accurate position data for a number of vehicles, including guided missiles, aircraft, submarines, and spacecraft.

SINS (Strap down inertial navigation system) is a type of inertial navigation system where the sensor package is simply strapped to the vehicle. Instead of a gimballed platform, small sensors called "rate gyroscopes" are used to measure the angular acceleration. These sensors work in a similar fashion to the linear accelerometers, in that they provide an instantaneous reading of the angular acceleration. This value must be recorded and integrated over time to calculate the vehicle's orientation, and is subject to the same accuracy and sample frequency problems as the linear acceleration sensors.

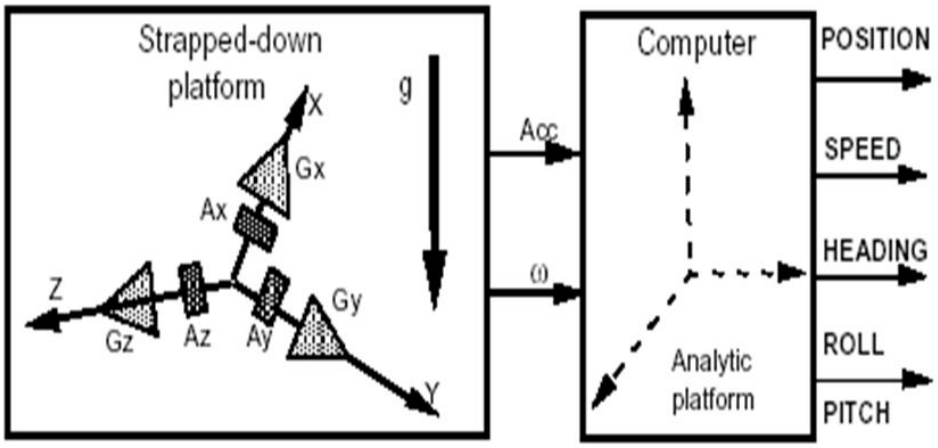

Figure 1.2 Strap down Inertial Navigation System

1.4 Why Inertial? (Naïve concept)

The concept of finding one's position lies in Newton's basic laws of inertia and motion. Newton's Second law of motion states that

Force (F) = Mass (m) x Acceleration (a)

So, if a body of mass (m) is moving under influence of some force (F), its acceleration (a) can be calculated using above equation easily.

Further, the acceleration (a) can be integrated over time (t) to calculate velocity (v) which can be further integrated over time (t) to calculate Distance (s).

$$\mathbf{s} = \iint_0^t a.\,dt$$

Thus, if the initial position of an object is known, current position of the object can be calculated by adding the distance to it. Although this is very naïve concept and requires a lot of other concepts, components and processing before developing into a self sufficient Inertial Navigation System (INS) moving freely in 3-dimensional space as explained ahead in this thesis.

1.5 Objective and scope of thesis

There are various algorithms to calculate Position of object as described in [2]. This thesis is limited to understanding the concept and developing a prototype of Inertial Navigation System by extracting Accelerometer data over I^2C protocol and transporting the data packet over UART protocol to the Computer where roll and pitch are calculated using either of the available algorithms. The accuracy of angular movement calculated entirely depends on algorithm used for computing. Although, the thesis discusses Inertial Navigation systems, the project is limited to developing a Roll/Pitch stabilised platform for use onboard ships and aircrafts using raw accelerometer data obtained from ADXL345 module.

The prototype is realised using ATMEGA 328 micro-controller over Arduino based open source platform. It is a faster way of converting ideas into prototypes. A hardware schematic for designing complete systems using ATMEGA 328 micro-controller without Arduino board is also described in thesis. Complete tool-chain of data acquisition and graphical display of information is developed using various open source software. Complete model is then realised using a micro-controller board interfaced to ADXL345 module interfaced over I^2C protocol on input side and servo motors controlling the X and Y planes of a platform at output side. The platform with two degrees of freedom is also built using fused deposition modelling (FDM) technique.

CHAPTER 2

NEED FOR STABILISED PLATFORMS ONBOARD IN SHIPS

2.1 Roll, Pitch and Yaw

Roll, pitch and yaw are three movements possible for an object in a three dimensional space. An aircraft, a ship or a submarine moving in a three dimensional space can experience either of these movements across its three axes.

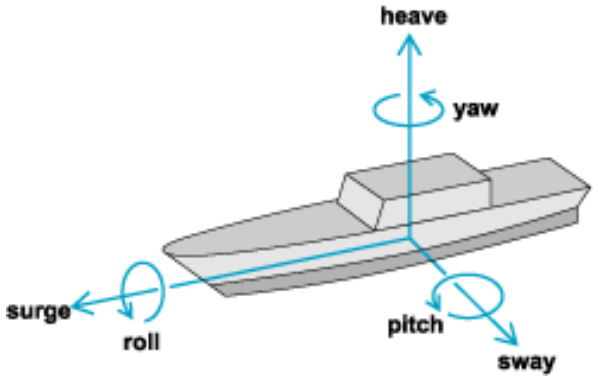

Figure 2.1 Roll, Pitch and yaw experienced by a ship

Roll is defined as oscillatory angular movement of object across x axis. Pitch is defined as oscillatory angular movement of object across y axis and Yaw refers to oscillatory angular movement of object across Z axis in a three dimensional space. It can be experienced by any object travelling in a three dimensional space such as ship, submarine, air craft or missile. While it is desirable to have these movements in aircrafts or missiles for manoeuvring them, they are undesirable on a platform like ship and submarine as it affect the stability of the vessel.

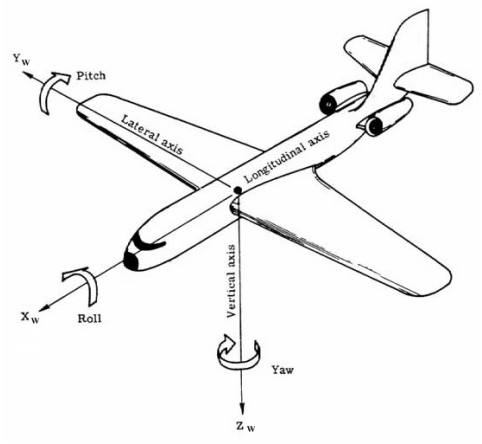

Figure 2.2 Roll, Pitch and yaw experienced by an Aircraft

2.2 Effect of Roll, pitch and yaw on ship's systems

The roll, pitch and yaw movements affect many of the ship's systems adversely. While, the stability of a ship is greatly affected by roll and pitch movement and may be dangerous for stability of big oil tankers, it can be fatal for warships. The effect of roll and pitch can be explained by taking an example of a gun mount pointing towards a target and ready to fire.

The ship being in open sea may experience roll or pitch due to sea waves just before firing through the gun mount and it may be fatal as the gun may hit its own surface or men. Hence, it is very important for a naval ship to counter the effect of Roll, pitch and yaw on regular basis, especially for weapon platforms.

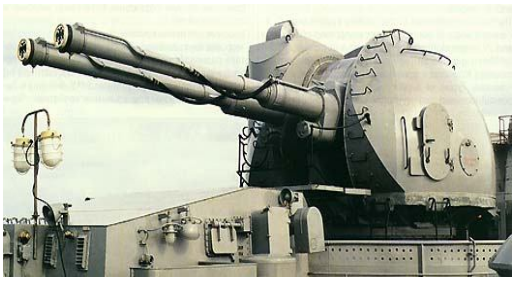

Figure 2.3 Gun mount locked on target

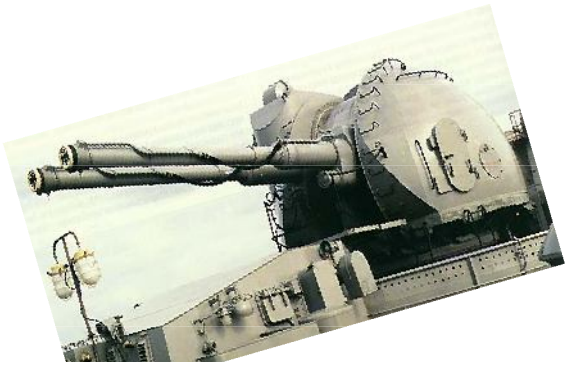

Figure 2.4 Direction of gun mount after ship experiencing Roll / Pitch

2.3 Concept of stabilised platforms onboard Naval ships

As shown in figures 2.3 and 2.4 above, the erratic movement of gun mounts and other weapons onboard ship due to roll and pitch during firing operations may be critical for its smooth operation. There exists a need for some setup that would constantly compute the error occurred in the original orientation of gun mount due to roll or pitch movement and apply the equal torque in opposite direction so as to keep the mount stable with respect to original position. Similarly, there are many other systems onboard ship whose performance is affected by roll, pitch and yaw movements. Some examples of such ship's systems are:-

(a) Communication antennas
(b) Satellite TV dish
(c) Solar panels

Figure 2.5 Satellite TV dish onboard Naval ship

However, unlike weapon platforms, above systems do not require very precise correction in roll and pitch. The performance is not affected much if some economic solution for stabilising the platform containing above systems is used.

The current thesis tries to achieve this goal by computing the acceleration across x and y axes of the ship's surface using ADXL345 sensor module and converting it into equivalent roll and pitch values using algorithm suggested in [1]. The acceleration data thus obtained is given to a micro-controller which computes equivalent roll and pitch values. The micro-controller further

computes equal and opposite torque to be applied to stabilise the platform. The angular position thus calculated is fed to servo motors interfaced to micro-controller for movement in opposite direction.

CHAPTER 3

LOW COST MEMS SENSORS FOR SENSING ROLL AND PITCH

3.1 MEMS sensor ADXL345

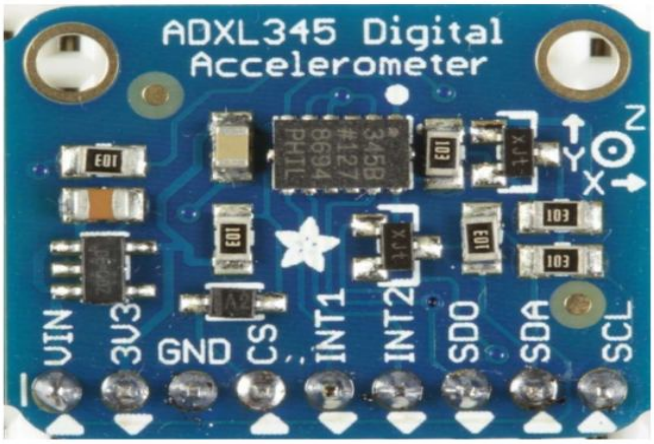

Figure 3.1 ADXL345 Accelerometer module

MEMS stands for Micro-Electro-Mechanical Systems which integrates mechanical units and electronic components together through micro-fabrication technology at the sub-millimeter scale. With this technology, we can build microstructures through micro-machining and create sensors which are very small in size. In this project we use MEMS based 3 dimensional Accelerometer sensor ADXL345. MEMS sensor ADXL345 senses acceleration in all three axes in terms of acceleration due to gravity and transmits the data over serial protocol.

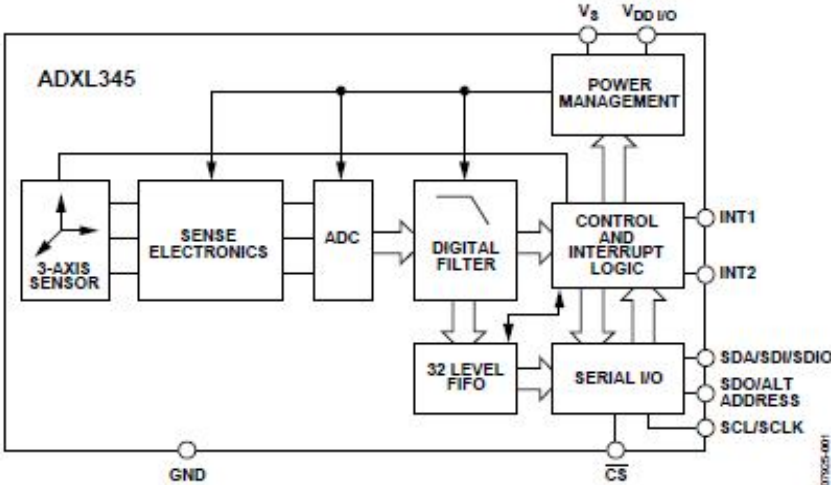

Figure 3.2 Functional block diagram of ADXL345 module

As shown in above schematic, the 3-axis MEMS sensor is embedded into the ADXL345 module along with associated blocks for conversion of analog data to digital data, a digital filter and a serial I/O module. The data obtained out of the above module is transferred to micro-controller either using SPI (Serial Peripheral Interface) or I^2C protocol.

ADXL345 supports four ranges of sensitivity from +/- 2G to +/-16G. It supports output data ranges from 100Hz to 3200 Hz. The sensor module consists of micro machined structure on a silicon wafer. The structure is suspended by poly-silicon springs which allow it to deflect in any direction smoothly when subject to acceleration in X, Y or Z axis. The deflection causes a change in capacitance between fixed plates and plates attached to the suspended structure. The change in capacitance is proportional to acceleration caused in X, Y and Z axes which is converted in to equivalent voltage and given out of MEMS sensor. The voltage data is then given to ADC module onboard ADXL345 to convert it into equivalent digital data.

The digital data obtained as raw acceleration for each of the X, Y and Z axes is in 16 bit format. The data is thus stored in two registers of eight bit each. Thus, every data read operation from ADXL345 module must request for six bytes of data.

The data can be extracted from ADXL345 module using I^2C protocol by wiring as in below table.

ADXL345	MICRO-CONTROLLER BOARD
GND	GND
Vin	+5V
SDA	SDA
SCL	SCL

Table 3.1 Connection between ADXL345 and Microcontroller board

Actual hardware developed for the project to test the module is as below. Connections are made as per table 3.1 above.

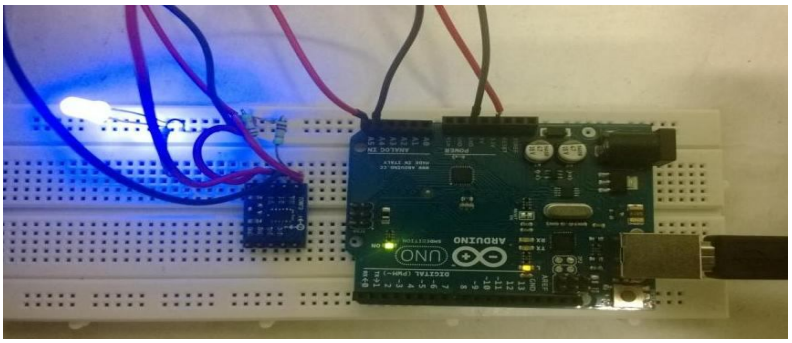

Figure 3.3 ADXL345 module connected to micro-controller board over I^2C protocol

3.2 Roll and Pitch Estimation Algorithms

The acceleration sensed using ADXL345 module needs to be converted into corresponding Roll and pitch values. This requires some complex mathematical algorithms described in [1]. The ADXL345 module can be configured for various ranges (*±2g, ±4g, ±8g, ±16g*) where 1g unit = 9.8 m/s^2. By default, the module gives 10-bit resolution data for *±2g range configuration. The raw accelerometer readings (Raw_{Accel}) can be converted into acceleration using following formula:-*

$$\boldsymbol{G_{Accel}} = Raw_{Accel} \cdot \frac{Range}{2^{Resolution-1}}$$

The raw values of acceleration in x, y and z axes were obtained and plotted over GUI developed using processing software. The results obtained are shown below.

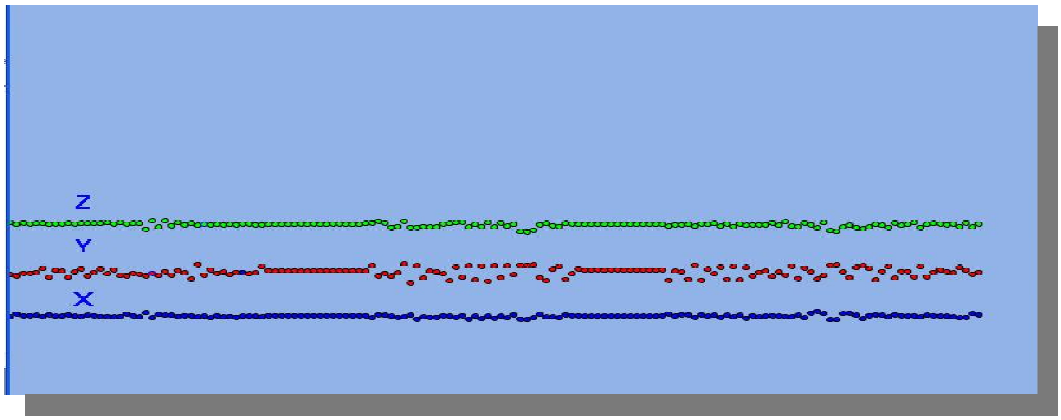

Figure 3.4 Plot of raw X,Y,Z acceleration values as obtained from ADXL345 module

The acceleration in all three axes (G_x, G_y and G_z) are obtained using ADXl345 module using above formula. The components of acceleration are now to be converted into roll and pitch that the platform would have experienced. For, converting the acceleration values to roll and pitch following formulae are used:-

The above roll pitch formulae can be easily converted into an algorithm for writing a computer program for calculation of roll and pitch using a micro-controller. In a C program, *arctan* formula can be replaced with *atan2()* function from <*math.h*> library as shown below:-

Roll = (atan2(-fYg, fZg)*180.0)/M_PI;

Pitch = (atan2(fXg, sqrt(fYg*fYg + fZg*fZg))*180.0)/M_PI;

3.3 Noise reduction using Digital Low pass Filter

A problem is faced in obtaining reliable data from ADXL345 module. The readings for raw acceleration values fluctuate a lot and thus some kind of filtering of data is required before estimating roll and pitch. A simple low pass filter can be employed for filtering noise from data obtained for each axis by passing the stream of data through a difference equation as shown below:-

$$y(n) = x(n) + x(n-1)$$

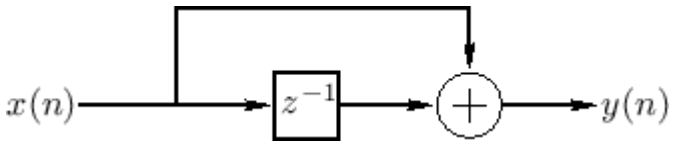

Figure 3.5 Signal flow graph of a digital low pass filter

For realising the above mentioned low pass filter in computer we can employ a simple equation that can be easily implemented using a simple C program.

$$y_t = \alpha . x_t + (1 - \alpha) . y_{t-1}$$

Here y_t is the current filtered value of acceleration while y_{t-1} is the previously filtered value, x_t is the raw acceleration value and α is the smoothening factor. The filtering is to be done to raw acceleration values for each of the three axes before calculation of roll and pitch angles.

The smoothening factor α is to be chosen wisely for proper filtering of data obtained from module. It should not be set too low as we would lose real time behaviour and it will take more time to stabilise.

The calibration of module along with selection of value for α needs to be done before it is actually employed for calculating roll and pitch values.

Data	X(t)	Alpha	y(t-1)	y(t)
1	0.31	0.5	0.28	0.30
2	0.31	0.5	0.30	0.30
3	0.27	0.5	0.30	0.29
4	0.24	0.5	0.29	0.26
5	0.31	0.5	0.26	0.29
6	-2.79	0.5	0.29	-1.25
7	-0.89	0.5	-1.25	-1.07
8	2.5	0.5	-1.07	0.71

Table 3.2 Sample of X axis acceleration values taken for digital low pass filter

The data obtained for x-axis acceleration was plotted for both filtered and non-filtered values as shown in graph below. Raw data is plotted a x(t) while filtered data is plotted as y(t). It is clearly evident that acceleration data after passing through low pass filter is gradually changing its value compared to non-filtered data which is abruptly changing value. Hence, it is better to use filter over raw data.

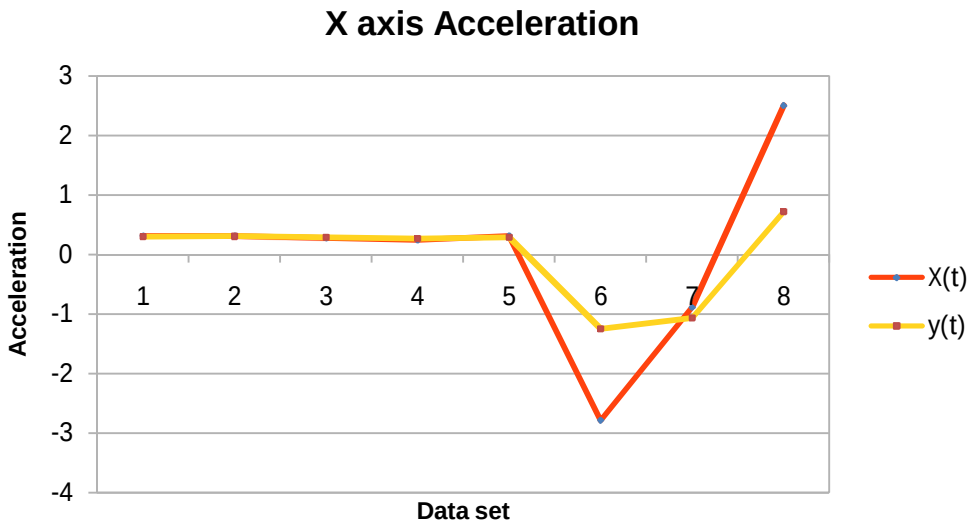

Figure 3.6 Plot of raw [x(t)] and filtered [y(t)] data as obtained from ADXL345 module

The value of α (alpha) chosen in above example is 0.5. However, α can have any value less than 1. It needs to be chosen wisely so as to have best possible smoothening of raw values without losing any considerable amount of data. Various values of α were considered and data obtained was plotted along with original values so as to see the effect of smoothening.

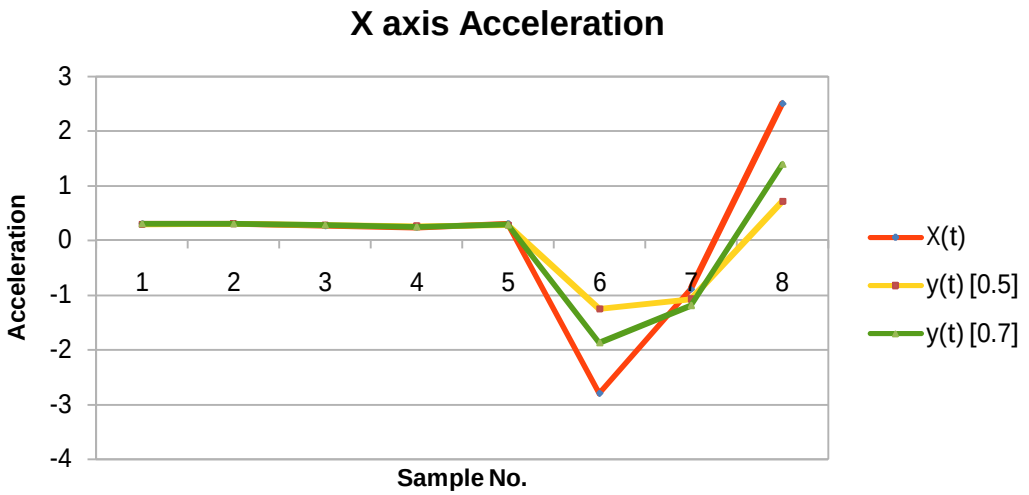

Figure 3.7 Plot of raw [x(t)] and filtered [y(t)] data for two different α values 0.5 and 0.7

Now when the concept of converting raw acceleration data into roll and pitch value is clear, next step is to convert it into an algorithm that can be used to design a computer based automatic system to control planes of a platform to counter roll and pitch. The same is explained in further chapters of this thesis.

CHAPTER 4

EMBEDDED SYSTEM DESIGN AND DEVELOPMENT

4.1 Definition of Embedded Systems

An embedded system is a combination of hardware and software (either fixed in capability or programmable) designed for a (or few) specific purpose. An embedded system maybe an independent system or a part of some bigger system. An embedded system involves combination of electronics (both digital and analog), mechanical parts and software. Various examples of embedded systems are digital watches, MP3 players and elevator control panel. This thesis explains the design and development of an embedded system to be used onboard naval ships and submarines for stabilising its various systems against roll and pitch. Various types of interfaces are explained and the sensor for acceleration input, ADXL345 is interfaced over I^2C protocol. On the output side servo motors are interfaced to micro-controller which are controlled based on input received from accelerometer module. The platform for stabilising the system is also developed as a part of complete system.

4.2 Development environment for Embedded systems

First step for designing any embedded system is to setup a development environment. An embedded system consists of a micro-controller, a micro-processor or an FPGA that needs to be programmed for the specific purpose it is to be used. A development environment consists of following components:-

a) Text Editor
b) Compiler
c) Linker
d) Simulator
e) Programmer or debugger

Nowadays, all the above components are integrated into a single setup and is called Integrated Development Environment (IDE). Based on the micro-controller used for development of embedded system, a specific IDE needs to be setup for developing an application for the desired embedded system.

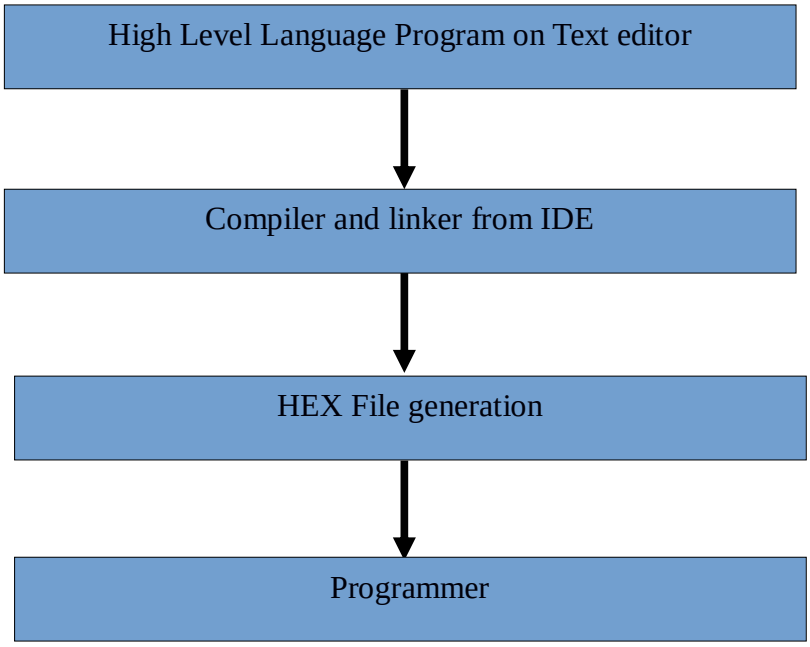

Figure 4.1 Flowchart showing the process of programming an embedded system

4.3 Micro-controller programming

For developing our application, we had vast range of micro-controllers available such as PIC, AVR, Intel 8051. Each micro-controller has its own IDE which can be setup for developing the application. PIC micro-controller requires MPLAB IDE from Mircrochip while AVR requires AVR studio and Intel 8051 can be setup on Keil IDE.

The application program is developed in High Level Language (HLL) which is then compiled using a compiler installed for the specific micro-controller used for designing the embedded system. The IDE generates a hexadecimal (Hex) file which is specific to the micro-controller for which the HLL program was compiled. This process is called *cross compiling*.

The Hex file thus generated is to be burnt on micro-controller's program memory using a programmer. Once the power is switched ON, the application runs continuously in an endless loop and performs the desired function.

A screenshot of MPLAB IDE used for programming PIC micro-controllers is shown in figure below.

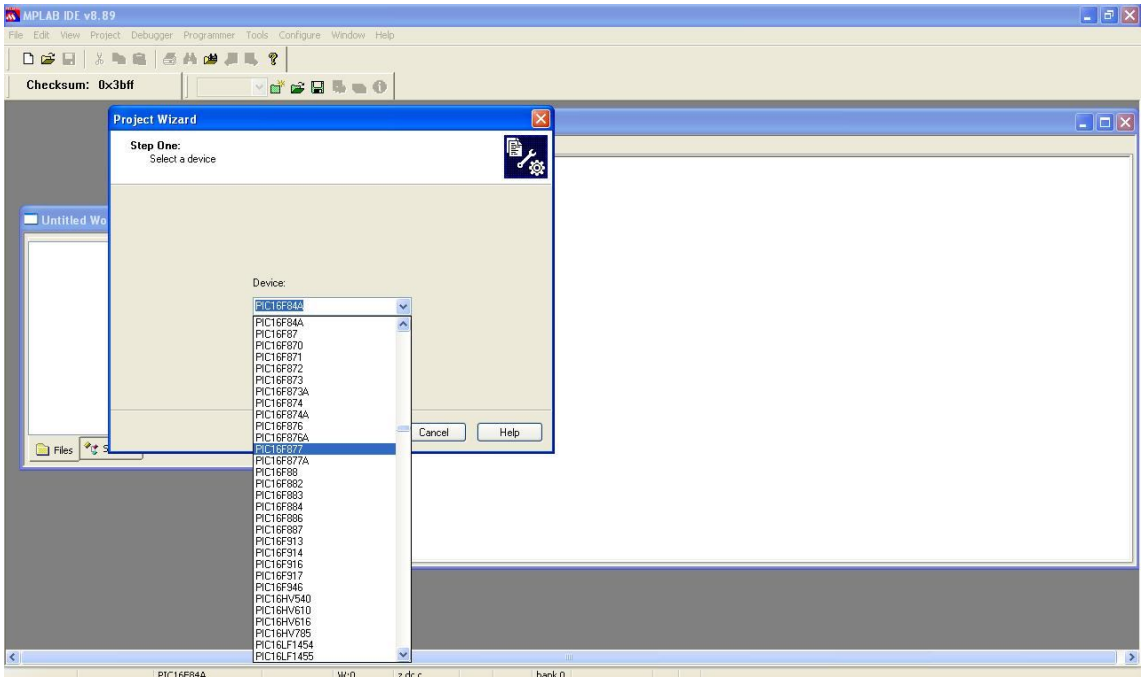

Figure 4.2 MPLAB IDE for programming PIC micro-controllers

4.4 Single Board Computers (SBCs)

For developing our application, we had vast range of micro-controllers available such as PIC, AVR and 8051. However, the choice of AVR micro-controller ATMEGA328 was based on following features that were required for development of the stabilised platform without any external interfaces required for essential components:-

PARAMETERS	VALUE
Flash	32 Kbytes
RAM	2 Kbytes
Pin Count	28
Max. Operating Frequency	20 MHz
CPU	8-bit AVR
A/D converter	10- bit / 6 ch
Max I/O Pins	26
Ext Interrupts	24

Table 4.1 Features of ATMEGA328 micro-controller (as per datasheet)

For development and testing of the prototype suggested in this thesis, various Single board computers (SBCs) were referred such as ARM micro-processor based Beagleboard and Raspberry Pi and AVR micro-controller based Arduino UNO R3.

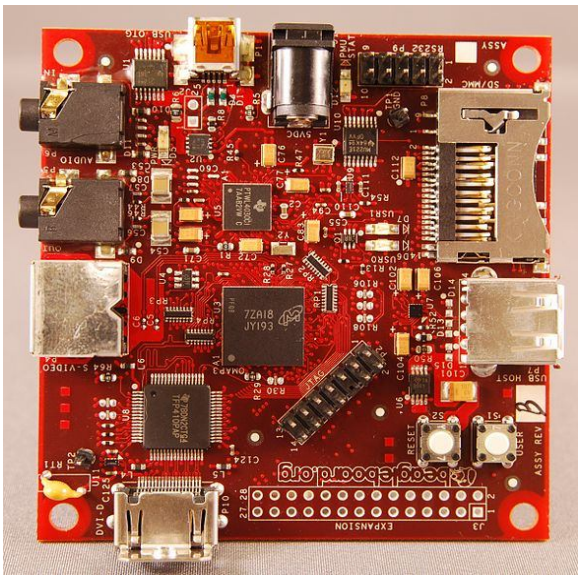

Figure 4.3 Beagleboard (ARM cortex A8 based) SBC

A single-board computer (SBC) is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) and other features required of a functional computer.

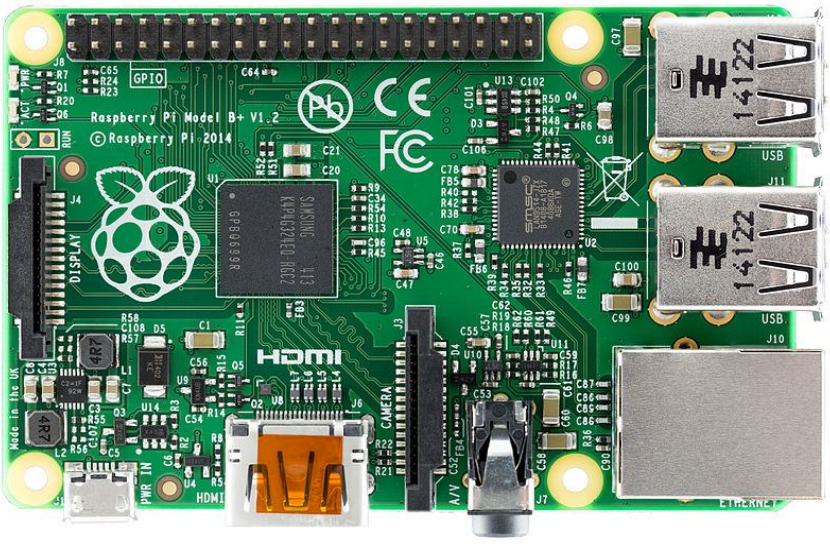

Figure 4.4 Raspberry Pi (ARM cortex A7 based) SBC

The choice of SBC for any prototype development is based on various factors such as economic to use, ease of setup and availability of required interfaces. Being an open source development platform and easy to setup, Arduino based SBC was preferred for development of prototype for the project. However, after testing the system on micro-controller board, a circuit for developing independent system to be implemented onboard Naval ships for actual systems is also suggested later.

Figure 4.5 Arduino UNO R3 development board

Arduino UNO R3 is an ATMEGA328 micro-controller based Single Board Computer. It consists of 8 bit AVR micro-controller with various onboard peripherals and all pins available for programming. It can be programmed using onboard programmer that can be connected to any computer using standard USB. The board draws power from either USB or 5V power adapter. It features the Atmega8U2 microcontroller chip programmed as a USB-to-serial converter.

Other features of Arduino UNO R3 board are:

a) Microcontroller: ATmega328
b) Operating Voltage: 5V
c) Input Voltage (recommended): 7-12V
d) Input Voltage (limits): 6-20V
e) Digital I/O Pins: 14 (of which 6 provide PWM output)
f) Analog Input Pins: 6
g) DC Current per I/O Pin: 40 mA

h) DC Current for 3.3V Pin: 50 mA

i) Flash Memory: 32 KB of which 0.5 KB used by bootloader

j) SRAM: 2 KB (ATmega328)

k) EEPROM: 1 KB (ATmega328)

l) Clock Speed: 16 MHz

CHAPTER 5

SIMULATION OF ROLL PITCH DATA USING ADXL345 SENSOR

5.1 Tool chain for Data Acquisition

To start with, a prototype has been developed over an AVR microcontroller ATMEGA328 based board that collects data from various sources and transmits data packet over UART interface. The data is further converted to USB protocol through a FTDI chip. The data obtained over USB is graphically displayed over a GUI developed on an open source software “Processing”. Hence, this setup prepares a complete chain of acquiring data as well as its graphical representation. Various applications are developed using the data obtained from hardware interface. The complexity of data acquired and processing involved can be increased as we proceed. It would help us later in acquiring data from more complex modules for our project.

A simple hardware for data acquisition console is developed using Arduino UNO R3 board as shown in graphical image in figure 5.1 below.

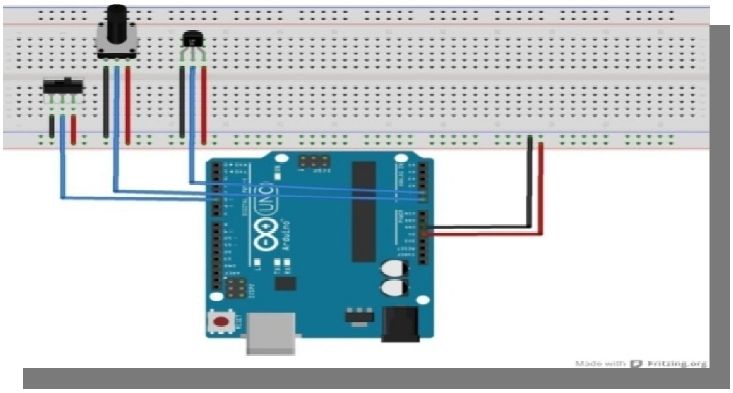

Figure 5.1 Graphical image of Circuit for interfacing Digital switch, Analog POT and LM35 IC interfaced to Arduino Uno R3 board (image developed using Fritzing)

The circuit was then realised using Arduino Uno R3 board and various components such as LM35 temperature sensing IC and potentiometer. The pin diagram of micro-controller ATMEGA 328 is shown in figure 5.2 below

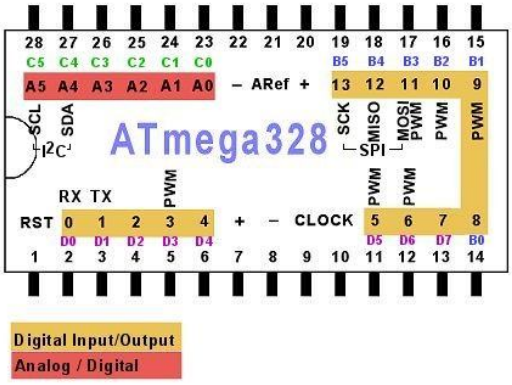

Figure 5.2 Pin diagram of Atmega328 micro-controller

Actual hardware for setting up a data acquisition chain is as shown in figure 5.3 below. All data acquired from analog sensors is fed to micro-controller through ADC (Analog to Digital Convertor). Complete data is then converted into packets and transferred out over serial UART protocol. UART data is then passed through FTDI chip so as to be sent to a computer over USB.

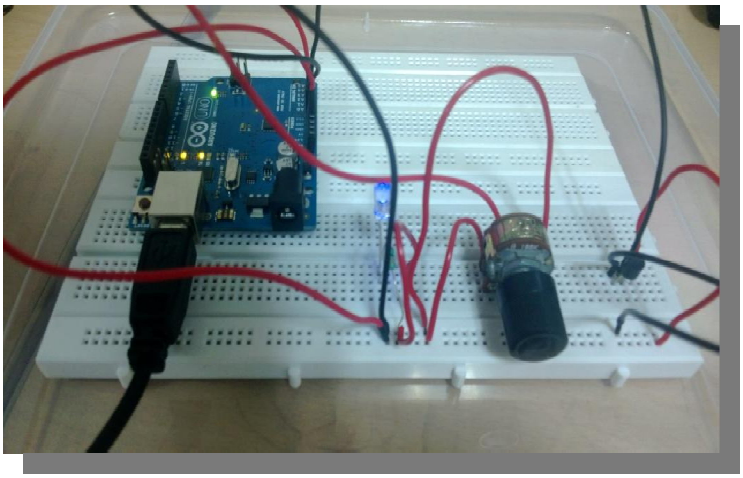

Figure 5.3 Actual hardware setup for Data Acquisition Console

Results obtained from above sensors interfaced to micro-controller board are plotted with respect to time and results are obtained as shown below.

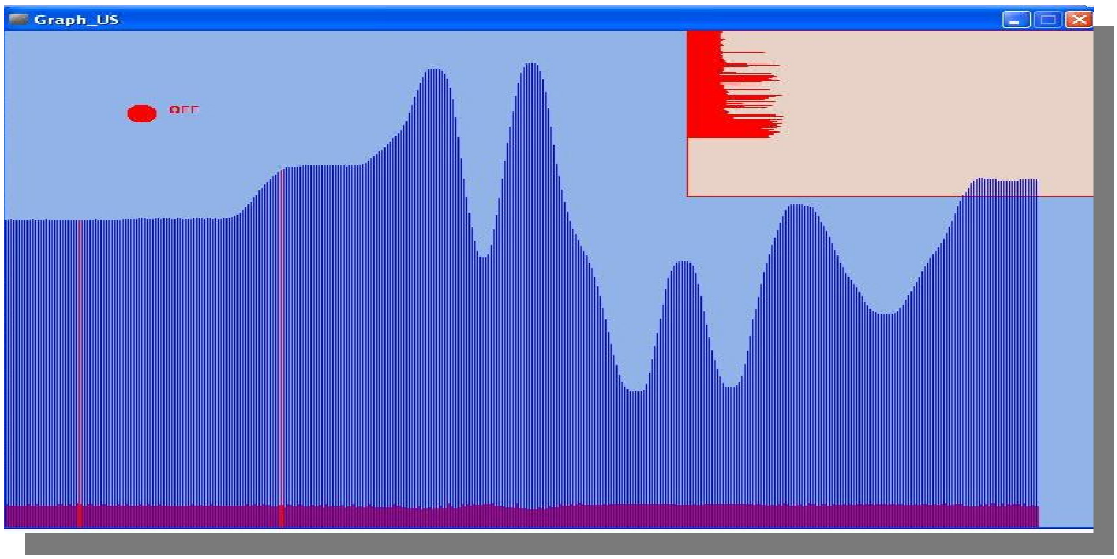

Figure 5.4 Plot of data obtained from sensors interfaced to micro-controller board

Similar model is applied for acquiring data from ADXL345 module which is explained further in this thesis.

5.2 I^2C protocol for data transfer with I^2C scanner

I^2C (Inter Integrated Circuit) popularly known as I-squared-C, is a multiple master, multiple slave, single-ended, serial computer bus designed by Philips Semiconductor. It was primarily used for attaching low-speed peripherals to computer motherboards and other embedded systems. I^2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V

The bus has two roles for nodes: master and slave:

- **Master node** — node that generates the clock and initiates communication with slaves
- **Slave node** — node that receives the clock and responds when addressed by the master

The bus is a multi-master bus which means any number of master nodes can be present. Additionally, master and slave roles may be changed between messages (after a STOP is sent).

There are four possible modes of operation for a given bus device, although most devices only use a single role and its two modes:

- Master transmit — master node is sending data to a slave
- Master receive — master node is receiving data from a slave
- Slave transmit — slave node is sending data to the master
- Slave receive — slave node is receiving data from the master

It uses 2 signal lines i.e. SDA (Serial Data) and SCL (Serial Clock). Inter Integrated circuit communication acts as a link between microcomputer and module to be connected for data acquisition. I^2C defines basic types of messages, each of which begins with a START and ends with a STOP:

- Single message where a master writes data to a slave;
- Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves.

5.2.1 I^2C Scanner

I^2C bus can be used to interface multiple modules using SCL (Serial Clock) and SDA (Serial Data) lines. Each module can be accessed individually using address assigned to it. To test this, an I^2C scanner was developed where few I^2C based devices were connected on same SDA and SCA lines from micro-controller. The devices have their unique device ids and can be accessed by sending address on SDA line. I^2C scanner is developed by running a loop for addresses, once a device is found, it is shown on the screen. The code for I^2C scanner is at Appendix A.1. The result is shown in figure below:-

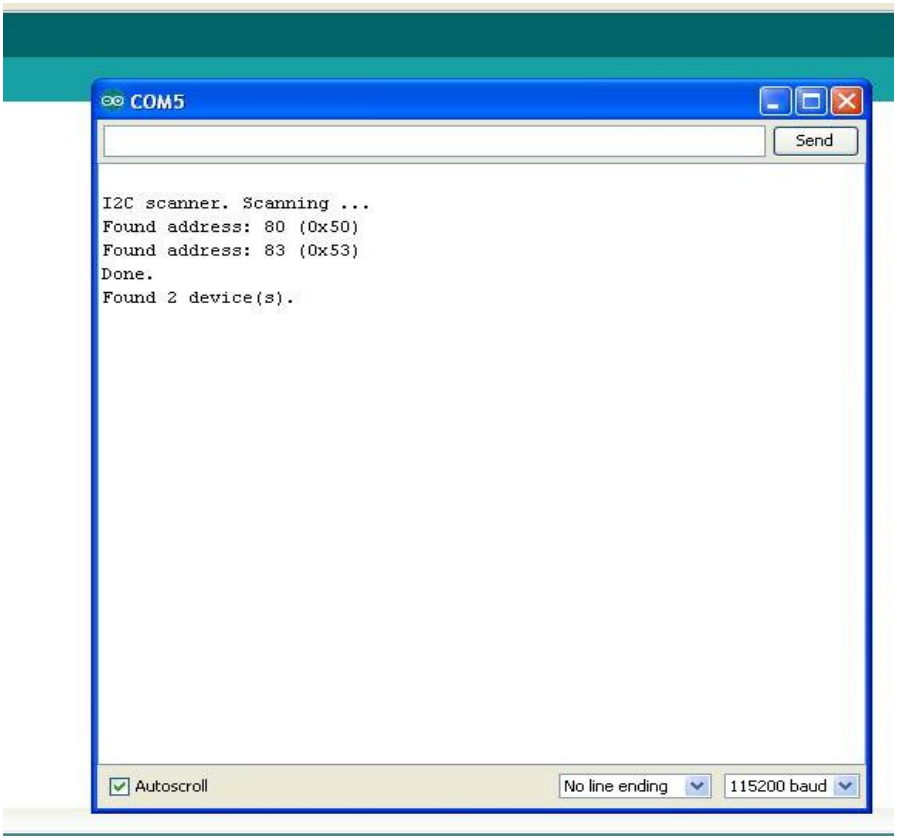

Figure 5.5 I^2C scanner result with two devices connected on bus

5.3 ADXL345 interfaced over I^2C protocol

For getting acceleration data from X and Y axes of the platform that needs to be stablised, an ADXL345 module may be placed parallel to its plane. The module is then interfaced to micro-controller board over I^2C protocol as shown in figure below.

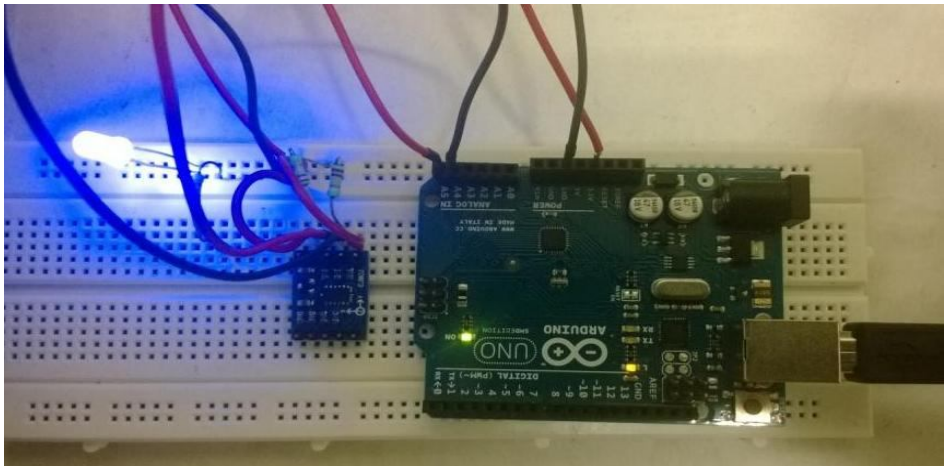

Figure 5.6 Hardware setup for ADXL345 interfaced over I^2C protocol

5.3.1 Connection of ADXL345 over I^2C

With CS tied high to VDD I/O, the ADXL345 is in I^2C mode, requires a simple 2-wire connection, as shown in Figure 5.7. The ADXL345 conforms to the UM10204 I^2C-Bus Specification. It supports standard (100 kHz) and fast (400 kHz) data transfer modes. Single or multiple-byte reads/writes are supported. With the ALT ADDRESS pin high, the 7-bit I^2C 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 address I^2C of 0x53 (followed by the R/W bit) can be chosen by grounding the ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

There are no internal pull-up or pull-down resistors for any unused pins; therefore, there is no known state or default state for the CS or ALT ADDRESS pin if left floating or unconnected. It is required that the CS pin be connected to VDD I/O and that the ALT ADDRESS pin be connected to either VDD I/O or GND when using I^2C.

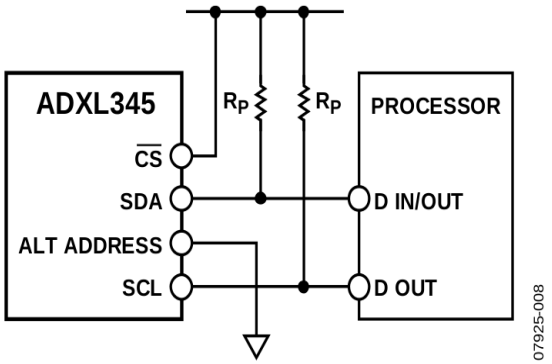

Figure 5.7 Connection diagram for ADXL345 module interfaced over I^2C protocol

Due to communication speed limitations, the maximum output data rate when using 400 kHz I^2C is 800 Hz and scales linearly with a change in the I^2C communication speed. For example, using I^2C at 100 kHz would limit the maximum ODR to 200 Hz. Operation at an output data rate above the recommended maximum may result in undesirable effect on the acceleration data, including missing samples or additional noise.

If other devices are connected to the same I^2C bus, the nominal operating voltage level of these other devices cannot exceed VDD I/O by more than 0.3 V. External pull-up resistors, R_P, are necessary for proper I^2C operation.

REGISTER MAP

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detec
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Figure 5.8 Register map for ADXL345 module

5.4 Development of simulation program

Simulation program is developed for obtaining data from ADXL345 module using following algorithm:-

a) Start communication over I^2C bus.
b) Initialise ADXL345 module.
c) Read raw acceleration values from ADXL345 module.
d) Convert the raw acceleration values into corresponding acceleration due to gravity (*g*) values.
e) Apply the formulae to convert acceleration into Roll and Pitch values.
f) Calculate the angle required for movement of planes of platform in opposite direction.
g) Transfer angle of rotation to servo motors in X and Y axes.

5.4.1 Initialising ADXL345 module

For initialising the module, I^2C protocol is used for communicating with the ADXL345. The basic principle of communication with device is as following. First, we have to send the address of register we either want to read or write. Then we send the new values to write in the corresponding register or request some amount of bytes from the device.

The initialization of ADXL345, consists of three things, enabling measurement mode in register POWER_CTL, specifying the data format and setting the offset in registers – OFSX, OFSY, OFSZ.

To start the measurements we just need to set bit 3 in POWER_CTL register, basically we just write 0x08 to it. The command required to do this is:-

```
writeTo(ADXL345_POWER_CTL, 0x08);
```

After starting the ADXL345 module, we need to specify the data format. The register DATA_FORMAT consists of eight bits where each bit corresponds to certain meaning as shown in table below:

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Table 5.1 Contents of DATA_FORMAT register as per *ADXL345 datasheet*

We need to set only three bits of DATA_FORMAT register D0, D1 and D3 for extracting data from ADXL345 module.

When FULL_RES bit is enabled, the device will run in full resolution mode, in other words, it will always maintain 4mG/LSB. No matter what range is specified, one bit will represent 4mG of acceleration. If it is not enabled the ADXL345 will run in 10-bit mode, and the range bits will determine how many mg/LSB.

The Range bits basically set the range of the measurements. The table below specifies possible configurations for range bits:

Setting		
D1	D0	*g* Range
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Table 5.2 D0 and D1 bits for Range setting as per *ADXL345 datasheet*

5.4.2 Reading the raw acceleration and converting to Gs

After initialising ADXL345 we are ready to read the data from the accelerometer. The accelerations in raw format are stored in registers – DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0 and DATAZ1.

The results are divided in two 8 bit registers forming 16 bit registers where the format is LSB is first then followed by MSB. Again the data are stored in two's compliment format. With I^2C protocol, it is possible to request multiple bytes in one reading session. So to read the data, we just provided the address of DATAX0 and requested 6 bytes.

After reading raw data from the acceleration, we need to convert them to Gs (acceleration due to gravity). We now need to multiply the raw data with a a pre-calculated constant, which depends on our "Range" and "full resolution" settings.

The code snippet for initialising and extracting raw data from ADXL345 module is as shown below. The data obtained from module is passed through a low pass filter function as discussed in chapter 3 of thesis. The filtered value is then converted into corresponding G values and returned to main function for further processing.

```
AccG ADXL345::readAccG()
{
     raw;
     raw = readAccel();
     double fXg, fYg, fZg;
     fXg =raw.x * 0.00390625 + _xoffset;
     fYg =raw.y * 0.00390625 + _yoffset;
     fZg =raw.z * 0.00390625 + _zoffset;

     AccG res;
     res.x = fXg * ALPHA + (xg * (1.0-ALPHA));
     xg = res.x;
     res.y = fYg * ALPHA + (yg * (1.0-ALPHA));
     yg = res.y;
     res.z = fZg * ALPHA + (zg * (1.0-ALPHA));
     zg = res.z;
     return res;
     }
```

```
AccelRaw ADXL345::readAccel()
{
     readFrom(ADXL345_DATAX0, ADXL345_TO_READ, _buff); //read the
     acceleration data from the ADXL345
     // each axis reading comes in 16 bit resolution, ie 2 bytes.
     Least Significat Byte first!!
     // thus we are converting both bytes in to one int
     AccelRaw raw;
     raw.x = (((int)_buff[1]) << 8) | _buff[0];
     raw.y = (((int)_buff[3]) << 8) | _buff[2];
     raw.z = (((int)_buff[5]) << 8) | _buff[4];
     return raw;
}
```

CHAPTER 6

REALISATION OF ROLL PITCH STABILISED PLATFORM

6.1 Design of a 3-D printed Platform with two axes degree of freedom

For realisation of a Roll Pitch stabilised platform, there should be two degree of freedom so that the platform can rotate freely on X and Y axes based on input received from the controller. Such a design for platform was made using Computer Aided Design (CAD) software and the same was printed using a 3-D printer.

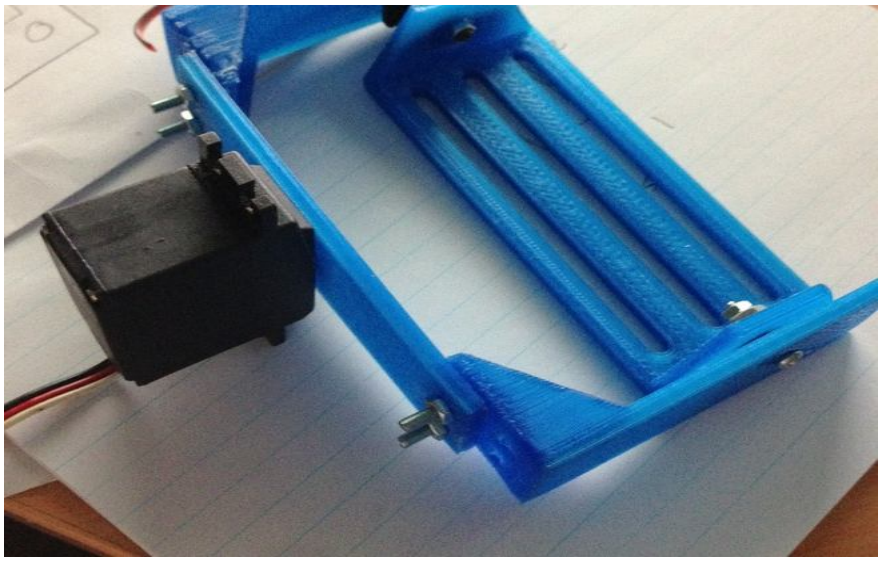

Figure 6.1 3-D printed part to align the platform in X-axis

For designing a 3-D printed part using 3-D printer, Fused deposition modelling (FDM) technology is used. It requires an STL (Stereo Lithography) file to be generated using any CAD software. The STL file is then given to the printing software which converts it into G-codes.

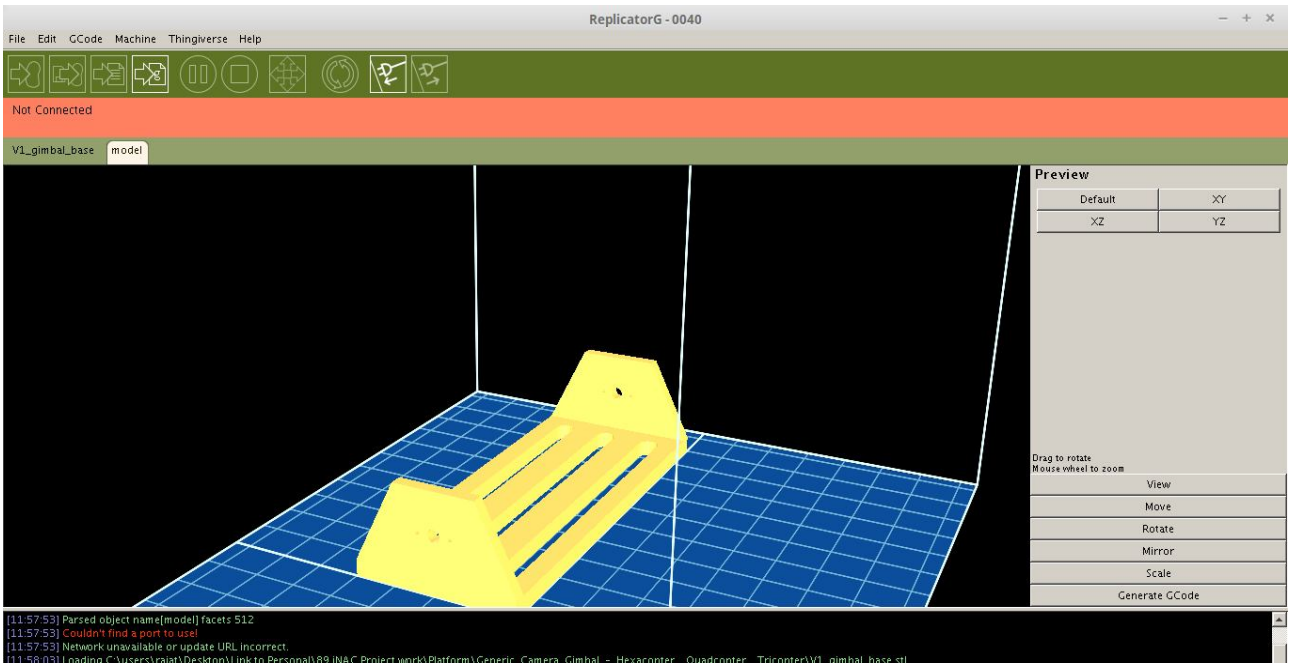

Figure 6.2 Software for generating G-codes from STL file

The G-code is then converted into a printer specific file which is given to 3-D printer that uses PLA (Poly Lactic acid) filament to print the 3-D part using layer deposition technique.

The project is an attempt to make a completely indigenised model of roll pitch stabilised platform. The model can be utilised for actual implementation onboard ships for various systems. It may not be directly used for weapon platforms as it would involve lot of testing and precision. However, for certain systems that do not require weapon like precision in stability such as a satellite TV dish, communication antenna, solar panel etc. may be placed on the platform stabilised using technique discussed in this thesis.

As shown in figure 6.2 above, the two planes of platform are connected to two servo motors that are controlled by the micro-controller that receives roll and pitch data from ADXL345 module interfaced over I^2C protocol.

6.2 Interfacing servo motors to micro-controller

Another part of this project is interfacing servo motors to the micro-controller that would be made to rotate in direction opposite to roll and pitch as obtained from the sensor module.

A servo motor consists of following parts:-

a) DC motor
b) Gearing set
c) Control circuit
d) Position sensor

PWM is used for the control signal of servo motors. Unlike DC motors its the duration of positive pulse that determines the position rather than the speed of servo shaft.

Increasing more than neutral value (pulse dependent on servo which keeps the servo shaft in the centre position) will make the servo turn clockwise and a shorter pulse will turn the shaft anticlockwise. The servo control pulse is usually repeated every 20 ms, essentially telling the servo where to go, even if that means remaining in the same position.

Figure 6.3 Servo motor

Servo motors have three wires:

a) Power

b) Ground

c) Control

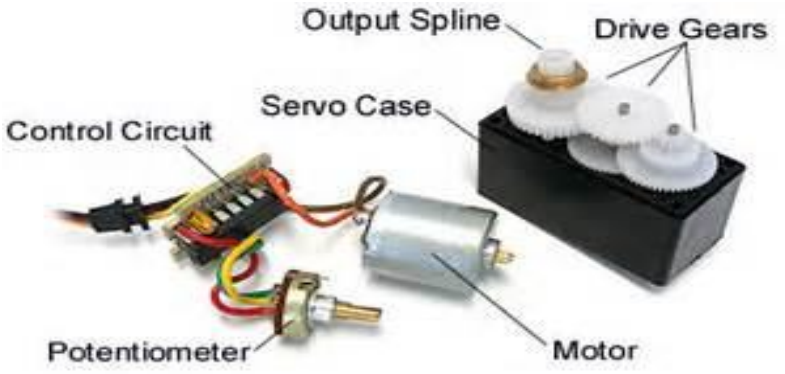

Figure 6.4 Parts of servo motor

For interfacing servo motors to micro-controller board following schematic is to be referred.

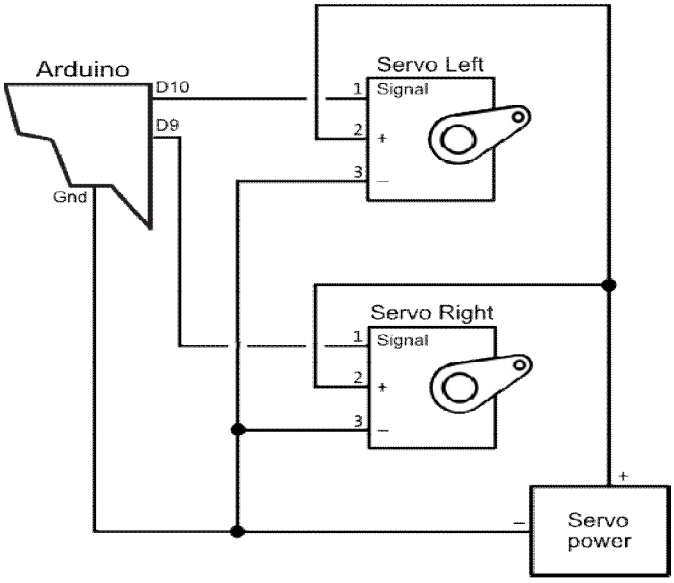

Figure 6.5 Schematic to interface two servo motors to micro-controller board

Using above schematic two different servo motors are connected to stabilising platforms in x and y axes respectively.

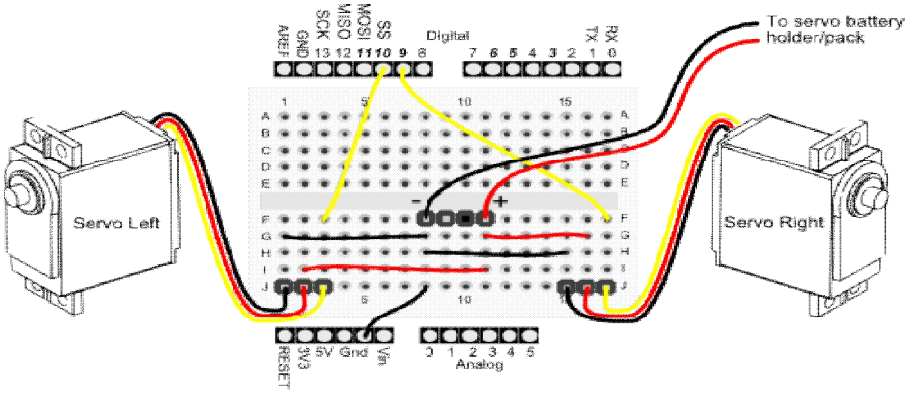

Figure 6.6 Actual connection diagram of servo motors interfaced to micro-controller board

6.3 Negative feedback to servo motors to stabilise platform

The roll and pitch calculated using algorithm described in chapter 3 above is further converted into angular position that the servo motors needs to be moved in directions opposite to experienced roll and pitch.

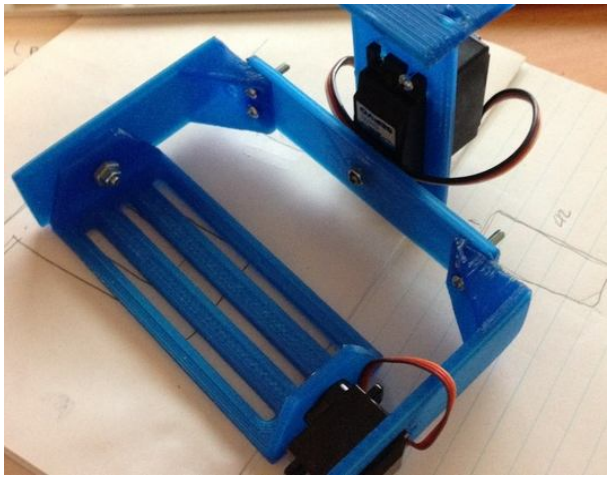

Figure 6.7 Servo motors connected to stabilising platform in X and Y axis

6.4 Circuit schematic for development of controller for stabilised platform

The realisation of circuit without using Arduino board was done using ATMEGA328 micro-controller based board with ADXL345 and servo motors with external power supply. The circuit is developed using open source software '*Fritzing*'.

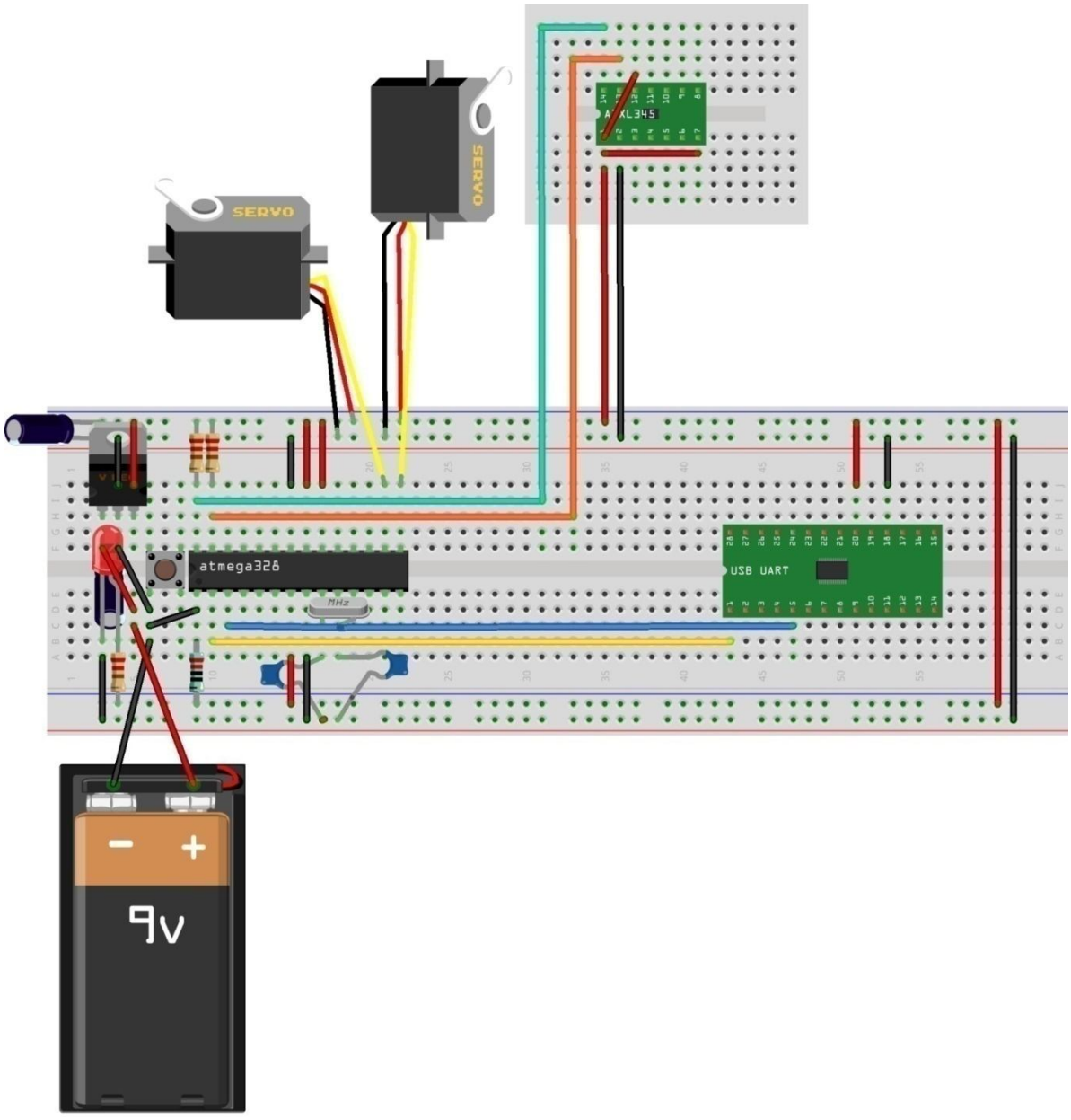

Figure 6.8 Circuit schematic for Controller for stabilised platform

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

The objective of proving the concept of stabilised platform against roll and pitch experienced by ship has been achieved. The platform which is free to rotate in two degrees has been controlled using inputs received from ADXL345 module which constantly updates the acceleration experienced in X and Y axes and transmits it to micro-controller over I^2C protocol. The controller calculates the equivalent roll and pitch experienced by the ship using algorithm specified in chapter 4. It then calculates the angle in opposite direction (negative feedback) that needs to be given to two servo motors connected to X and Y planes of platform and rotates it to counter roll and pitch thereby keeping the object placed on platform stable.

Realisation of concept was done using Arduino based ATMEGA 328 micro-controller board and ADXL345 module. The servo motors used for developing the model are 5V rating. However, a circuit schematic for complete setup developed using micro-controller over printed circuit board, FTDI chip and 12 V servo motors is suggested in chapter 6.

7.2 Future Work

The future scope of this project would be testing and employing the project onboard naval ships / submarines and recording the performance. Later, it can be gradually employed for stabilising some of the ships systems such as satellite dish and solar panels as suggested in chapter 2.

Further, the same setup can be upgraded to develop an Inertial Navigation System (INS) by interfacing a gyroscope module which would calculate angular momentum in three axes continuously. The data would be used in coordination with accelerometer data to calculate the position of vessel in space without any external input using algorithms suggested in [5]. However, as discussed INS is prone to drifting problem and its accuracy is doubtful in long duration. Hence, it would require lot of testing and tuning before actual deployment onboard Naval platforms.

REFERENCES

[1] Seungbae Lee, Gi-Joon Nam, Junseok Chae, Hanseup Kim and Alan J. Drake, "Two-Dimensional Position Detection System with MEMS Accelerometer for Mouse Applications", Department of EECS, University of Michigan.

[2] Kevin J Walchko and Dr. Paul A.C, "Inertial Navigation", Florida Conference on Recent Advances in Robotics, 2002.

[3] Xudong Wang and Li Quan "Filtering the Acceleration Signal in Different Algorithm for Comparison and Analysis", IEEE ICCP 2011 Proceedings pp. 171-173.

[4] Guangchun Li, Yunfeng He, Yanhui Wer, "The MEMS gyro stabilized platform design based on Kalman Filter" in IEEE 978-1-4799-1216-2/13, pp. 14 – 17, 2013.

[5] Qu Pingping, Fu Li, Zhao Xin, "Design of Inertial Navigation System Based on Micromechanical Gyroscope and Accelerometer", IEEE 978-1-4244-2723-9/09, pp. 1351-1354, CCDC 2009.

[6] Tao Liu, Yoshio Inoue and Kyoko Shibata, "A Novel Motion Sensor with Nine Degrees of Freedom", 4th International Conference on Biomedical Engineering and Informatics, 2011.

[7] Alexandre Patarot, Mehdi Boukallel and Sylvie Lamy-Perbal, "A Case Study On Sensors And Techniques For Pedestrian Inertial Navigation", IEEE 978-1-4799-0916-2/14, 2014

APPENDIX A

A.1 I^2C scanner code developed over Arduino IDE

```
#include <Wire.h>
void setup()
{
  Serial.begin (115200);

  while (!Serial)
    {
    }

  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;

  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
      {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      count++;
      delay (1);
      } // end of good response
  } // end of for loop
  Serial.println ("Done.");
  Serial.print ("Found ");
  Serial.print (count, DEC);
  Serial.println (" device(s).");
}  // end of setup

void loop()
{
}
```

A.2 Program code to read raw X,Y,Z acceleration values of ADXL345

```
/*  Program to initialise control registers of ADXL345
    Register 0x31  =  0x0B
    Register 0x2D  =  0x08
    Register 0x2E  =  0x80
    and read RAW X,Y,Z data */

#include <Wire.h>

int data=0;

void setup()
{
  Serial.begin (115200);

  Wire.begin();
  Serial.println("Regsiters Status before---");
  read_ADXL345();

  Wire.beginTransmission(0x53);
  Wire.write(0x31);
  Wire.write(0x0B);
  Wire.endTransmission();

  Wire.beginTransmission(0x53);
  Wire.write(0x2D);
  Wire.write(0x08);
  Wire.endTransmission();

  Wire.beginTransmission(0x53);
  Wire.write(0x2E);
  Wire.write(0x80);
  Wire.endTransmission();
  delay(250);

  Serial.println("Regsiters Status After---");
  read_ADXL345();
}
void read_ADXL345()
 {
  for(int i=0;i<=127;i++)
   {
    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(i);
    Wire.endTransmission();
```

```
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    Serial.print("Register No. ");
    Serial.print(i,HEX);
    Serial.print(" --Data= ");
    Serial.print(data,HEX);
    Serial.println();
  }
 }

void loop()
{
    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x32);
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    Serial.print("X0=");
    Serial.print(data,HEX);
    Serial.println();

    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x34);
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    Serial.print("Y0=");
    Serial.print(data,HEX);
    Serial.println();

    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x36);
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    Serial.print("Z0=");
    Serial.print(data,HEX);
    Serial.println();
}
```

A.3 Program code to transmit raw X,Y,Z acceleration values from ADXL345 module

```
/*   Program to Transmit XYZ registers values of ADXL345 over UART in Comma Separated Values
*/

#include <Wire.h>

int data=0;
int XYZ[3];

void setup()
{
  Serial.begin (4800);

// Write Control registers

  Wire.beginTransmission(0x53);
  Wire.write(0x31);
  Wire.write(0x0B);
  Wire.endTransmission();

  Wire.beginTransmission(0x53);
  Wire.write(0x2D);
  Wire.write(0x08);
  Wire.endTransmission();

  Wire.beginTransmission(0x53);
  Wire.write(0x2E);
  Wire.write(0x80);
  Wire.endTransmission();

  delay(250);

}
```

```
void loop()
{

    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x32);              // X0 Reg
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    XYZ[0]=data;

    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x34);              // Y0 Reg
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    XYZ[1]=data;

    Wire.beginTransmission(0x53);  // Address of ADXL345
    Wire.write(0x36);
    Wire.endTransmission();
    Wire.requestFrom(0x53,1);
    if (Wire.available())
      data=Wire.read();

    XYZ[2]=data;

    for (int i=0;i<3;i++)
    {
      Serial.print(XYZ[i]);
      Serial.print(",");
    }
      Serial.println();
  delay(500);
}
```

A.4 Processing program code to make GUI to plot acceleration values

```
import processing.serial.*;
Serial serial;

String stringAccX, stringAccY;

int width = 800;
int height = 600;

float[] accX = new float[width];
float[] accY = new float[width];



boolean drawValues = false;

void setup() {

size(width, height);

println(Serial.list()); // display connected serial devices

serial = new Serial(this, Serial.list()[0], 115200);
serial.bufferUntil('\n'); // Buffer until line feed

for (int i = 0; i < width; i++) // center all variables
{
    accX[i] = height/2;
    accY[i] = height/2;
}

  drawGraph(); // Draw graph at startup

}

void draw() {

  if (drawValues) {

    drawValues = false;
    drawGraph();

  }

}
```

```
void drawGraph()

{
 background(255); // White

  for (int i = 0; i < width; i++) {

    stroke(200); // Grey

    line(i*10, 0, i*10, height);

    line(0, i*10, width, i*10);

  }



  stroke(0); // Black

  for (int i = 1; i <= 3; i++)

    line(0,  height/4*i,  width,  height/4*i);  //  Draw  line,
indicating -90 deg, 0 deg and 90 deg

  convert();
  drawAxisX();
  drawAxisY();

}


void serialEvent (Serial serial) {

  // Get the ASCII strings:

  stringAccX = serial.readStringUntil('\t');
  serial.readStringUntil('\t'); // Ignore extra tab
  stringAccY = serial.readStringUntil('\t');
  serial.clear(); // Clear buffer
  drawValues = true; // Draw the graph

}
```