INTERNSHIP REPORT
On
X-Y motor control

Submitted by
RAYAN
JAUHAR

in partial fulfillment of requirement for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION

DIVISION OF ELECTRONICS ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
KOCHI-682022
held from
30th May 2022 to 26th June 2022

inQbe Innovations pvt Ltd.

Let us Incubate Technology

inQbe

IIPV/CR/2022-23/002 ***26/07/2022***

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. RAYAN ABDUL NASSER,** with **B.Tech Degree** in **Electronics and Communication Engineering** from **CUSAT**, Ernakulam had done an internship in our organization from 30-5-2022 to 26-6-2022.

During this period, he got familiarized with the activities of our various departments and was able to handle a part of our work in Research and Development side.

He has mainly worked on coding during the internship tenure.

We wish him all success in future endeavours.

For INQBE INNOVATIONS PVT. LTD.

DIRECTOR

INQBE INNOVATIONS PVT. LTD. KOCHI 682 017 KALOOR

PH: 0484-4039369/ 7907481165
Email: inqbeinnovations@gmail.com
www.inqbe.in

B.No- 65/1418, 1st Floor, Shenoy Road, Kaloor
PIN- 682017

inQbe Innovations pvt Ltd.

Let us Incubate Technology

inQbe

IIPV/CR/2022-23/001 ***26/07/2022***

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. JAUHAR K**, with **B.Tech Degree** in **Electronics and Communication Engineering** from **CUSAT**, Ernakulam had done an internship in our organization from 30-5-2022 to 26-6-2022.

During this period, he got familiarized with the activities of our various departments and was able to handle a part of our work in Research and Development side.

He has mainly worked on coding during the internship tenure.

We wish him all success in future endeavours.

For INQBE INNOVATIONS PVT. LTD.

DIRECTOR

INQBE INNOVATIONS PVT. LTD. KOCHI 682 017 KALOOR

PH: 0484-4039369/ 7907481165
Email: inqbeinnovations@gmail.com
www.inqbe.in

B.No- 65/1418, 1st Floor, Shenoy Road,
Kaloor
PIN- 682017

ACKNOWLEDGEMENT

We express our overwhelming indebtedness to God Almighty for the successful completion of our internship.Our sincere efforts have made us accomplish the task of completing this internship. However it would not have been possible without the kind support and help of many individuals.We are highly indebted to our mentor Aslam Sir for their guidance and constant supervision while doing the internship. Their constant guidance and willingness to share their vast knowledge made us understand more about the field in great depth and helped us to complete assigned tasks on time. Lastly we would like to express our sincere thanks and gratitude to our university for giving time and opportunity for doing internship.

ABSTRACT

Industrial training is an important phase of a student's life. A well planned, properly executed and evaluated industrial training helps a lot in developing a professional attitude. It develops an awareness of industrial approach to problem solving, based on a broad understanding of process and mode of operation of organisation. The aim and motivation of industrial training is to receive discipline, skills, teamwork and technical knowledge through a proper training environment, which will help me, as a student in the field of Electronics and Communication, to develop a responsiveness of the self-disciplinary nature of problems in the professional world.
The internship was one month long offline internship offered by inqube .The purpose of this internship was to get familiarized in labview and embedded c coding as per the customer’s needs.

TABLE OF CONTENTS

1- Introduction

2- Modbus protocol

3- NI LabView

4- X y motor control

5- Motor frame design

6- Circuit design

6.1-Motor driver Module

6.2-RS485 to UART module

6.3- Voltage Regulator

6.4- Sensor

7- Embedded C software

8-Software development using LabView

9-Finished product

1.INTRODUCTION

Our internship program was a one month Hands on internship program at InQbe Innovations Pvt Ltd, an Electronics Research and Development company based on Ernakulam.

During our time there, we got a chance to work on projects and learn new things under the guidance of professionals in Electronics field. We were introduced to different tools, softwares, equipments and protocols used in the Field.

We were tasked with Software part of a project, which will be detailed in coming chapeters. Firstly, we had to learn two new things.

1- Modbus RTU protocol

2- NI Labview.

Modbus is communication protocol used in industries for communication.

2.MODBUS PROTOCOL

MODBUS© Protocol is a messaging structure, widely used to establish master-slave communication between intelligent devices. A MODBUS message sent from a master to a slave contains the address of the slave, the 'command' (e.g. 'read register' or 'write register'), the data, and a check sum (LRC or CRC).

Since Modbus protocol is just a messaging structure, it is independent of the underlying physical layer. It is traditionally implemented using RS232, RS422, or RS485

The Request

The function code in the request tells the addressed slave device what kind of action to perform. The data bytes contains any additional information that the slave will need to perform the function. For example, function code 03 will request the slave to read holding registers and respond with their contents. The data field must contain the information telling the slave which register to start at and how many registers to read. The error check field provides a method for the slave to validate the integrity of the message contents.

The Response

If the slave makes a normal response, the function code in the response is an echo of the function code in the request. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

Controllers can be setup to communicate on standard Modbus networks using either of two transmission modes: ASCII or RTU.

In Our Project, We have used MODBUS RTU, implemented using RS485

RTU Mode

When controllers are setup to communicate on a Modbus network using RTU (Remote Terminal Unit) mode, each eight-bit byte in a message contains two four-bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII for the same baud rate. Each message must be transmitted in a continuous stream.

Coding System

Eight-bit binary, hexadecimal 0 ... 9, A ... F
Two hexadecimal characters contained in each eight-bit field of the message

Bits per Byte

1 start bit
8 data bits, least significant bit sent first
1 bit for even / odd parity-no bit for no parity
1 stop bit if parity is used-2 bits if no parity

Error Check Field

Cyclical Redundancy Check (CRC)

Start	Address	Function	Data	LRC	End
:	2 Chars	2 Chars	N Chars	2 Chars	CR LF

Address Field

The address field of a message frame contains two characters (ASCII) or eight bits (RTU). The individual slave devices are assigned addresses in the range of 1 ... 247.

Function Field

The Function Code field tells the addressed slave what function to perform.

The following functions are supported by Modbus poll

Function Code	Action	Table Name
01 (01 hex)	Read	Discrete Output Coils
05 (05 hex)	Write single	Discrete Output Coil
15 (0F hex)	Write multiple	Discrete Output Coils
02 (02 hex)	Read	Discrete Input Contacts
04 (04 hex)	Read	Analog Input Registers
03 (03 hex)	Read	Analog Output Holding Registers
06 (06 hex)	Write single	Analog Output Holding Register
16 (10 hex)	Write multiple	Analog Output Holding Registers

The data field contains the requested or send data.

3.NI LABVIEW

INTRO

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming environment which has become prevalent throughout research labs, academia, and industry. It is a powerful and versatile analysis and instrumentation software system for measurement and automation. It's graphical programming language called G programming is performed using a graphical block diagram that compiles into machine code and eliminates a lot of the syntactical details.

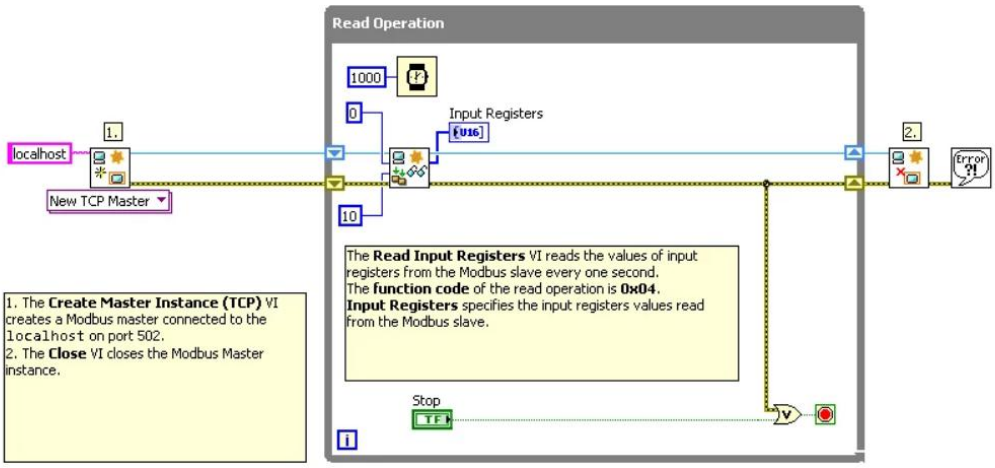
LabVIEW offers more flexibility than standard laboratory instruments because it is software-based. Using LabVIEW, the user can originate exactly the type of virtual instrument needed and programmers can easily view and modify data or control inputs. The popularity of the National Instruments LabVIEW graphical dataflow software for beginners and experienced programmers in so many different engineering applications and industries can be attributed to the software's intuitive graphical programming language used for automating measurement and control systems.

LabVIEW programs are called virtual instruments (VIs), because their appearance and operation imitate physical instruments like oscilloscopes. LabVIEW is designed to facilitate data collection and analysis, as well as offers numerous display options. With data collection, analysis and display combined in a flexible programming environment, the desktop computer functions as a dedicated measurement device. LabVIEW contains a comprehensive set of VIs and functions for acquiring, analyzing, displaying, and storing data, as well as tools to help you troubleshoot your code.

All test, measurement and control applications can be divided into three main components and the key to virtual instrumentation is the ability to acquire, analyze and present data. LabVIEW can acquire data using the devices like GPIB, Serial, Ethernet, VXI, PXI Instruments, Data Acquisition (DAQ), PCI extensions for Instrumentation (PXI), Image Acquisition (IMAQ), Motion Control, Real-Time (RT) PXI, PLC (through OPC Server), PDA, and Modular Instruments.

Modbus using Labview

The essential part we used in LabVIEW is the modbus communication feature Which is readily availbale in it.

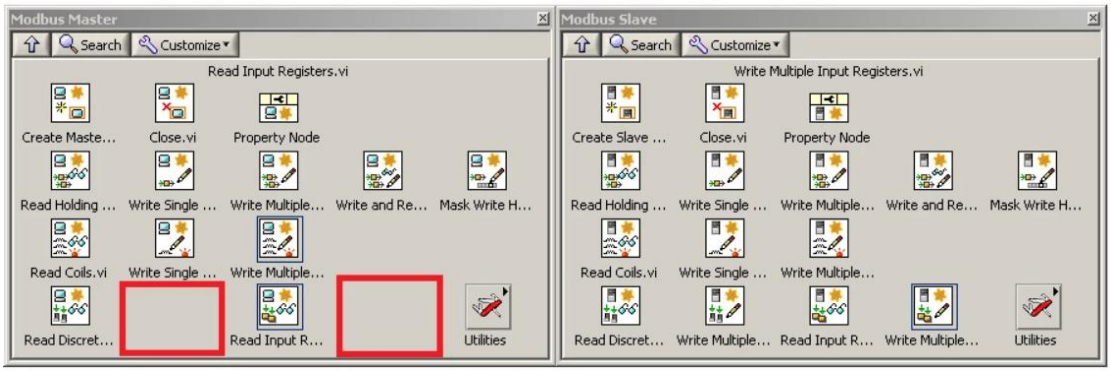


The above code explains the core requirements of a Modbus application using the LabVIEW API. First, a Modbus instance is created. In this case, a TCP master. However, you can switch this example to a serial master by changing the polymorphic instance selector

Figure 2. Changing the Type of Modbus Master

When the instance is created, you can begin polling the slave device for data. The example shows the use of the function code Read Input Registers. All Modbus function codes supported by the API are shown on the appropriate palette. Because of implementation of the protocol, the slave API has additional functions that the master

cannot implement. For example, a slave can write to the input register range, while a master may only read from that range. Figure 3 shows the function codes.



Finally, the Modbus instance is closed, de-allocating the memory associated with the instance. This also closes any references, including the TCP connection or NI-VISA serial references used by the instance

4.PROJECT: X-Y Motor Control

Customer requirement:

User has to take readings from an outdoor rectangular pond of 15*5 meters in dimensions. Our task was to provide a solution for user to take sensor readings from anywhere in the pond.

Fig 1

Our solution was to design a two axis controllable platform that can run over the top of rectangular pond and take readings. Three motors were used for movement in two axes. For lengthwise movement, we used two motors on each side, and for width wise, only One motor was used. As the pond is placed in outdoor, and the motor platform is permanently fixed over it, it has to show high reliability in different climates and situations. Also the user has to control and monitor the motor positions over a distance.

So the tasks are -

1- Implementing a two axis motor system
2- Control and monitor over a distance
3- Reliability in different conditions

There were 4 different parts in the Project.

1- Motor and Frame design
2- Circuit design

3- Embedded C software development for Arduino
4- Software development for PC (for Control and Monitoring).

We were tasked with developing software code for control and monitoring of a two axis motor plane.

5.Motor and Frame Design

The original Rectangular pond was 15 meters long and 5 meters wide. But during development, we built a 2 meters long and 1 meter wide prototype, and designed the project such that it can be easily scaled to any dimensions as required. The frame was rectangular Aluminium frame on which two motors were mounted. The motors used were 12v wiper motors used in Automobiles.

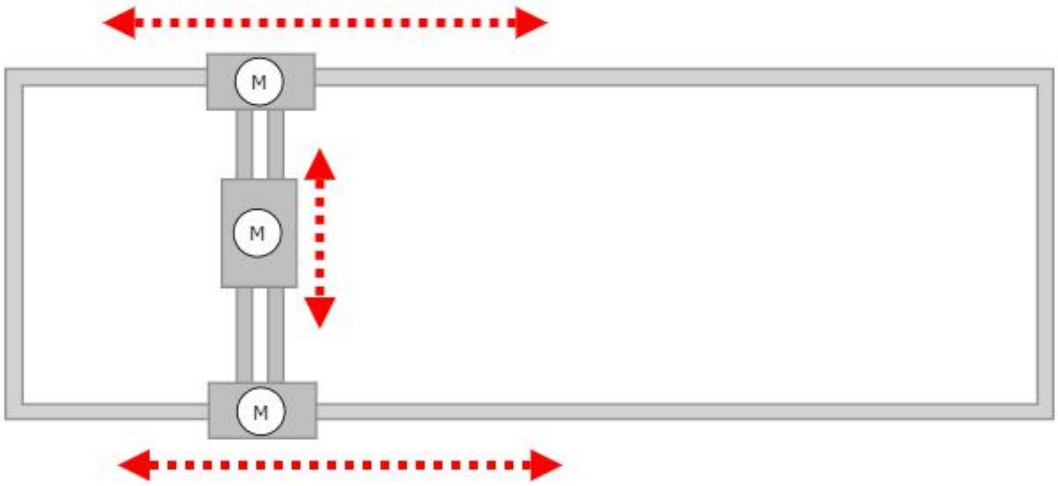


Fig 2

Fig 3

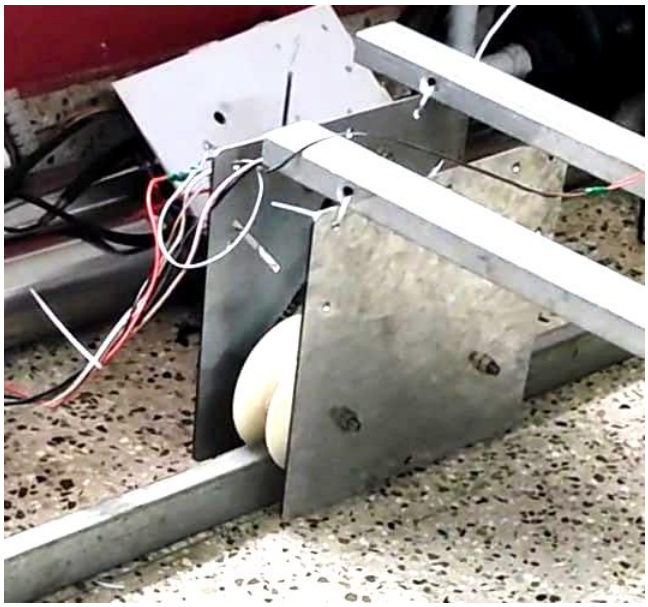


Fig 4

We didn’t have much to contribute in the motor and frame design. But knowing the hardware that we work on is very important in embedded coding, as we have to directly interact with the hardware.

6.Circuit Design

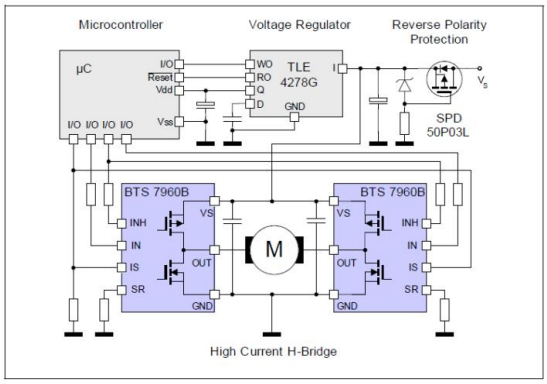
Circuit has to be designed for running the 12V motors. Arduino will provide pwm signals for motor control, which has to be fed to corresponding motor driver modules. Also 5v voltage regulator and communication modules has to be implemented in the circuit board.

Circuit board has to contain

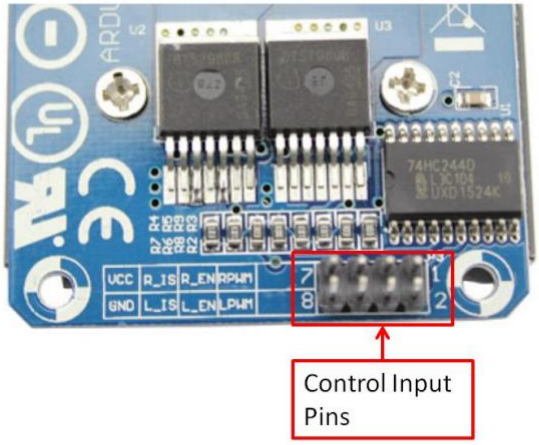
1- 3 Motor driver modules (one for each motor)
2- RS485 to UART module (for communication over a distance)
3- Voltage regulators
4- Connectors for sensors

1-**Motor driver Module - BTS7960 H-bridge Module**

BTS7960 is a fully integrated high current H bridge module for motor driving applications. It supports Input voltage from 6-27Vdc and supports current upto 43Amps. Control input voltage from 3.3 to 5Vdc, making it compatible with both 3.3 and 5V logic level controllers. It also provides slew rate adjustments and dead time insertion and protection against over temperature, over voltage and over current situations.

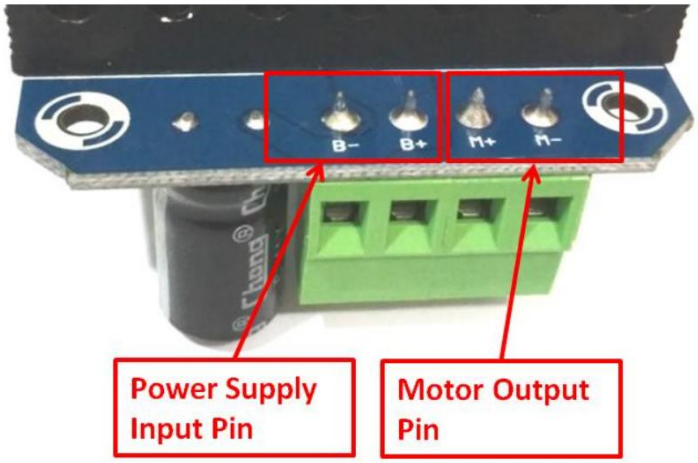


Control Input Pin Function:



Pin No	Function	Description
1	RPWM	Forward Level or PWM signal, Active High
2	LPWM	Reverse Level or PWM signal, Active High
3	R_EN	Forward Drive Enable Input, Active High/ Low Disable
4	L_EN	Reverse Drive Enable Input, Active High/Low Disable
5	R_IS	Forward Drive, Side current alarm output
6	L_IS	Reverse Drive, Side current alarm output
7	Vcc	+5V Power Supply microcontroller
8	Gnd	Ground Power Supply microcontroller

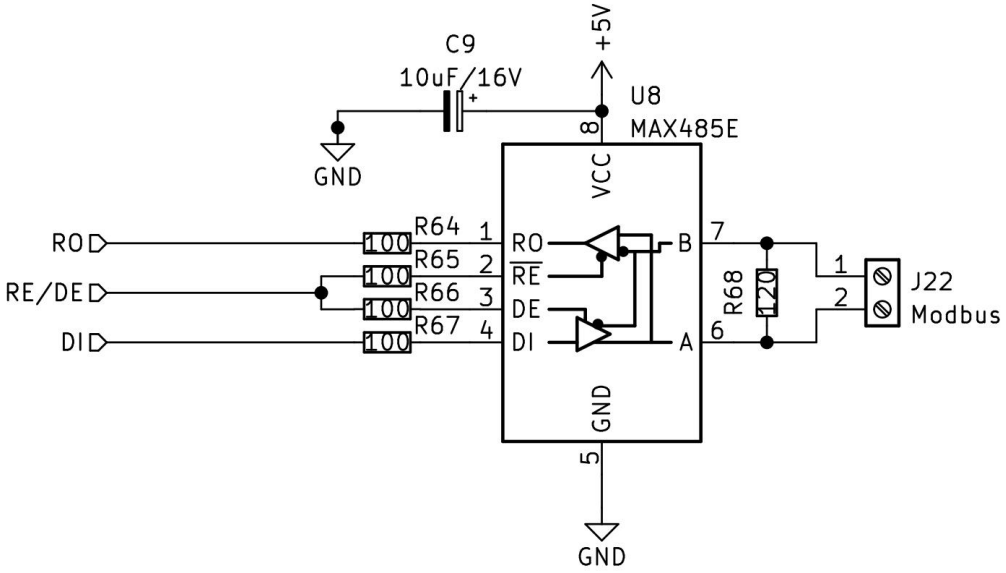
Motor Power Supply & Output Pin Assignment:



Pin No	Function	Description
1	B+	Positive Motor Power Supply. 6 ~ 27VDC
2	B-	Negative Motor Power Supply. Ground
3	M+	Motor Output +
4	M-	Motor Output -

2-RS485 to UART module (MAX485E)

Modbus protocol us used for communication. Its is a serial wire protocol, and can be easily send and received over UART. But as our Motor and Control part are located over some distance, UART signals can be corrupted by noise over distance. So the UART signal from Arduino and PC is converted to RS485, which can then be transmitted over longer distance.



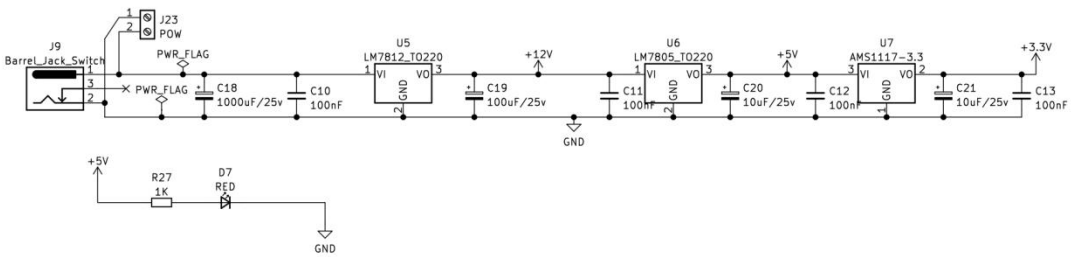
3- Voltage Regulator (12V-5V-3.3V outputs)

12V, 5V, and 3.3V outputs were provided in the circuit. 12V are required for some sensors while 5V and 3.3V are used for operating Microcontrollers. Voltage regulators

LM7812 used for 12V fixed output

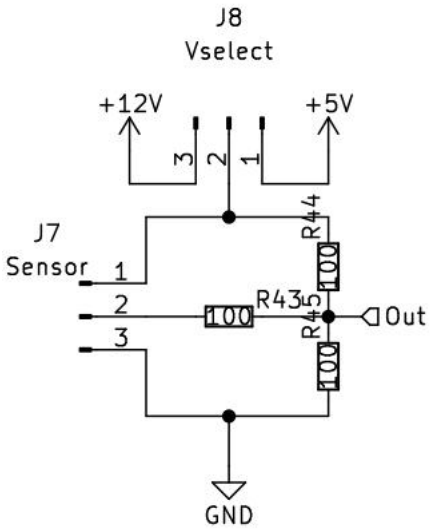
LM7805 used for 5V

AMS1117 used for 3.3V



4- Sensor inputs

Sensor inputs supporting 12v, 5V sensors both NPN and PNP type are added.



EVEN THOUGH, We didn't have much contribution towards the component selection and circuit design, understanding of every components and its working is very important for writing its driver code.

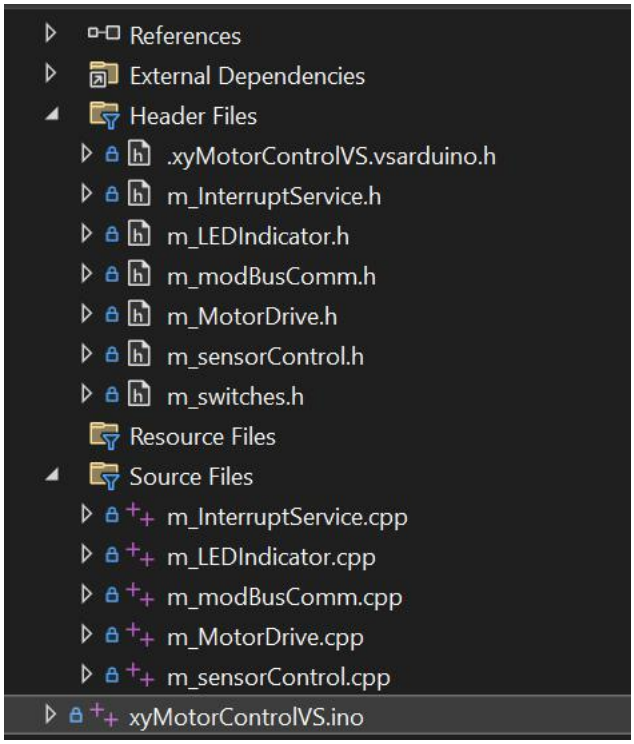
7.Embedded C software development for Arduino

We were assigned with the task for Embedded C software development for Arduino and also with control and monitoring software app generation for windows PC.

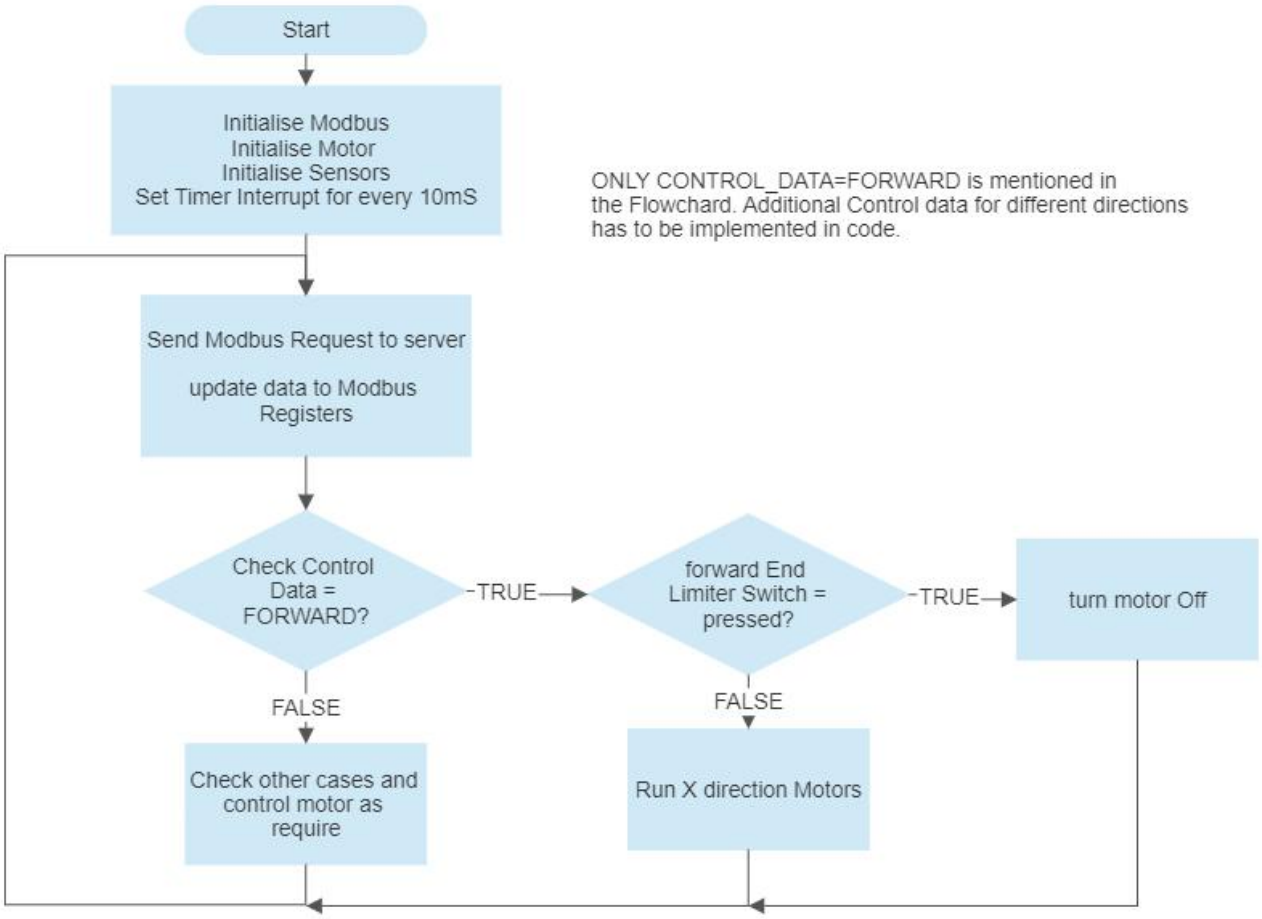
For Embedded C software in Arduino, we need to

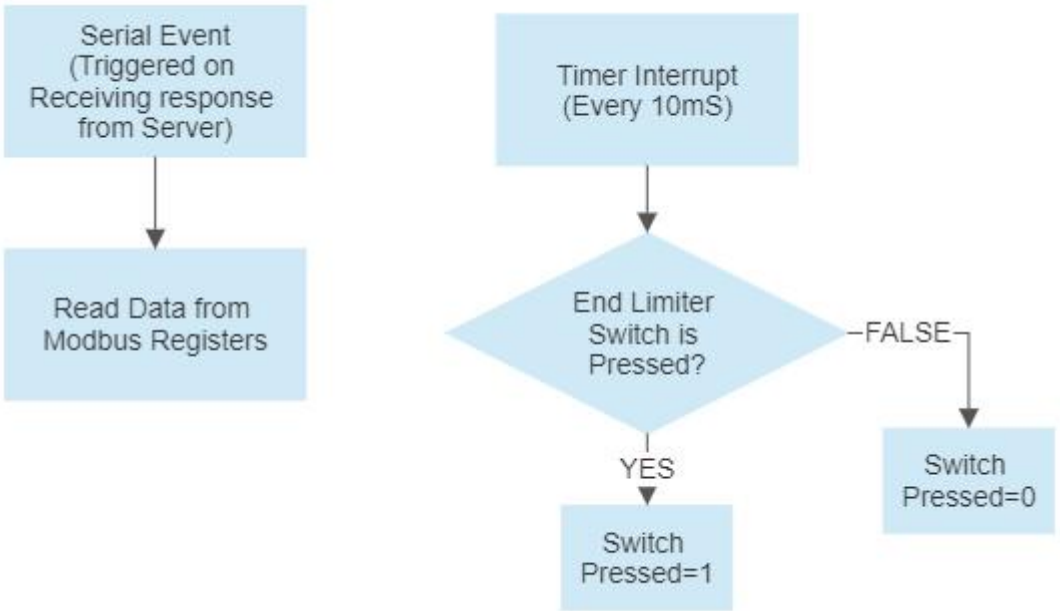
1- Send and receive controls and data through Modbus
2- Control motor using PWM signal
3- Read and store motor rotation count using sensors
4- Keep motor under limits using end limiter switches
5- Calibrate and check errors in motor positions

Arduino Code Project Structure:



xyMotorControlVS.ino contain the setup and Loop part of the Arduino code. Code for each modules like Motor, sensor, Switch… are written seperately in seperate .c and .h files.


Start
Initialise Modbus
Initialise Motor
Initialise Sensors
Set Timer Interrupt for every 10mS
ONLY CONTROL_DATA=FORWARD is mentioned in the Flowchard. Additional Control data for different directions has to be implemented in code.
Send Modbus Request to server
update data to Modbus Registers
Check Control Data = FORWARD?
TRUE
forward End Limiter Switch = pressed?
TRUE
turn motor Off
FALSE
FALSE
Check other cases and control motor as require
Run X direction Motors


Serial Event (Triggered on Receiving response from Server)
Read Data from Modbus Registers
Timer Interrupt (Every 10mS)
End Limiter Switch is Pressed?
FALSE
YES
Switch Pressed=1
Switch Pressed=0

Xy_MotorControlVS.ino

```
/*
* XY MOTOR CONTROL PROJECT
* @JAUHAR K
*/

// the setup function runs once when you press reset or power the board

#include "m_switches.h"
#include <NoDelay.h>
#include "m_LEDIndicator.h"
#include "m_InterruptService.h"
#include "m_sensorControl.h"
#include <ArduinoRS485.h>
#include <ArduinoModbus.h>
#include "m_modBusComm.h"
#include "m_MotorDrive.h"


/* ============================================================== */
/* SETUP ----------------------------------*/

/* ---------------------------------------- */

void setup() {

    /* Motor Initialisation */

    /* X direction motor */
    MotorXa.motorInit();
    MotorXa.enableMotor();

    MotorXb.motorInit();
    MotorXb.enableMotor();

    /* Y direction motor */
    MotorYa.motorInit();
    MotorYa.enableMotor();


    /* Initialisint sensor for motor xa */
    while (m_Sensor_Xa.initSensor() == false) {
        Serial.println("FAILED TO INITIALISE SENSOR Xa");
        Serial.println("CHECK THE SENSORS");
        errorIndicator();
    }
    Serial.println("SUCCESSFULLY INITIALISED SENSOR Xa");

    /* Initialisint sensor for motor xb */
    while (m_Sensor_Xb.initSensor() == false) {
        Serial.println("FAILED TO INITIALISE SENSOR Xb");
        Serial.println("CHECK THE SENSORS");
        errorIndicator();
    }
    Serial.println("SUCCESSFULLY INITIALISED SENSOR Xb");

    /* Initialisint sensor for motor Ya*/
    while (m_Sensor_Ya.initSensor() == false) {
```

```
        Serial.println("FAILED TO INITIALISE SENSOR Ya");
        Serial.println("CHECK THE SENSORS");
        errorIndicator();
    }
    Serial.println("SUCCESSFULLY INITIALISED SENSOR Ya");

    /* Connecting modbus to client */
    while (myController.connectToClient()==false)
    {
        Serial.println("FAILED TO CONNECT TO MASTER");
        errorIndicator();
    }
    Serial.println("CONNECTED TO CLIENT");

    setTimerInterrupt();   /* setting timer interrupt 10ms */

    /* ============================================================ */
    /* Limiter switch initialisation */
    Xa_switch.init(); /* initialising Xa limiter switch */
    /*
    * create new limiter switch objects in m_switches.h lib file. details there.
    * add limiter switch initialisations for every limtier switch here...
    */
    /* ============================================================ */
    /* initialising led indicators.. */
    setupLEDIndicators();
}

/* ============================================================ */
/* LOOP ----------------------------------*/

noDelay fun1Time(1);
int CONTROL_KEY = 0;   /* creating a non blocking delay object for motor pwm
update */

void loop()
{
    myController.poll();
    /* COMMUNICATION DATA UPDATE*/
    /*-----------------------------------------------*/
    /*
    * data in myController.toClientData array will be send to client through
modbus
    */
    myController.toClientData[0] = MotorXa.getState(); /*index 0 in labview*/
    myController.toClientData[1] = MotorXb.getState();
    myController.toClientData[2] = m_Sensor_Xa.getCount();
    myController.toClientData[3] = m_Sensor_Xb.getCount(); /*index 3 in
labview*/
    myController.toClientData[4] = MotorYa.getState();
    myController.toClientData[5] = m_Sensor_Ya.getCount();

    myController.sendDataToClient();
    /* MOTOR CONTROLS */

    CONTROL_KEY = myController.getCtrlData(); /* reading the control
instruction from client */

    /*
============================================================================
========================== */
    /* motor should'nt move if limiterSwitch is pressed and still the control
is pressed down */
```

```
    /* LIMITER SWITCH CODES AND CONDITIONS TO BE CHECKED HERE */
    if ((Xa_switch.getFlag() == 1) && (CONTROL_KEY == CTRL_DOWN)) {
        MotorXa.motorHALT();
        MotorXb.motorHALT();
        m_Sensor_Xa.resetSensor();
        m_Sensor_Xb.resetSensor();
    }

    /*
    * CODE FOR LIMITER SWITCH 2
    * CODE FOR LIMITER SWITCH 3
    * ...
    */

    /*
================================================================================
========================== */

    switch (CONTROL_KEY) {
        /*-----------------------------------------------*/
    case 0:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_OFF(IND_LED2);
        ledIndicator_OFF(IND_LED3);
        ledIndicator_OFF(IND_LED4);
        ledIndicator_OFF(IND_LED5);
        MotorXa.motorSTOP();
        MotorXb.motorSTOP();
        MotorYa.motorSTOP();
        break;
        /*-----------------------------------------------*/
    case CTRL_UP:
        ledIndicator_ON(IND_LED1);
        MotorYa.motorSTOP();
        if (fun1Time.update()) {   /* non blocking delay for soft start */
            MotorXa.ccRotate();
            MotorXb.cRotate();
        }
        break;
        /*-----------------------------------------------*/
    case CTRL_DOWN:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_ON(IND_LED2);
        MotorYa.motorSTOP();
        if (fun1Time.update()) {
            MotorXa.cRotate();
            MotorXb.ccRotate();
        }
        break;
        /*-----------------------------------------------*/
    case CTRL_RIGHT:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_OFF(IND_LED2);
        ledIndicator_ON(IND_LED3);
        MotorXa.motorSTOP();
        MotorXb.motorSTOP();
        if (fun1Time.update()) {
            MotorYa.cRotate();
        }
        break;
        /*-----------------------------------------------*/
    case CTRL_LEFT:
        ledIndicator_OFF(IND_LED1);
```

```
        ledIndicator_OFF(IND_LED2);
        ledIndicator_OFF(IND_LED3);
        ledIndicator_ON(IND_LED4);
        MotorXa.motorSTOP();
        MotorXb.motorSTOP();
        if (fun1Time.update()) {
            MotorYa.ccRotate();
        }
        break;
        /*-------------------------------------------------*/
    case CTRL_SET_ORIGIN:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_OFF(IND_LED2);
        ledIndicator_OFF(IND_LED3);
        ledIndicator_OFF(IND_LED4);
        ledIndicator_ON(IND_LED5);
        while (Xa_switch.getFlag() != 1) {
            ledIndicator_ON(IND_LED1);
            ledIndicator_ON(IND_LED2);
            if (fun1Time.update()) {
                MotorXa.cRotate();
                MotorXb.ccRotate();
            }
            myController.poll();
        }
        MotorXa.motorHALT();
        MotorXb.motorHALT();
        m_Sensor_Xa.resetSensor();
        m_Sensor_Xb.resetSensor();
        break;
        /*-------------------------------------------------*/
    case CTRL_CALIBRATE_BOUNDS:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_OFF(IND_LED2);
        ledIndicator_OFF(IND_LED3);
        ledIndicator_OFF(IND_LED4);
        ledIndicator_OFF(IND_LED5);
        /* find the origin by ctrl_set_origin procedure
        * then move the Xmotor until it presses the x_switch
        * store the counter data
        * then move Ymotor until it presses the y_switch
        * store the counter data
        */
        break;
        /*-------------------------------------------------*/
        /*
        * ADD OTHER CONTROL CASES HERE...
        * ...
        */
        /*-------------------------------------------------*/
    default:
        ledIndicator_OFF(IND_LED1);
        ledIndicator_OFF(IND_LED2);
        ledIndicator_OFF(IND_LED3);
        ledIndicator_OFF(IND_LED4);
        ledIndicator_OFF(IND_LED5);
        MotorXa.motorSTOP();
        MotorXa.motorSTOP();
        MotorYa.motorSTOP();
    }

    /* Auto correcting motors if their count differ by 2 */
//int diff = m_Sensor_Xa.getCount() - m_Sensor_Xb.getCount();
```

```
//if (diff >= 2 || diff <= -2) {
//  /*
//  * if motorXa is ahead by 2 counts,
//  * then HALT motorXa, ccRotate motorXb untill count of both
//  * becomes equal
//  */
//  while (m_Sensor_Xa.getCount() > m_Sensor_Xb.getCount()) {
//      MotorXa.motorHALT();
//      MotorXb.ccRotate();
//      m_Sensor_Xa.updateData(1);
//      m_Sensor_Xb.updateData(1);
//  }
//  /* if motorXb is ahead by 2 counts,
//  * then HALT motorXb, cRotate motorXa until count of both
//  * becomes equal
//  */
//  while (m_Sensor_Xb.getCount() > m_Sensor_Xa.getCount()) {
//      MotorXb.motorHALT();
//      MotorXa.cRotate();
//      m_Sensor_Xa.updateData(1);
//      m_Sensor_Xb.updateData(1);
//  }
//}
    toggleLedIndicator(IND_LED6);
    /*-------------------------------------------------*/
}

/* ============================================================ */
/* ISR ------------------------------------*/
ISR(TIMER2_COMPA_vect) {
    //interrupt commands for TIMER 2 here, (10mS)

    /* ============================================================ */
    /*
    * Call switchObject.checkSwitch() on every switch object to update switch
flags.
    * switch flags can be then checked whenever required.
    */
    Xa_switch.checkSwitch();
    /*
    * update for limiter switch 2 added hre
    * update for limiter switch 3 added here.
    * ...
    */

    /* ============================================================ */
    /* updating counter data */
    /* counter data updated for every cases. additional cases to be added here.
*/
    switch (CONTROL_KEY){
    case CTRL_UP:
        m_Sensor_Xa.updateData(1); /* increment sensor count when motor goes
forward */
        m_Sensor_Xb.updateData(1);
        break;
    case CTRL_DOWN:
        m_Sensor_Xa.updateData(-1);
        m_Sensor_Xb.updateData(-1);
        break;
    case CTRL_RIGHT:
        m_Sensor_Ya.updateData(1);
        break;
    case CTRL_LEFT:
```

```
        m_Sensor_Ya.updateData(-1);
        break;
    }
}

/* ============================================================ */
/* EVENT ----------------------------------*/
void serialEvent1() {
    //statements
    myController.readFromClient();
}
```

M_MotorDrive.h

```
// m_MotorDrive.h

#ifndef _M_MOTORDRIVE_h
#define _M_MOTORDRIVE_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*
* PROGRAM CODE FOR MOTOR DRIVE
*/
/*=============================================================*/
/* MOTOR inits----------------------------*/

#define MXa_C_pwm   12
#define MXa_C_en    13
#define MXa_CC_pwm  11
#define MXa_CC_en   10

#define MXb_C_pwm   8
#define MXb_C_en    9
#define MXb_CC_pwm  7
#define MXb_CC_en   6

#define MYa_C_pwm   4
#define MYa_C_en    5
#define MYa_CC_pwm  3
#define MYa_CC_en   2

#define T_STATE_RESOLUTION 100
#define T_STATE_PWM_INIT_VALUE 24
#define T_STATE_PWM_INCREMENT 1
#define PWM_LIMIT 250

typedef enum motorState {
    /*
    * M_OFF: Motor is OFF State
    * M_ONCW: Motor in ON in CW direction
    * M_ONCCW: Motor is ON in CCW direction
    * M_TONCW: Motor is turning ON in CW direction
    * M_TONCCW: Motor is turning ON in CCW direction
    */
    M_OFF, M_ONCW, M_ONCCW, M_TONCW, M_TONCCW
}motorState;

/*
* class motorObject:
* @Constructor_Params: CW_EN, CCW_EN, CW_PWM_PIN, CCW_PWM_PIN
* @public_funs:
*   motorInit()    - Initialises motor by setting the pins
```

```
*   enableMotor()   - Enables the motor EN pins
*   disableMotor()  - disables the motor EN pins
*   cRotate()      - rotates motor Clock wise, with Soft start
*                 - to be used only with if else ladder
*   ccRotate()     - rotates motor Counter Clock wise, with soft start
*                 - to be used only with if else ladder
*   motorHalt()    - Stop the motor
*   setState(param) - sets the motorstate
*   getState()     - returns the motorState
*
*/
/*---------------------------------------*/
class motorObject
{
private:
    int cEnablePin; /* clockwise enable pin */
    int ccEnablePin; /* counter clockwise enable Pin */
    int cPwmPin; /* clockwise Pwm pin */
    int ccPwmPin; /* cc pwm pin */

    motorState state;

    void _motorRotate(int, int); /* analogwrite given pin */
    void _motorStop(int);   /* analogWrite given pin */

    void stop_ccRotate();   /* stop CCW */
    void stop_cRotate();    /* stop CW */

    int T_State_pwm_Increment; /* increment value of pwm */
    int pwmValue;

public:

    motorObject(int, int, int, int);/*cEN, ccEN, cPwm, ccPwm*/

    void motorInit();   /* Motor Initialisation */
    void enableMotor();   /* Motor ENABLE */
    void disableMotor() {   /* Motor DISABLE */
        digitalWrite(cEnablePin, LOW);
        digitalWrite(ccEnablePin, LOW);
    }
    void cRotate(); /* Rotate CW */
    void ccRotate();    /* Rotate CCW */
    void motorHALT();   /* stop motor */

    void motorSTOP();

    void setState(motorState _state) {
        state = _state;
    }
    int getState() {
        return state;
    }


};
/*---------------------------------------*/
/*=======================================================================*/
```

```
extern motorObject MotorXb;
extern motorObject MotorXa;
extern motorObject MotorYa;

#endif
```

M_MotorDrive.cpp

```
//
//

#include "m_MotorDrive.h"

/*---------------------------------------*/
/*constructor*/
motorObject::motorObject(int _cEnablePin, int _ccEnablePin, int _cPwmPin,
    int _ccPwmPin)
{
    cEnablePin = _cEnablePin;
    ccEnablePin = _ccEnablePin;
    cPwmPin = _cPwmPin;
    ccPwmPin = _ccPwmPin;
    T_State_pwm_Increment = T_STATE_PWM_INCREMENT;

    pwmValue = 0;
    state = M_OFF;
}
/*---------------------------------------*/
/* Init */
void motorObject::motorInit() {
    pinMode(cEnablePin, OUTPUT);
    pinMode(ccEnablePin, OUTPUT);
    pinMode(cPwmPin, OUTPUT);
    pinMode(ccPwmPin, OUTPUT);
    motorHALT();
}


/*---------------------------------------*/
/* rotate cw */
/*
* the switch is pressed,  and this function is called.
* if the motor current state is anything other than M_ONCW or M_TONCW,
* then motor is set to M_TONCW. then pwmValue is incremented until it
* reaches PWM_LIMIT. then motor state is set to M_ONCW. i.e Motor is fully ON
*/
void motorObject::cRotate() {
    if (state == M_TONCW) {
        pwmValue += 1;
        if (pwmValue > PWM_LIMIT) {
            pwmValue = PWM_LIMIT;
            state = M_ONCW;
        }
    }
    else if (state == M_ONCW) {

    }
    else {
        state = M_TONCW;
        pwmValue = 0;
    }
```

```
    enableMotor();
    /* if CW rotation is ON, then CCW rotation should be OFF*/
    stop_ccRotate();
    _motorRotate(cPwmPin, pwmValue);
}

/*--------------------------------------*/
/* stop cw */
void motorObject::stop_cRotate() {
    _motorStop(cPwmPin);
}


/*--------------------------------------*/
/* rotate ccw */
void motorObject::ccRotate() {


    if (state == M_TONCCW) {
        pwmValue += 1;
        if (pwmValue > PWM_LIMIT) {
            pwmValue = PWM_LIMIT;
            state = M_ONCCW;
        }
    }
    else if (state == M_ONCCW) {

    }
    else {
        state = M_TONCCW;
        pwmValue = 0;
    }

    enableMotor();
    stop_cRotate();
    _motorRotate(ccPwmPin, pwmValue);
}

/*--------------------------------------*/
/* stop ccw */
void motorObject::stop_ccRotate() {
    _motorStop(ccPwmPin);
}
/*--------------------------------------*/
/* motor rotate */
/* TODO: Slow start */
void motorObject::_motorRotate(int pin, int val) {
    analogWrite(pin, val);
}

/*--------------------------------------*/
/* motor turning off */
/* TODO: Slow stop */
void motorObject::_motorStop(int pin) {
    analogWrite(pin, 0);
}

/*--------------------------------------*/
```

```
/* Sudden stop motor */
/*
* Motor state is reset ONLY when motorHALT() is called.
*/
void motorObject::motorHALT() {
    pwmValue = 0;
    state = M_OFF;
    analogWrite(cPwmPin, 0);
    analogWrite(ccPwmPin, 0);
}

void motorObject::motorSTOP() {
    disableMotor();
    state = M_OFF;
}

/*-----------------------------------------*/
void motorObject::enableMotor() {
    pinMode(ccEnablePin, OUTPUT);
    pinMode(cEnablePin, OUTPUT);
    pinMode(ccPwmPin, OUTPUT);
    pinMode(cPwmPin, OUTPUT);
    digitalWrite(cEnablePin, HIGH);
    digitalWrite(ccEnablePin, HIGH);
}
/*-----------------------------------------------------------------------*/

motorObject MotorXb(MXb_C_en, MXb_CC_en, MXb_C_pwm, MXb_CC_pwm);
motorObject MotorXa(MXa_C_en, MXa_CC_en, MXa_C_pwm, MXa_CC_pwm);
motorObject MotorYa(MYa_C_en, MYa_CC_en, MYa_C_pwm, MYa_CC_pwm);
```

M_ModbusControl.h

```
// m_modBusComm.h

#ifndef _M_MODBUSCOMM_h
#define _M_MODBUSCOMM_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*
* PROGRAM CODE FOR COMMUNICATION USING MODBUS
*/

#include <ArduinoRS485.h>
#include <ArduinoModbus.h>


#define SLAVE_ID 1
#define BAUDRATE 19200

#define HRNUM 10
#define HRADD 0x00

#define NUM_CLIENT_DATA 7

#define CTRL_UP 1
#define CTRL_RIGHT 2
#define CTRL_DOWN 3
#define CTRL_LEFT 4
#define CTRL_SET_ORIGIN 10
#define CTRL_CALIBRATE_BOUNDS 11

/*================================================================*/
/* MODBUS inits----------------------------*/
class myClass {
private:

    int ctrlValue;
    int ctrlData; //UP, RIGHT, DOWN, LEFT Data from Control Input

    int slaveID;  //slave ID of the device
    int baudRate;
    int noOfHR; //number of holding registers required in MODBUS comm


public:
    int toClientData[10];
    int no_ClientData;

    myClass(int, int, int);
    bool connectToClient();
```

```
    void readFromClient();

    int getCtrlData() {
        return ctrlData;
    }

    void poll() {
        ModbusRTUServer.poll();
    }

    void sendDataToClient();
};

/*-----------------------------------*/
/*===================================================================*/
extern myClass myController;

#endif
```

M_ModbusControl.cpp

```
#include "m_modBusComm.h"

/*================================================================*/

/*
   Constructor class
   @params: slaveId, BaudRate, noOfHoldingRegsRequired
*/
myClass::myClass(int ID = 1, int baud_Rate=9600, int no_of_HR=8) {
    slaveID = ID;
    baudRate = baud_Rate;
    noOfHR = no_of_HR;
    ctrlValue = 0;
    ctrlData = 0;
    no_ClientData = NUM_CLIENT_DATA;
}

/*
    Connect to client
*/

bool myClass::connectToClient() {
    if (!ModbusRTUServer.begin(slaveID, baudRate)) {
        return false; //failed to connect
    }
    // configure holding register address
    ModbusRTUServer.configureHoldingRegisters(HRADD, noOfHR);

    return true; //connected
}

/*
  Accepts data from Client to local registers
*/
void myClass::readFromClient() {
    ctrlValue = ModbusRTUServer.holdingRegisterRead(HRADD);
    ctrlData = ctrlValue;
}

/*
writes data to holding registers
*/


void myClass::sendDataToClient() {
    int starting_Add = 1;
    for (int i = 0; i < 8; i++) {
        ModbusRTUServer.holdingRegisterWrite(starting_Add + i, toClientData[i]);
    }
}

myClass myController(SLAVE_ID, BAUDRATE, HRNUM);
```

M_SensorControl.h

```
// m_modBusComm.h

#ifndef _M_MODBUSCOMM_h
#define _M_MODBUSCOMM_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*
* PROGRAM CODE FOR COMMUNICATION USING MODBUS
*/

#include <ArduinoRS485.h>
#include <ArduinoModbus.h>


#define SLAVE_ID 1
#define BAUDRATE 19200

#define HRNUM 10
#define HRADD 0x00

#define NUM_CLIENT_DATA 7

#define CTRL_UP 1
#define CTRL_RIGHT 2
#define CTRL_DOWN 3
#define CTRL_LEFT 4
#define CTRL_SET_ORIGIN 10
#define CTRL_CALIBRATE_BOUNDS 11

/*============================================================*/
/* MODBUS inits-----------------------------*/
class myClass {
private:

    int ctrlValue;
    int ctrlData; //UP, RIGHT, DOWN, LEFT Data from Control Input

    int slaveID;  //slave ID of the device
    int baudRate;
    int noOfHR; //number of holding registers required in MODBUS comm


public:
    int toClientData[10];
    int no_ClientData;

    myClass(int, int, int);
    bool connectToClient();
    void readFromClient();
```

```
    int getCtrlData() {
        return ctrlData;
    }

    void poll() {
        ModbusRTUServer.poll();
    }

    void sendDataToClient();
};

/*-----------------------------------------*/
/*=========================================================================*/
extern myClass myController;

#endif
```

M_sensorControl.cpp

```
//
//
//

#include "m_sensorControl.h"

/*=============================================================*/
/* constructor */
ir_Sensor::ir_Sensor(int pinNo) {
    pin = pinNo;
    memset(raw_SensorVal, 0, sizeof(raw_SensorVal));
    arrayMagnitude = 0;
    counter = 0;
    stateChangeDetected = 0;
}

/*-------------------------------------------------------*/
/* initialisation */
bool ir_Sensor::initSensor() {
    pinMode(pin, INPUT_PULLUP);
    arrayMagnitude = 0;
    for (int i = 0; i < IR_BUFF_SIZE; i++){
    raw_SensorVal[i] = digitalRead(pin);
    Serial.println(raw_SensorVal[i]);
    if (raw_SensorVal[i] == IR_SENSOR_DETECTED)arrayMagnitude++;
    delay(10);
    }
    Serial.println();

    /* change state by checking buffer */
    if (arrayMagnitude > IR_NOISE_HIGH_THRESH) { //taking sample of initial
state and storing the state
        initialSensorState = true; /* IR SENSOR DETECTED STATE */
        //SensorState = initialSensorState;
        return true;   /* returns true when detected or not detected value */
    }
    else if (arrayMagnitude < IR_NOISE_LOW_THRESH) {
        initialSensorState = false; /* IR SENSOR NOT DETECTED STATE */
        //SensorState = initialSensorState;
        return true;
    }
    else return false; /* returns false when high noise */
}

/*-------------------------------------------------------*/
/* read and update buffer and checks magnitude and update state */
void ir_Sensor::updateData(int inc) {

    arrayMagnitude = 0;
    for (int i = 0; i < (IR_BUFF_SIZE - 1); i++) {
        raw_SensorVal[i] = raw_SensorVal[i + 1];
        /* if sensor moves from initial state */
        if (raw_SensorVal[i] != initialSensorState)arrayMagnitude++;
```

```
    }
    raw_SensorVal[IR_BUFF_SIZE - 1] = digitalRead(pin);

    /* change state by checking buffer */
    /* if sensor reading current state isn't initial state, then increment counter
*/
    if (arrayMagnitude > IR_NOISE_HIGH_THRESH) {  //taking sample of initial
state and storing the state
        if (SensorState == initialSensorState) {
            SensorState = !(initialSensorState);
            counter += inc;
        }
    }
    else if (arrayMagnitude < IR_NOISE_LOW_THRESH) {
        SensorState = initialSensorState;
    }

    //if (counter > IR_COUNTER_LIMIT) { counter = IR_COUNTER_LIMIT; }
    //if (counter < IR_COUNTER_LIMIT) { counter = IR_COUNTER_LIMIT; }
}

//void ir_Sensor::updateData(int inc) {
//  static int switch_pressed = 0;
//  static int switch_notPressed = 0;
//
//  if (digitalRead(pin) != initialSensorState) {
//      switch_notPressed = 0;
//      switch_pressed++;
//      if (switch_pressed > ) {
//          switch_pressed = 100;
//          counter += inc;
//      }
//  }
//  else {
//      switch_pressed = 0;
//  }
//
//}
/*-------------------------------------------------------*/
/* resets sensor datas */

void ir_Sensor::resetSensor() {
    memset(raw_SensorVal, 0, sizeof(raw_SensorVal));
    arrayMagnitude = 0;
    counter = 0;
    stateChangeDetected = 0;
    //initSensor();
}




/*=============================================================*/
ir_Sensor m_Sensor_Xa(SENSOR1_PIN);
ir_Sensor m_Sensor_Xb(SENSOR2_PIN);
ir_Sensor m_Sensor_Ya(SENSORY_PIN);
```

M_InterruptService.h

```
// m_InterruptService.h

#ifndef _M_INTERRUPTSERVICE_h
#define _M_INTERRUPTSERVICE_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*================================================*/
extern int timer100mS;

void setTimerInterrupt();

/*================================================*/

#endif
```

M_InterruptService.cpp

```
//
//
//

#include "m_InterruptService.h"

/* 10mS timer */

void setTimerInterrupt() {
    noInterrupts();
    // Clear registers
    TCCR2A = 0;
    TCCR2B = 0;
    TCNT2 = 0;

    // 100.16025641025641 Hz (16000000/((155+1)*1024))
    OCR2A = 155;
    // CTC
    TCCR2A |= (1 << WGM21);
    // Prescaler 1024
    TCCR2B |= (1 << CS22) | (1 << CS21) | (1 << CS20);
    // Output Compare Match A Interrupt Enable
    TIMSK2 |= (1 << OCIE2A);
    interrupts();
}
```

M_LedIndicator.h

```
// m_LEDIndicator.h

#ifndef _M_LEDINDICATOR_h
#define _M_LEDINDICATOR_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*=========================================================*/
#define DELAY_LED 200
#define DELAY_LED_2 100

#define IND_LED1 A8
#define IND_LED2 A9
#define IND_LED3 A10
#define IND_LED4 A11
#define IND_LED5 A12
#define IND_LED6 A13
#define IND_LED7 A14
#define IND_LED8 A15

/*---------------------------------------------------------*/
/* ADD LED INDICATORS HERE ----------------------------------*/
#define motorCW_LED IND_LED5
#define motorCCW_LED IND_LED6
#define timerInterrupt_LED IND_LED1
/*---------------------------------------------------------*/

void setupLEDIndicators();
void ledIndicator_ON(uint8_t);
void ledIndicator_OFF(uint8_t);
void blinkLedIndicator(uint8_t);
void toggleLedIndicator(uint8_t);
void errorIndicator();
/*=========================================================*/

#endif
```

M_LedIndicator.cpp

```
//
//
//

#include "m_LEDIndicator.h"


uint8_t ledIndicatorArray[] = {
    IND_LED1,
    IND_LED2,
    IND_LED3,
    IND_LED4,
    IND_LED5,
    IND_LED6,
    IND_LED7,
    IND_LED8
};

void setupLEDIndicators() {
    int NO_OF_INDICATORS=sizeof(ledIndicatorArray);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        pinMode(ledIndicatorArray[i], OUTPUT);
    }

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], LOW);
    }
    delay(DELAY_LED);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], HIGH);
    }
    delay(DELAY_LED);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], LOW);
    }
    delay(DELAY_LED);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], HIGH);
        delay(DELAY_LED_2);
    }

    delay(DELAY_LED_2);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], LOW);
        delay(DELAY_LED_2);
    }
```

```
    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], HIGH);
    }
    delay(DELAY_LED);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], LOW);
    }
}

void errorIndicator() {

    int NO_OF_INDICATORS = sizeof(ledIndicatorArray);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        pinMode(ledIndicatorArray[i], OUTPUT);
    }
    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], LOW);
    }
    delay(DELAY_LED);

    for (int i = 0; i < NO_OF_INDICATORS; i++) {
        digitalWrite(ledIndicatorArray[i], HIGH);
    }
    delay(DELAY_LED);
}

void ledIndicator_ON(uint8_t pinNo) {
    digitalWrite(pinNo, HIGH);
}

void ledIndicator_OFF(uint8_t pinNo) {
    digitalWrite(pinNo, LOW);
}

void blinkLedIndicator(uint8_t pinNo) {
    digitalWrite(pinNo, HIGH);
    delay(1);
    digitalWrite(pinNo, LOW);
}

void toggleLedIndicator(uint8_t pinNo) {
    digitalWrite(pinNo, !(digitalRead(pinNo)));
}
```

M_Switches.h

```
// m_switches.h

#ifndef _M_SWITCHES_h
#define _M_SWITCHES_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif

/*=====================================================*/
/*
* define pins for limiter switches here.
*/
#define Xa_limSwitchPin 25
/*
* pin definition for limiter switch 2 here
* pin definition for limiter switch 3 here.
* ...
*/

/* ============================================================ */
/*
* create objects for every limiter switches.
* for avoiding noise effects.
*
* @Constructor: switchInput(PINNUMBER) creates switch object on given pin
*
* @init() initialises the given pinnumber as input
*
* @checkSwitch() should be called on timer for analysing consecutive readings and
* decide switch position
*
* @getFlag() returns the current pin Status. 1= pressed, (active LOW), 0= not pressed
(HIGH CONDITION)
*
*/

class switchInput {
private:
    uint8_t pin;
    uint16_t countHigh;
    uint16_t countLow;
    uint8_t Flag;
public:
    switchInput(int _pin) {
        pin = _pin;
        countHigh = 0;
        countLow = 0;
        Flag = 0;   //low
    }
    /*-----------------------------------------------*/
```

```
    void init() {
        pinMode(pin, INPUT_PULLUP);
    }
    /*-----------------------------------------------*/
    void checkSwitch() {
        if (digitalRead(pin) == HIGH) {
            countLow = 0;
            countHigh++;
            if (countHigh > 5) {
                countHigh = 5;
                Flag = 0;
            }
        }
        else {
            countHigh = 0;
            countLow++;
            if (countLow > 5) {
                countLow = 5;
                Flag = 1;  /* Flag set on Active LOW*/
            }
        }
    }
    /*-----------------------------------------------*/
    uint8_t getFlag() {
        return Flag;
    }
};

/* ============================================================ */
/* creating switch objects for all limiter switches */
switchInput Xa_switch(Xa_limSwitchPin);
/*
* create object for limiter switch 2
* create object for limtier switch 3
*/

/*=================================================*/


#endif
```

8.Software development for PC (for Control and Monitoring) USING LABVIEW

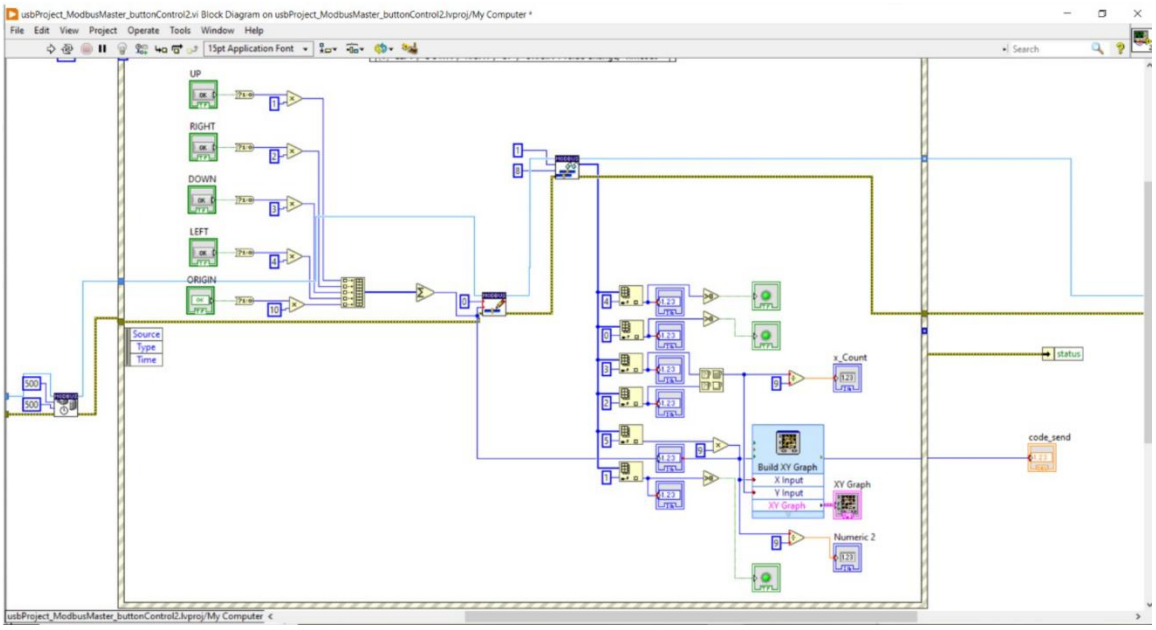
Objectives:

1- User has to control motors in different directions

2- User has to view the motor position relative to pond

The PC which is used to control the motors are taken as server, and the Arduino is taken as modbus Client.

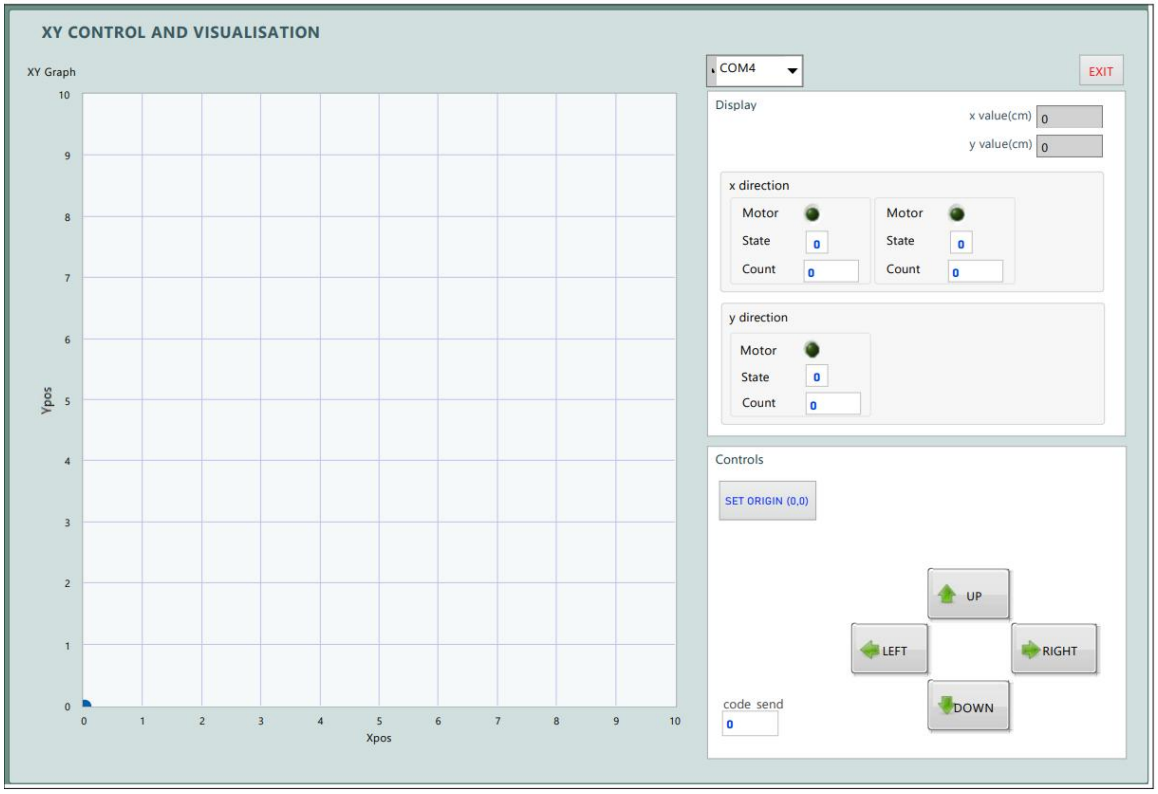
Block Diagram



Here when a button is pressed, the corresponding control value is stored in the modbus holding registers, which is then transmitted to Arduino.

Also, PC receives the data from Arduino through modbus, and the corresponding registers are read and interpreted in the software. These data are used to show the motor position, status and other values.

User Interface



1- User can connect Modbus to required port using the drop down box.

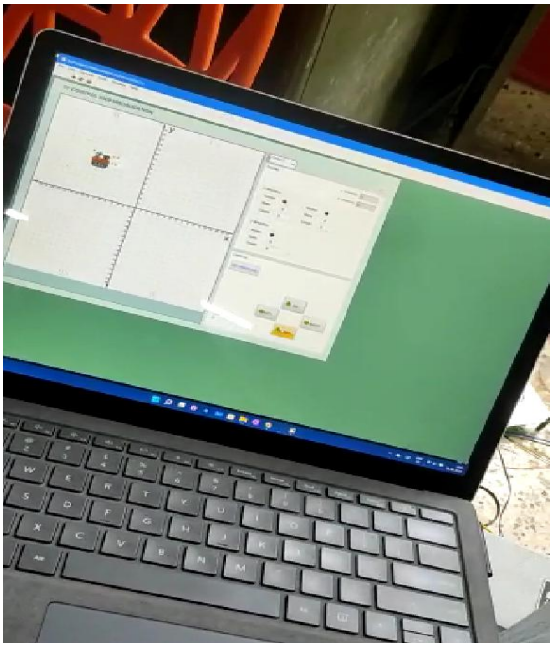
2- User can send commands to Arduino using different buttons, like UP, DOWN, RIGHT, LEFT…

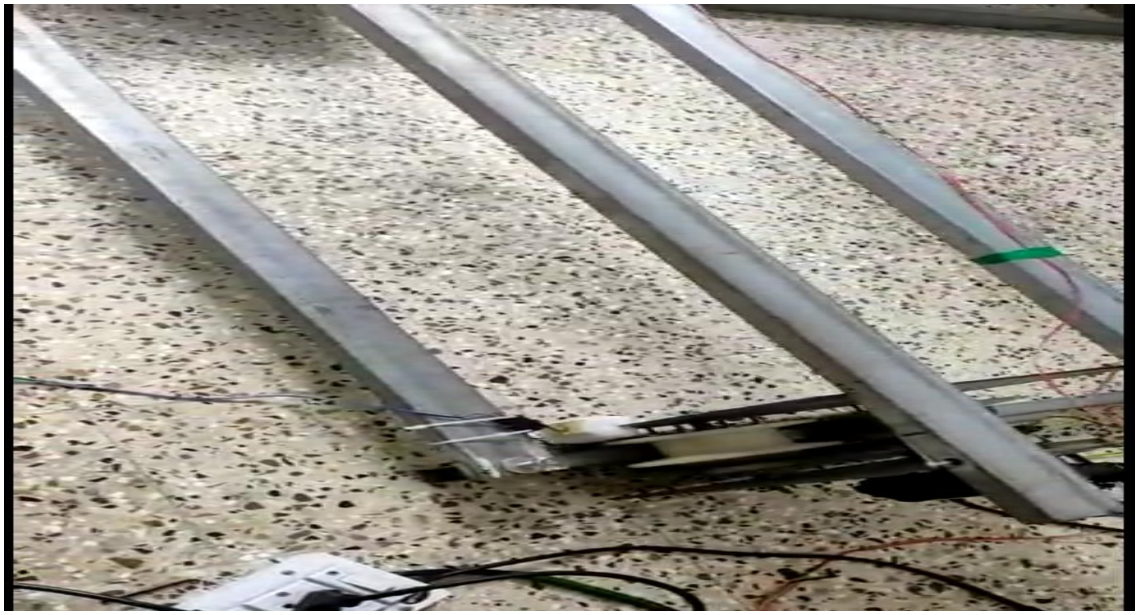
3- The send command Value will be shown for troubleshooting purposes

4- Individual motor running states are also made available to user.

5- The Grid layout takes the Motor Position data from arduino and shows the Probs relative position.

9.FINISHED PRODUCT





Project has been successfully completed