## References


[1] W. O. Stevens and A. Westcott, revolution in naval warfare, in A history of sea power, GHD co, 1920, ch. XIV, pp

[2] A.B. Menahem, Evolution of Submarine Design, in Historical Encyclopedia of Nautical and Mathematical sciences, vlo 2, NY, Springer, 2009, ch5. Pp 4707

[3] R. Burcher and L Rydill, Concepts in Submarine Design, Series 2, Cambridge Ocean Technology, UK, 1994, pp 20-23

[4] Williams, D.; Folbert, A., "Energy considerations for a long range diesel electric submarine," *Electric Ship Technologies Symposium (ESTS), 2011 IEEE* , vol., no., pp.406,413, 10-13 April 2011 doi: 10.1109/ESTS.2011.5770906

[5] Martin, M., "Technological evolution [submersibles]," *OCEANS '02 MTS/IEEE* , vol.3, no., pp.1330,1331 vol.3, 29-31 Oct. 2002 doi: 10.1109/OCEANS.2002.1191830

[6] Ishibashi, S.; Yoshida, H.; Hyakudome, T., "Light duty works performed by an underwater vehicle," *Underwater Technology (UT), 2011 IEEE Symposium on and 2011 Workshop on Scientific Use of Submarine Cables and Related Technologies (SSC)* , vol., no., pp.1,6, 5-8 April 2011 doi: 10.1109/UT.2011.5774102.

[7] Askew, T., "Submersibles for Science - JOHNSON-SEA-LINK I & II," *OCEANS 1984* , vol., no., pp.612,616, 0-0 Sept. 1984 doi: 10.1109/OCEANS.1984.1152263.

[8] T. I. Fossen, Introduction, in Handbook of marine craft hydrodynamics and motion control, ed 1st, JWS, West Sussex, UK, 2011, ch 1, part 1.

[9] Code on intact stability for all types of ships covered by IMO instruments, Resolution A.749(18), 4 November 1993

[10] Basic Stability for Small Vessels, The Shipowners' Protection Ltd, UK, November 2007, pp 2-23

[11] B. Lautrup, Physics of Continuous Matter, 2nd ed, Boka Rator, FL, CRC, 2011, ch 3, sec 3.1

[12] C.B. Barras and D.R. Derrett, Ship stability for masters and mates, 7th ed, Waltham, USA, 2012, ch 2, pp 15

[13] P.N Joubert, Some Aspects of Submarine Design, Part 1 Hydrodynamics, Victoria, Australia, DSTO, 2004, pp 9-10

[14] C. Aage and S. L. Wagner, Hydrodynamic manoeuvrability data of a flatfish type AUV, In, Oceans, vol 3, 1994

[15] P.N Joubert, Some Aspects of Submarine Design, Part 2 Hydrodynamics, Victoria, Australia, DSTO, 2006, pp 2

[16] J. Vaganay et al., Ship hull inspection by hull-related navigation and control, in Oceans, 2005 © Proceedings of MTS/IEEE. Doi: 10.1109/OCENANS.2005.1639844

[17] W. Wang et al., Hybrid Intelligent Control for Submarine Stabilization, in International Journal of Advanced Robotic Systems, 2013 © 2013 Wang et al; licensee InTech. Doi:10.5772/56392

[18] W. Naeem, Model predictive control of an autonomous underwater vehicle, presented at the Proceedings of UKACC 2002 postgraduate symposium, Sheffield, 2002



[19] J. Eskesen et al, Design and performance of Odyssey IV: A deep ocean hover-capable AUV, MIT Sea grant college program report, Cambridge, MA, Rep MIT Sea Grant Project No. 2005-R/RCM-14, 2009

[20] Akcakaya et al (2002), Sliding mode control of autonomous underwater vehicle [online]. Available: http://www.emo.org.tr/ekler/e118a0a734b2caf_ek.pdf

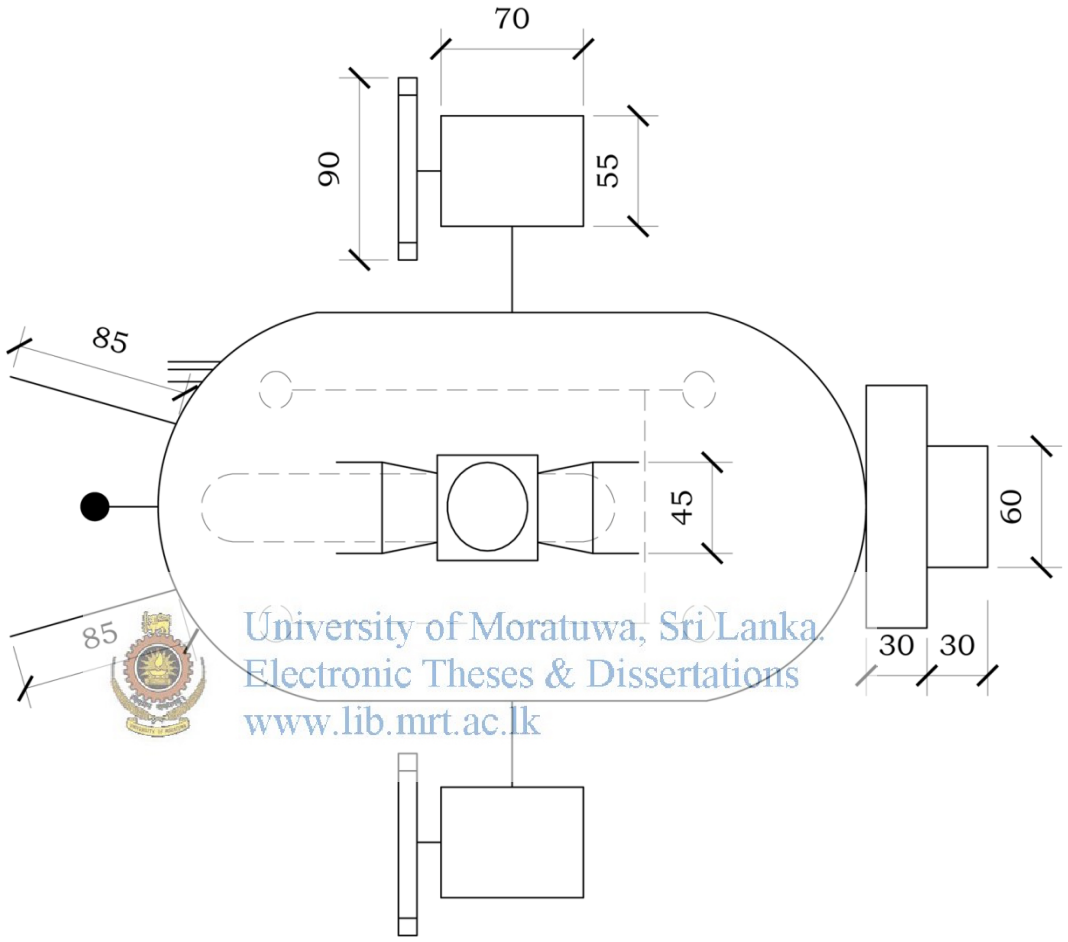
[21] T. I. Fossen, in Guidance and control of ocean vehicles, West Sussex, England, John Wiley & Sons Ltd, 1994, ch. 2, sec. 2.4, pp 29 – 40.

[22] Nomenclature for treating the motion of a submerged body through a fluid, The society of Naval Architects and Marine Engineers, Technical and Research Bulletin No. 1-5, April 1950, pp 1-15.

[23] M. E. Rentchler et al, System identification of open-loop maneuvers leads to improved AUV flight performance, IEEE journal of Oceanic Engineering, vol 31, pp 200-2008, Jan. 2006

[24] R.D, Christ et al, ROV components, in The ROV Manual: A user guide for observation class remotely operated vehicle, 1 ed. Oxford, MA: BH, 2007, ch 3, pp 46-71

# Top View

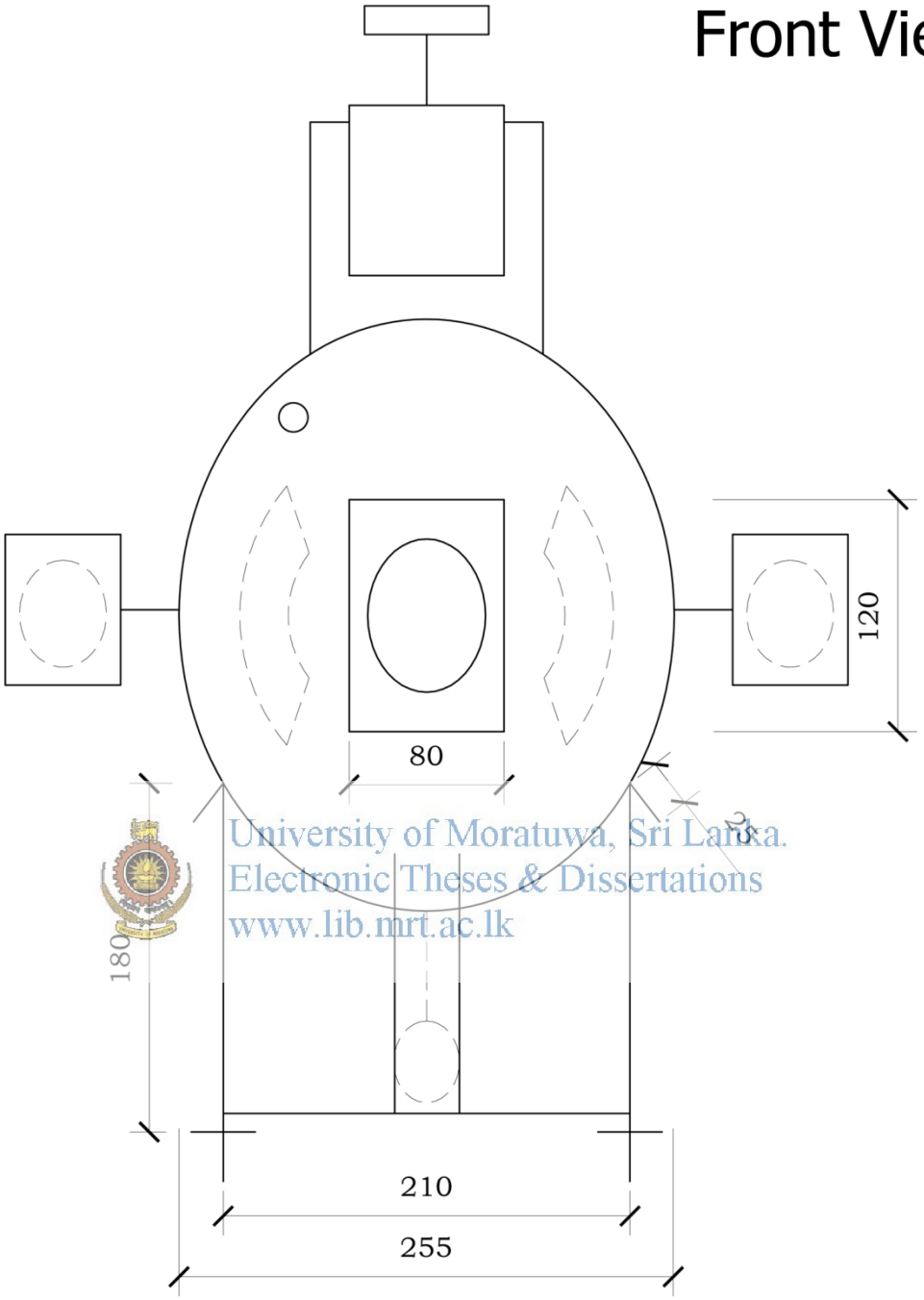





# DRAWING -01

SCALE - 1:10

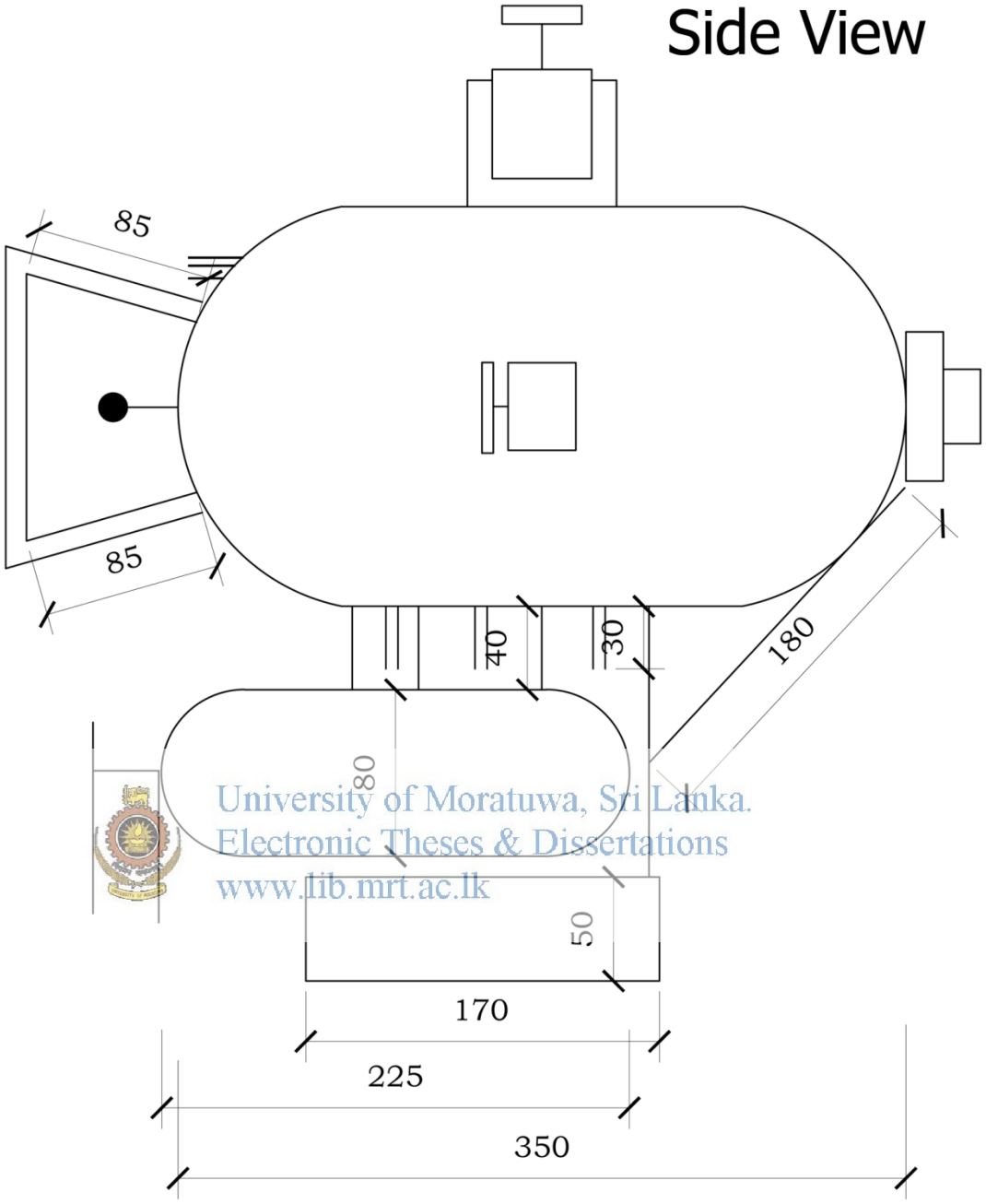
Front View






# DRAWING -02

SCALE - 1:10






# DRAWING -03

SCALE - 1:10

## Mbed Codes used

### 1. Main program

```
#include "mbed.h"
#include "MS5803.h"
#include "string"
#include <stdio.h>

# define DENSITY 1000
# define GRAVITY 9.8

PwmOut pwm_M1_clk(p21);      // clockwise rotation pwm pin for M1
PwmOut pwm_M1_anticlk(p22);  // anti - clockwise rotation pwm pin for M1
PwmOut pwm_M2_clk(p23);     // clockwise rotation pwm pin for M2
PwmOut pwm_M2_anticlk(p24);      // anti-clockwise rotation pwm pin for M2
PwmOut pwm_M3M4_clk(p25);
PwmOut pwm_M3M4_anticlk(p26);
DigitalIn safety_1(p13);
DigitalIn safety_2(p16);
AnalogIn sol(p15);
DigitalOut M3_clk(p5);
DigitalOut M4_clk(p6);
DigitalIn abc1(p7);
DigitalIn abc2(p8);
DigitalOut M3_anticlk(p29);
DigitalOut M4_anticlk(p30);
AnalogIn position1(p17);
AnalogIn position2(p18);
DigitalIn position3(p19);
DigitalIn position4(p20);
DigitalIn auto_pilot(p11);
DigitalOut sol1(p12);
DigitalOut sol2(p13);
DigitalOut sol3(p14);
I2C i2c_gyro(p9, p10);
I2C i2c_pres(p28, p27);

const int addr = 0xD0; //gyro

//customs variables
```

```
int MANUAL_OPS = true; //true means operate manually, false mean auto pilot
float my_pressure=0;
float my_depth=0;
float my_angle=0;
float depth = 0;
int depth_val = 0;
float test1 = 0;
float test2 = 0;

Serial pc(USBTX, USBRX);
MS5803 press_sensor(p28,p27,ms5803_base_addr); // sda, scl, I2C_address 0x76 or
0x77 look at MS5803.h

float read_gyro(void);
float read_pressure(void);
float read_depth(float pressure);
string find_gyro_sign(float my_gyro);
int main()
{
   pwm_M1_clk.period_us(10);
   pwm_M1_anticlk.period_us(10);
   pwm_M2_clk.period_us(10);
   pwm_M2_anticlk.period_us(10);
   pwm_M3M4_clk.period_us(100);
   pwm_M3M4_anticlk.period_us(100);

   pwm_M1_clk = 0.0;
   pwm_M1_anticlk = 0.0;
   pwm_M2_clk = 0.0;
   pwm_M2_anticlk = 0.0;
   pwm_M3M4_clk = 0.0;
   pwm_M3M4_anticlk = 0.0;



   while (1) {
      //test pressure sensor
      my_pressure = read_pressure();
      //pc.printf("%4.1f \r\n", my_pressure);
      my_depth = read_depth(my_pressure);
      //pc.printf("%4.1f \r\n", my_depth);
```

```
//pc.printf("%4.1f \r\n", my_pressure);
if(auto_pilot == 0) {


   if(position1.read()>0.9) {
      pwm_M1_anticlk= 0.2;


   } else {
      pwm_M1_anticlk= 0.0;

   }

   if(position1.read()==0.0) {
      pwm_M1_clk= 0.2;


   } else {
      pwm_M1_clk= 0.0;

   }

if(position2.read()>0.9) {
   pwm_M2_clk= 0.9;
} else {
   pwm_M2_clk= 0.0;

   }

   if(position2.read()==0.0) {
      pwm_M2_anticlk= 0.9;

   } else {
      pwm_M2_anticlk= 0.0;
   }

   if(abc1==1) {
      // M4  and M3 clk
      pwm_M3M4_clk = 0.9;
      M3_anticlk = 0;
      M4_clk = 1;
```

```
    M4_anticlk = 0;
    M3_clk = 1;

}

if(abc2==1) {
    // M4  and M3 anticlk
    pwm_M3M4_anticlk = 0.9;
    M3_anticlk = 1;
    M4_clk = 0;
    M4_anticlk = 1;
    M3_clk = 0;
}

if(sol.read()==0.0) {
    sol3=0;
} else if (sol.read()>0.9) {
    sol3=1;
}

if(position3==1) {
    // M4 clk and M3 anticlk
    pwm_M3M4_clk = 0.9;
    pwm_M3M4_anticlk = 0.9;
    M3_anticlk = 1;
    M4_clk = 1;
    M4_anticlk = 0;
    M3_clk = 0;
}


if(position4==1) {
    // M4 clk and M3 anticlk
    pwm_M3M4_clk = 0.9;
    pwm_M3M4_anticlk = 0.9;
    M3_anticlk = 0;
    M4_clk = 0;
    M4_anticlk = 1;
    M3_clk = 1;

}
```

```
            if((abc1== 0)&&(abc2 ==0)&&(position3==0)&&(position4==0)) {
               pwm_M3M4_clk = 0.0;
               pwm_M3M4_anticlk = 0.0;
               M3_anticlk = 0;
               M4_clk = 0;
               M4_anticlk = 0;
               M3_clk = 0;
            }



         } // end if manual
         if(auto_pilot == 1) {

            /*printf ("Enter Depth Value: ");
            scanf ("%d",&depth_val);*/
            pwm_M2_anticlk= 0.9;
            if(my_depth >= 10) {
               pwm_M2_anticlk= 0.0;
               float my_gyro = read_gyro();
               string gyro_sign = find_gyro_sign(my_gyro);
               if(gyro_sign == "plus") {

               }
               if(gyro_sign == "minus") {

               }
               if (gyro_sign == "zero") {

               }
            }
         }// end if auto pilot
      } // end while
   }// end main



   float read_pressure()
   {
      //pressure sensor
```

```
    //wait( 3 );
    //pc.printf("MS5803 demo");
#ifdef debug
#endif
    press_sensor.Barometer_MS5803();
    // pc.printf("%4.1f (mBar)\r\n", press_sensor.MS5803_Pressure());
    //pc.printf("%4.1f (deg C)\r\n", press_sensor.MS5803_Temperature());
    //wait( 1 );
    return press_sensor.MS5803_Pressure();
}

float read_depth(float pressure)
{

    pressure = pressure/100;
    depth = pressure/(DENSITY*GRAVITY);
    //pc.printf("%f (depth m)\r\n", depth);
    //depth = depth *1000000;
    pc.printf("%d (depth m)\r\n", depth);
    //depth = depth + 1;
    return depth;
}

float read_gyro()
{
    char cmd[2];
    cmd[0] = 0x8c;

    cmd[1] = 0x00;
    i2c_gyro.write(addr, cmd, 2);
    wait(0.5);
    cmd[0] = 0x8c;

    i2c_gyro.write(addr, cmd, 1);
    i2c_gyro.read(addr, cmd, 2);

    float gyro_value = (float((cmd[0]<<8)|cmd[1]) / 256.0);
    //printf("tilt = %.2f\n", gyro_value);
    return gyro_value;
}
```

```
string find_gyro_sign(float my_gyro)
{
   string gyro_sign;
   int  integral, binaryInt = 0, i = 1;
   float  binaryFract = 0, k =0.1f, fractional, temp1, binaryTtotal;

   //Separating the integral value from the floating point variable
   integral = (int)my_gyro;

   //Separating the fractional value from the variable
   fractional = my_gyro - (int)my_gyro;

   while(integral>0) {
      binaryInt = binaryInt + integral % 2 * i;
      i = i * 10;
      integral = integral / 2;
   }
//Loop for converting Fractional value to binary
   while(k>0.00000001) {
      temp1 = fractional *2;
      binaryFract = binaryFract+((int)temp1)*k;
      fractional = temp1 - (int)temp1;
      k = k / 10;
   }

   //Combining both the integral and fractional binary value.
   binaryTotal = binaryInt +binaryFract;
   printf(" \nbinary equivalent = %f\n\n\n\n\n", binaryTotal);
   gyro_sign = "plus";
   return gyro_sign;
}
```



## 2. Header program

```
/*
```

**Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is**

```
   furnished to do so, subject to the following conditions:

   The above copyright notice and this permission notice shall be included in
   all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
   THE SOFTWARE.

 * Barometer Sensor (Altimeter) MS5803-01BA of MEAS Switzerland (www.meas-spec.com).
 * The driver uses I2C mode (sensor PS pin low).
 * Other types of MEAS are compatible but not tested.
 * Written by Raig Kaufer distribute freely!
 */
#include "mbed.h"


#ifndef MS5803_H
#define MS5803_H

#define MS5803_RX_DEPTH 3
#define MS5803_TX_DEPTH 2


#define ms5803_addrCL 0x77 //0b1110111  CSB Pin is low
#define ms5803_addrCH 0x76 //0b1110110  CSB Pin is high

#define ms5803_base_addr ms5803_addrCH // choose your connection here

#define ms5803_reset       0x1E // Sensor Reset
```

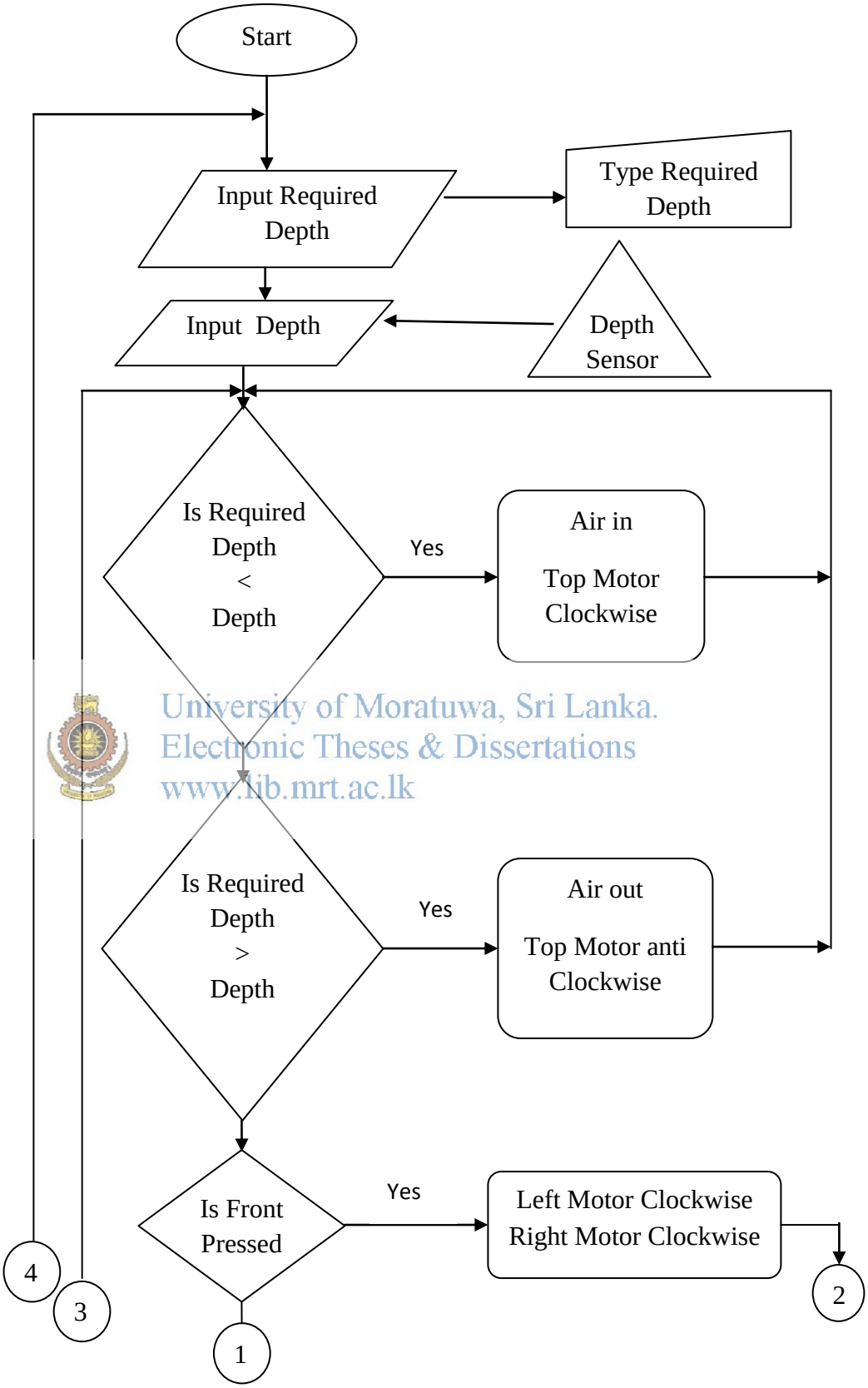
```
#define ms5803_convD1_256  0x40 // Convert D1 OSR 256
#define ms5803_convD1_512  0x42 // Convert D1 OSR  512
#define ms5803_convD1_1024 0x44 // Convert D1 OSR 1024
#define ms5803_convD1_2048 0x46 // Convert D1 OSR 2048
#define ms5803_convD1_4096 0x48 // Convert D1 OSR 2048

#define ms5803_convD1 ms5803_convD1_4096 // choose your sampling rate
here

#define ms5803_convD2_256  0x50 // Convert D2 OSR  256
#define ms5803_convD2_512  0x52 // Convert D2 OSR  512
#define ms5803_convD2_1024 0x54 // Convert D2 OSR 1024
#define ms5803_convD2_2048 0x56 // Convert D2 OSR 2048
#define ms5803_convD2_4096 0x58 // Convert D2 OSR 2048

#define ms5803_convD2 ms5803_convD2_4096 // choose your sampling rate
here

#define ms5803_ADCread     0x00 // read ADC command
#define ms5803_PROMread    0xA0 // read PROM command base address
```



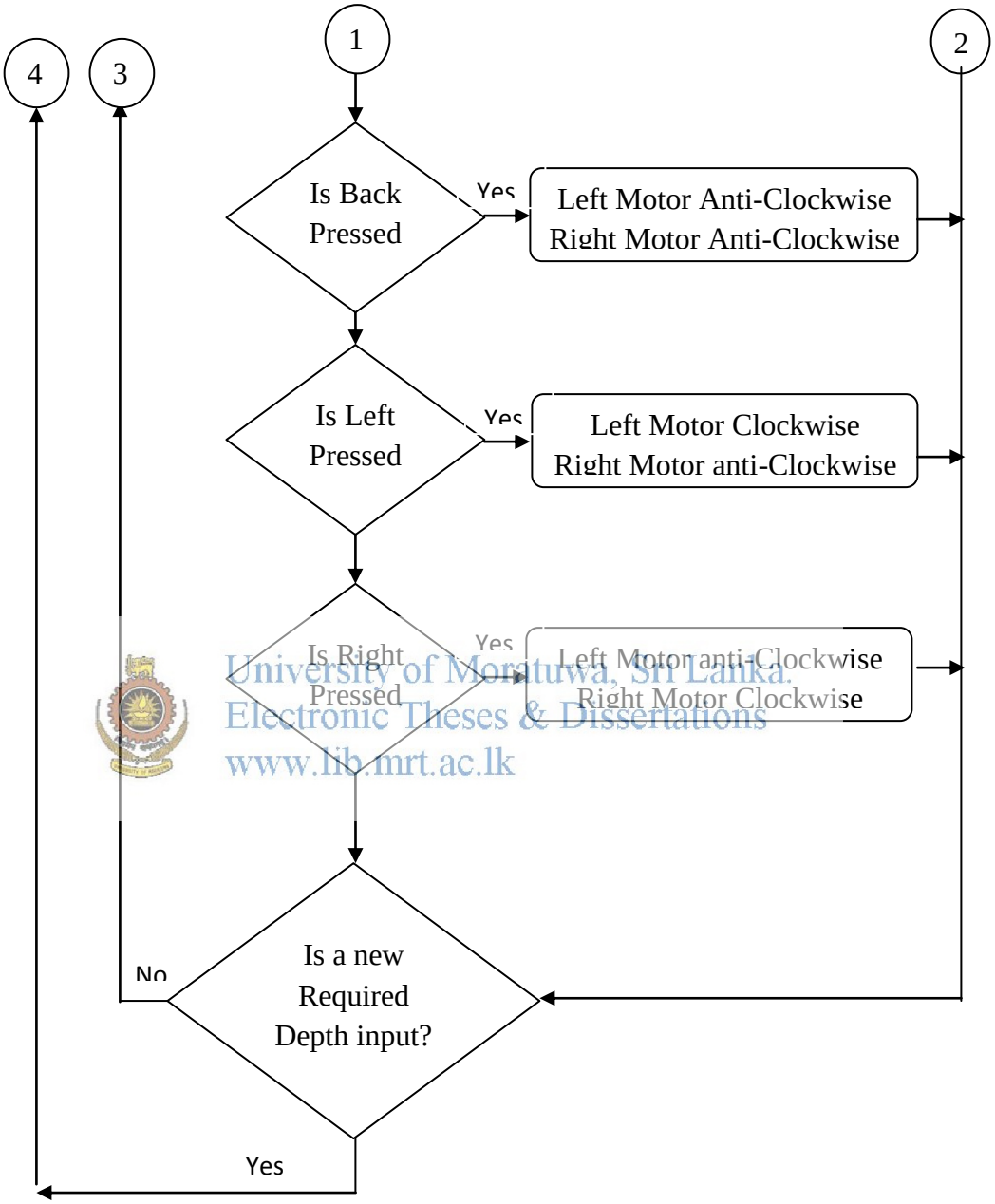
```
class MS5803 {
private:
   int D1, D2, Temp, C[8];
   float T_MS5803, P_MS5803;
   /* Data buffers */
   char ms5803_rx_data[MS5803_RX_DEPTH];
   char ms5803_tx_data[MS5803_TX_DEPTH];

public:
   MS5803 (PinName sda, PinName scl,
         char ms5803_addr = ms5803_base_addr  )
         : i2c( sda, scl ), device_address( ms5803_addr << 1 ) {
   }
   void MS5803Reset(void);
```

```
    void MS5803ReadProm(void);
    void MS5803ConvertD1(void);
    void MS5803ConvertD2(void);
    int32_t MS5803ReadADC(void);
    float MS5803_Pressure (void);
    float MS5803_Temperature (void);
    void Barometer_MS5803(void);


private:
    I2C     i2c;
    char    device_address;

};
#endif
```

# FLOW CHART



# FLOW CHART



# FLOW CHART