REFERENCES

[1]. A.M.H.S.Abeykoon, and K.Ohnishi, "Tactile sensation improvement of a bilateral forceps robot with a switching virtual model," in *10th IEEE Int. Workshop on Advanced Motion Control, 2008*,pp.526-531, doi: 10.1109/AMC.2008.4516122

[2]. N. Murata, and S. Katsura, "Bilateral force feedback control using multi-degree-of-freedom mobile robot," in *2011 Proc. of SICE Annu. Conf.* , 2011,pp.1041-1046.

[3]. A.Hace, and M.Franc, "Pseudo-Sensorless High-Performance Bilateral Teleoperation by Sliding-Mode Control and FPGA," in *IEEE/ASME Trans. on Mechatronics*, vol.19, no.1, pp.384,393, Feb. 2014

[4]. L.Chan; F.Naghdy, and D.Stirling, "Application of Adaptive Controllers in Teleoperation Systems: A Survey," in *IEEE Trans. on Human-Machine Systems,* , vol.44, no.3, pp.337,352, June 2014. doi: 10.1109/THMS.2014.2303983

[5]. A.M.H.S.Abeykoon, and K.Ohnishi, "Realization of Virtual Slave Model of a Forceps Robot Using Bilateral Control," in *32nd Annu. Conf. on IEEE Industrial Electronics*,2006, pp.4468-4473. doi: 10.1109/IECON.2006.348100

[6]. D.T. Pham and E.Tacgin, “An Expert system for selection of robot grippers,” *Expert Syst. Appl.*, vol.5, pp. 289-300, 1992.

[7]. M. K. Brown, “A Controlled Impedance Robot Gripper” in *AT&T Technical J.*,1985, vol. 64, no. 4, pp. 937-969.

[8]. J.M. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K.J. Kuchenbecker, "Human-Inspired Robotic Grasp Control With Tactile Sensing," in *IEEE Trans. on Robotics*, vol.27, no.6, pp.1067-1079, doi: 10.1109/TRO.2011.2162271.

[9]. J. Becedas, I. Payo, and V. Feliu, "Two-Flexible-Fingers Gripper Force Feedback Control System for Its Application as End Effector on a 6-DOF Manipulator," in *IEEE Trans. on Robotics*, 2011, vol.27, no.3, pp.599-615, doi: 10.1109/TRO.2011.2132850.

[10]. M. Florescu, D. Strimbeanu, and D.M. Matei, "Force control of grasping operation for a tentacle robot," in *18th Int. Conf. on System Theory, Control and Computing*, 2014, pp.832-837, doi: 10.1109/ICSTCC.2014.6982522.

[11]. M. Goldfarb, and N. Celanovic, “ A flexure-based gripper for small-scale manipulation” on *Robotica*, vol.null, pp. 181-187, Sept. 2000.

[12]. J.Y. Wang, and C.C. Lan, “ A Constant Force Compliant Gripper for Handling Objects of Various Sizes” in *ASME Trans. of J. of Mechanical Design*, 2014, vol. 136.

[13]. M. Kamali, S.A.A. Moosavian, and F. Cheraghpour, "Improving grasp capabilities of KNTU hand using position & force sensors," in *22nd Iranian Conf. on Electrical Engineering*, 2014, pp.1278-1283, doi: 10.1109/IranianCEE.2014.6999731

[14]. R.D. Lorenz, K.M. Meyer, and D.M.V De Riet, "A novel, compliant, four degree-of-freedom, robotic fingertip sensor," in *IEEE Trans. on Industry Applications*, 1990, vol. 26, no. 4, pp. 613-619.

[15]. T.L. Gibo, D.R. Deo, Z.F. Quek; Okamura, A.M., "Effect of load force feedback on grip force control during teleoperation: A preliminary study," in *IEEE Haptics Symp.(HAPTICS)*, 2014 , pp.379-383, doi: 10.1109/HAPTICS.2014.6775485.

[16]. G.S. Gupta, S.C. Mukhopadhyay, C.H. Messom, and S.N. Demidenko, "Master–Slave Control of a Teleoperated Anthropomorphic Robotic Arm With Gripping Force Sensing," in *IEEE Trans. on Instrumentation and Measurement* , 2006, vol.55, no.6, pp.2136-2145, doi: 10.1109/TIM.2006.884393.

[17]. R.M. Pierce, E.A. Fedalei, and K.J. Kuchenbecker, "A wearable device for controlling a robot gripper with fingertip contact, pressure, vibrotactile, and grip force feedback," *IEEE Haptics Symp.*, 2014, pp.19-25, doi: 10.1109/HAPTICS.2014.6775428.

[18]. Y. Yokokura, S. Katsura, and K. Ohishi, "Motion copying system based on real-world haptics," in *10th IEEE Int. Workshop on Advanced Motion Control*, pp. 613-618, Mar. 2008. doi: 10.1109/AMC.2008.4516137

[19]. S.Katsura, W. Yamanouchi, and Y. Yokokura, "Real-World Haptics: Reproduction of Human Motion," in *IEEE Ind. Electronics. Mag.*, vol.6, no.1, pp.25-31, Mar. 2012. doi: 10.1109/MIE.2012.2182854

[20]. A.M.H.S.Abeykoon, and K.Ohnishi, "Virtual tool for bilaterally controlled forceps robot-for minimally invasive surgery," in *The Int. J. of Medical Robotics and Comput. Assisted Surgery*, vol.3, no.3, pp. 271-280, Aug 2007.

[21]. A.M.H.S. Abeykoon, and K. Ohnishi, "Bilateral Control interacting with a Virtual Model and Environment," in *IEEE Int. Conf. on Ind. Technology*, pp.1320-1325, Dec. 2006. doi: 10.1109/ICIT.2006.372520

[22]. T. Shimono, S. Katsura, and K. Ohnishi, "Abstraction and Reproduction of Force Sensation From Real Environment by Bilateral Control,"in *IEEE Trans. on Ind. Electronics*, vol.54, no.2, pp.907-918, April 2007. doi: 10.1109/TIE.2007.892744

[23]. N. Tsunashima, and S. Katsura, "Spatiotemporal Coupler: Storage and Reproduction of Human Finger Motions,"in *IEEE Trans. on Ind. Electronics*, vol.59, no.2, pp.1074-1085, Feb. 2012. doi: 10.1109/TIE.2011.2161247

[24]. S. Yajima, and S. Katsura, "Multi-DOF Motion Reproduction Using Motion-Copying System with Velocity Constraint,"in *IEEE Trans. on Ind. Electronics*, vol.61, no.7, pp.3765-3775, July 2014. doi: 10.1109/TIE.2013.2286086

[25]. T. Shimono, S. Katsura, and K. Ohnishi, "Reproduction of Real World Force Sensation by Bilateral Motion Control Based on Contact Impedance Model Taking Environmental Hysteresis into Account,"in *IEEE Int. Conf. on Mechatronics*, pp.613-618, July 2006. doi: 10.1109/ICMECH.2006.252596

[26]. T.Murakami, K.Ohnishi, " Observer Based Motion Control- Application to Robust Control and Parameter Identification," in *IEEE trans. on Ind. Electronics.*,1993

[27]. Y.S.E. Ali, S.B.M. Noor, S.M. Bashi, and M.K. Hassan,"Microcontroller performance for DC motor speed control system," in *Proc.of Nat. Power Engineering Conf.*, pp. 104- 109, 2003.

[28]. G. Huang, and S. Lee, “PC-based PID speed control in DC motor,” in. *Proc. of the Int.Conf. on Audio, Language and Image Processing*, pp 400-407, July 2008

[29]. M. Mizuochi, T. Tsuji, and K. Ohnishi, "Improvement of disturbance suppression based on disturbance observer," in *9th IEEE Int. Workshop on Advanced Motion Control*, pp.229-234, 2006.

[30]. B.J.A. Helouvry, P. Dupont, and C.De Wit, "A survey of models, analysis tools and compensation method for the control of machines with friction,"in *Automatica*, Vol. 30, No.7 , pp. 1083-1138,1994

[31]. K. Ohnishi, M. Shibata, and T. Murakami : “Motion control for advanced mechatronics,” in *IEEE/ASME Trans. on Mechatronics*, , vol.1, no.1pp.56-67, Mar 1996.

[32]. S. Katsura, K. Irie, and K. Ohishi, "Wideband Force Control by Position-Acceleration Integrated Disturbance Observer," in *IEEE Trans. on Ind. Electronics,* vol.55, no.4, pp.1699-1706, April 2008.

[33]. T.Murakami, K.Ohnishi, (1993) “Disturbance Observer Based Motion Control- Application to Robust Control and Parameter Identification,” in *IEEE trans. on Ind. Electronics*.

[34]. S. Katsura, Y. Matsumoto, and K. Ohnishi : "Modeling of force sensing and validation of disturbance observer for force control," in *29th Ann. Conf. of the IEEE Ind. Electronics Society,* vol.1, pp. 291-296, 2003.

[35]. K.Ohishi, and K.Miyachi,"Torque-speed regulation of DC motor based on load torque estimation," *IEEJ IPEC-Tokyo*, vol.2, pp.1209-1216,1983.

[36]. A. Sabanovic, and K. Ohnishi (2011) “*Motion control systems,”* in IEEE press John Willey and sons (Asia) pte Ltd, (First Edition).,2011

[37]. A.M H.S. Abeykoon, and K. Ohnishi: “Improvement of Tactile Sensation of a Bilateral Forceps Robot by a Switching Virtual Model,” in *Trans.on Advanced Robotics*”, Vol. 8, pp. 789-806 , 2008.

[38]. A.M.H.S. Abeykoon, and K. Ohnishi, “Virtual tool for bilaterally controlled forceps robot-for minimally invasive surgery,” in *Trans. on int. J. of Medical Robotics and Computer Assisted Surgery*”, ISSN 1478-5951, vol 3; No 3, pp. 271-280, 2007.

[39]. Y.S.E. Ali, S.B.M. Noor, S.M. Bashi, and M.K. Hassan, "Microcontroller performance for DC motor speed control system," in *Proc.of Nat. Power Engineering Conf*, 104- 109, Dec. 2003.

[40]. https://developer.mbed.org/users/hexley/notebook/qei_hw-interface-implementation-notes/

[41]. T. Shimono, S. Katsura, and K. Ohnishi, "Abstraction and Reproduction of Force Sensation From Real Environment by Bilateral Control,"in *IEEE Trans. on Ind. Electronics*, vol.54, no.2, pp.907-918, April 2007. doi: 10.1109/TIE.2007.892744

[42]. Y. Yokokohji, and T. Yoshikawa, “Bilateral control of Master-slave Manipulator for ideal Kinesthetic Coupling- Formulation and Experiment,” in *IEEE Trans. on Robotics and Automation* vol. 10, no. 5, pp 605-620,1994

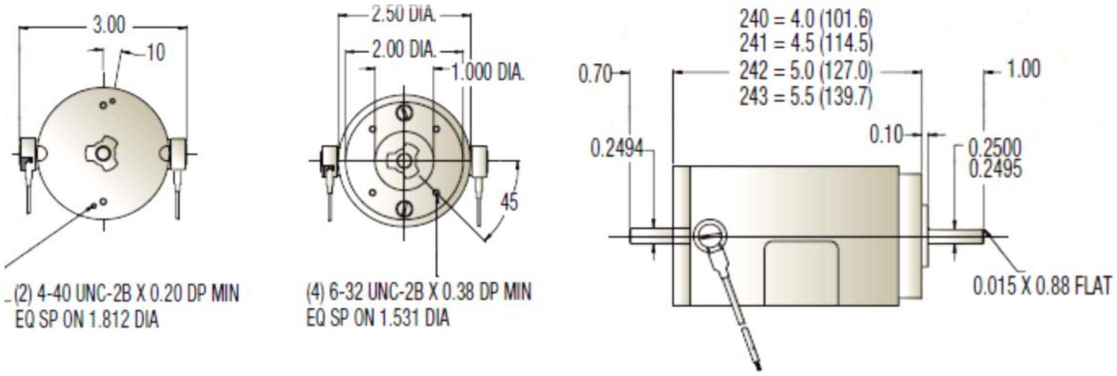
[43]. M. K. C. D. Chinthaka, “Position Based Friction Estimation for Precise Motion Control”, University of Moratuwa, M.Sc thesis, July 2014

[44]. B. M. Pillai, “Parameters Estimation for Motion Control and Friction Compensation”, University of Moratuwa, M.Sc thesis, August 2013

APPENDIX

APPENDIX I : Specification of the motor

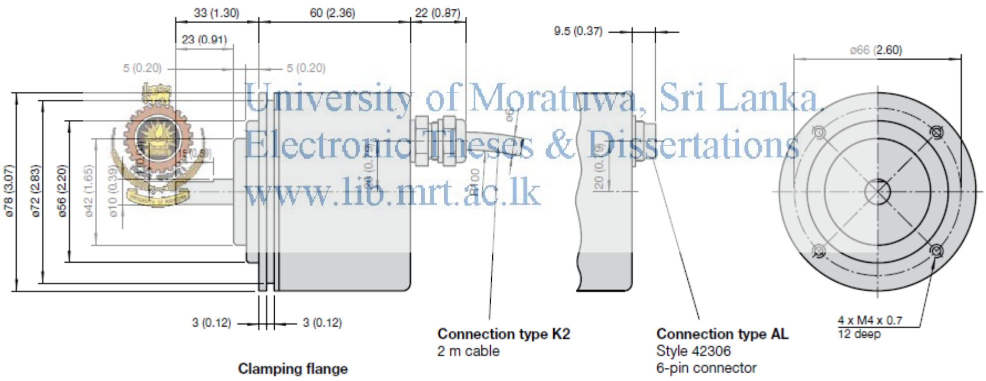
Manufacturer	:	Electrocraft
Version	:	E240
Maximum Terminal Voltage	:	60V DC
Supply Voltage	:	32V DC
Continuous stall Torque	:	20.5 Ncm
Peak Torque	:	169.5 Ncm
Maximum Speed	:	5000 rpm
Rotor Inertia	:	0.268 $Kgcm^2$
Maximum Friction Torque	:	2.1 Ncm
Weight	:	1Kg
Torque Constant	:	13.5 Ncm/Amp
Terminal Resistance	:	5.4 Ω
Armature Inductance	:	8.2 mH



Dimensions of the motor

APPENDIX II : Specification of the master encoder

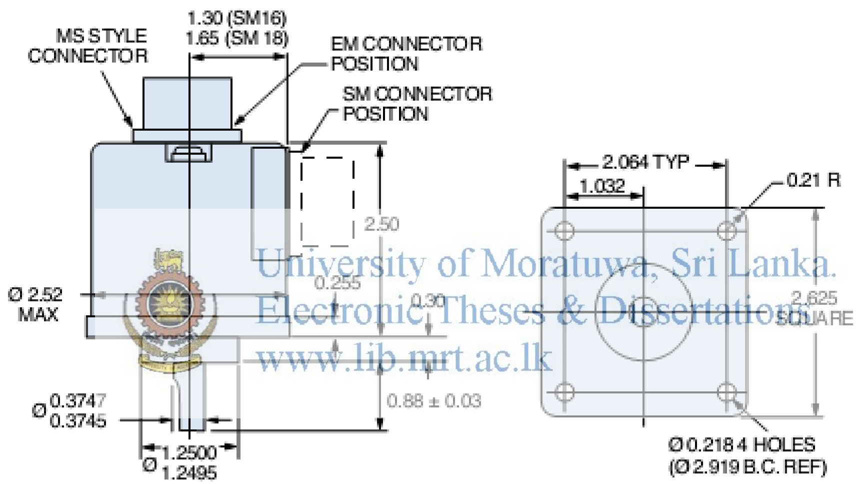
Pulse Count	:	5,000 ppr (pulse per revolution)
Pulse count with QEI	:	20,000 ppr
Output Frequency	:	$\leq$ 100 kHz
Supply Voltage	:	10-30 VDC
Applied voltage	:	12 VDC
Output	:	Push-pull
Current Consumption	:	$\leq$ 80 mA
Load Current	:	$\leq$ 40 mA, short circuit,
Voltage Drop	:	< 4 V
Response Time	:	250 ns
Rotational Speed	:	< 6000 rpm
Moment of Inertia	:	$\leq 1.4 \times 10^{-3}$ oz-in-sec^2



Dimensions of the master encoder

APPENDIX III : Specification of the slave encoder

Model	:	H25D (BEI)
Pulse Count	:	2,500 ppr (pulse per revolution)
Pulse count with QEI	:	10,000 ppr
Supply Voltage	:	5 to 28 VDC available
Applied voltage	:	12 VDC
Current Requirements	:	100 mA typical +output load, 250 mA (max)
Voltage/Output	:	28V/5: Line Driver, 5–28 VDC in, V out= 5 VDC
Frequency Response	:	100 kHz, up to 1MHz with interpolation option
Moment of Inertia	:	5.2 X 10^{-4}oz-in-sec^2



Dimensions of the slave encoder

APPENDIX IV : Specification of the coupler

Shaft usage	:	8mm x 8mm (0.315" x 0.315")
Length	:	25mm
Diameter	:	18mm
Material	:	Aluminum
Rated Torque	:	1N.m
Max. Torque	:	2N.m
Eccentricity Error	:	±0.2mm
Shaft Angel	:	≤2°
Max. Rotational(RPM)	:	19000
Rated	:	In-Phase Operate

APPENDIX V: MATLAB m code

```
function [x_response_s]=BC_D_MandS(t)
%PD controller Block parameters of master
Kp_m=350;
Kd_m=25;
PD_out_m=0.0;
filter_pd_m=0.0;
filter_pd_m_pre=0.0;
g_pd_m=50.0;

%PD controller Block parameters of slave
Kp_s=350;
Kd_s=25;
filter_pd_s=0.0;
filter_pd_s_pre=0.0;
PD_out_s=0.0;
g_pd_s=50.0;

%Motor parameters of master
Mn_m=0.40;
Kfn_m=24.0;

%Motor parameters of slave
Mn_s=0.4;
Kfn_s=24.0;

%DOB & RTOB paremeter of master
g_dis_m=150.0;
to_filter_m=0.0;
filter_out_m=0.0;
filter_out_pre_m=0.0;
DOB_out_m=0.0;
friction_m=0.0;
RTOB_out_m=0.0;

%DOB & RTOB paremeter of slave
g_dis_s=150.0;
to_filter_s=0.0;
filter_out_s=0.0;
filter_out_pre_s=0.0;
DOB_out_s=0.0;
friction_s=0.0;
RTOB_out_s=0.0;

%load model
B=5;
M=0.1;
k=250;

%controller Block parameters of slave force controller
Kp_FC=3.3;
Kd_FC=1.1;
g_FC=150.0;
g_FC_r=150.0;
filter_FC=0.0;
```

```
filter_FC_pre=0.0;
FC_error=0.0;
filter_FC_r=0.0;
dF_error=0.0;
dF_error_r=0.0;
filter_FC_r_pre=0.0;
F_defined=9.0;
FC_out=0.0;

%spring controller master
k_sp=100.0;

%Constant parameter define
c=10.0;
torque_error=0.0;
x_response_m=0.0;
x_response_pre_m=0.0;
x_error_m=0.0;
dx_response_m=0.0;
dx_response_m_pre=0.0;
F_m=0.0;
I_error_m=0.0;
load_force_m=0.0;
force_error_m=0.0;
motor_acc_m=0.0;
motor_velocity_m=0.0;
motor_velocity_pre_m=0.0;
x_response_s=0.0;
x_response_pre_s=0.0;
x_error_s=0.0;
dx_response_s=0.0;
dx_response_s_pre=0.0;
F_s=0.0;
I_error_s=0.0;
load_force_s=0.0;
force_error_s=0.0;
motor_acc_s=0.0;
motor_velocity_s=0.0;
motor_velocity_pre_s=0.0;
j=0.0;

dt=0.0001;
A=zeros(500001,5);
i=0;

for t=0.0:0.0001:50
    i=i+1;
%External force profile for master
if (t>=0.0)&(t<5.0)
    load_force_m=0.05+0.02*t;
elseif (t>=5.0)&(t<10)
    load_force_m=1.97*t-9.7;
elseif (t>=10.0)&(t<15)
    load_force_m=c;
elseif (t>=15.0)&(t<25)
    load_force_m=-t+25;
elseif (t>=25.0)&(t<30)
```

```
    load_force_m=0.0;
elseif (t>=30.0)&(t<31)
    load_force_m=-0.05*t+1.5;
elseif (t>=31)&(t<33)
    load_force_m=0.05*t-1.6;
else
    load_force_m=0.05;
end

x_error_m=x_response_s-x_response_m;
x_error_s=x_response_m-x_response_s;

filter_pd_s=filter_pd_s_pre+dx_response_s*dt;
filter_pd_m=filter_pd_m_pre+dx_response_m*dt;
filter_pd_s_pre=filter_pd_s;
filter_pd_m_pre=filter_pd_m;

dx_response_s=g_pd_m*(x_response_s-filter_pd_s);        %filter
gains are equel for both master and slave PD controller
dx_response_m=g_pd_s*(x_response_m-filter_pd_m);

PD_out_m=Kp_m*x_error_m+Kd_m*(dx_response_s-dx_response_m);
PD_out_s=Kp_s*x_error_s+Kd_s*(dx_response_m-dx_response_s);

F_s= PD_out_s*Mn_s-torque_error;
F_m=PD_out_m*Mn_m-torque_error;

if (abs(RTOB_out_s) <F_defined)|(load_force_m<0)

if (x_response_s>-0.05&load_force_m>0)
    load_force_s=-0.05+5.5*motor_velocity_s;
elseif (-0.05-x_response_s)>=0
load_force_s=M*motor_acc_s+B*motor_velocity_s+k*(x_response_s+0.05);
elseif
(load_force_m<0&(abs(x_response_m)<=0.95*abs(x_response_s))&x_respon
se_s<=-0.05)
load_force_s=M*motor_acc_s+B*motor_velocity_s+k*(x_response_s+0.05);
elseif (x_response_s>-0.05&(load_force_m<0))
    load_force_s=0.05+10.0*motor_velocity_s;
end

torque_error = RTOB_out_m + RTOB_out_s;
I_error_s=F_s*(1/Kfn_s)+DOB_out_s*(1/Kfn_s);
end

if (abs(RTOB_out_s) >= F_defined)&(load_force_m>=0)
 FC_error= F_defined+RTOB_out_s;

 filter_FC=filter_FC_pre+dF_error*dt;
 filter_FC_r=filter_FC_r_pre+dF_error_r*dt;
 filter_FC_pre=filter_FC;
 filter_FC_r_pre=filter_FC_r;

 dF_error=g_FC*(F_defined-filter_FC);
 dF_error_r=g_FC_r*(RTOB_out_s-filter_FC_r);
```

```
 FC_out=Kp_FC*FC_error+Kd_FC*(dF_error+dF_error_r);

 torque_error = RTOB_out_m;
if (j-x_response_s)>=0
load_force_s=M*motor_acc_s+B*motor_velocity_s+k*(x_response_s+0.05);
end
 I_error_s=FC_out*(1/Kfn_s)*(-1)+DOB_out_s*(1/Kfn_s);
 F_m=k_sp*(x_response_s-x_response_m)-5.0*motor_velocity_m-
torque_error;
end

I_error_m=F_m*(1/Kfn_m)+DOB_out_m*(1/Kfn_m);

%Motors
force_error_m=I_error_m*Kfn_m-load_force_m;
motor_acc_m=force_error_m*(1/Mn_m);
force_error_s=I_error_s*Kfn_s-load_force_s;
motor_acc_s=force_error_s*(1/Mn_s);
motor_velocity_m=motor_acc_m*dt+motor_velocity_pre_m;
motor_velocity_s=motor_acc_s*dt+motor_velocity_pre_s;
x_response_m=motor_velocity_m*dt+x_response_pre_m;
x_response_s=motor_velocity_s*dt+x_response_pre_s;

%Mster DOB & RTOB
to_filter_m=I_error_m*Kfn_m+motor_velocity_m*g_dis_m*Mn_m;
filter_out_m=filter_out_pre_m+g_dis_m*(to_filter_m-filter_out_m)*dt;
filter_out_pre_m=filter_out_m;
DOB_out_m=filter_out_m-motor_velocity_m*g_dis_m*Mn_m;
RTOB_out_m=DOB_out_m-friction_m;

%Slave DOB & RTOB
to_filter_s=I_error_s*Kfn_s+motor_velocity_s*g_dis_s*Mn_s;
filter_out_s=filter_out_pre_s+g_dis_s*(to_filter_s-filter_out_s)*dt;
filter_out_pre_s=filter_out_s;
DOB_out_s=filter_out_s-motor_velocity_s*g_dis_s*Mn_s;
RTOB_out_s=DOB_out_s-friction_s;
motor_velocity_pre_m=motor_velocity_m;
motor_velocity_pre_s=motor_velocity_s;
x_response_pre_m=x_response_m;
x_response_pre_s=x_response_s;

A(i,:)=[t, x_response_m, x_response_s, RTOB_out_m, RTOB_out_s];
hold on;
figure(1);
f1=plot(A(:,1),A(:,2),A(:,1),A(:,3));
xlabel('time(s)');
ylabel('position (m)');
hold on;
figure(2);
f2=plot(A(:,1),A(:,4),A(:,1),A(:,5));
hold on;
xlabel('time(s)');
ylabel('force responce (N)');
end
```

APPENDIX VI: C code on mbed microcontroller

```
#include "mbed.h"
#include "rtos.h"
#include "SDFileSystem.h"
#include "qeihw.h"
#include "math.h"

#define PI 3.141592653
#define Kp 900.0
#define Kv 10.0
#define Ki 10.0
#define Gd 100.0
#define F_limit 0.10

QEIHW qei_s(QEI_DIRINV_NONE, QEI_SIGNALMODE_QUAD, QEI_CAPMODE_4X,
QEI_INVINX_NONE );

// Configuring two encoder modules
void ethernet_init();
Ethernet eth;

//Variable for get angle from ethernet
char buf[0x600];
float recv_m_angle = 0.0;
float recv_s_angle = 0.0;
float inc_now = 0.0, inc_pre = 0.0;

//Safety for mbed unused pins
DigitalIn safety_19(p19);
DigitalIn safety_20(p20);
DigitalIn safety_25(p25);
DigitalIn safety_26(p26);

//Motor pwm mbed pins
PwmOut pwm_m_clk(p21);    // clockwise rotation pwm pin MASTER
PwmOut pwm_m_anticlk(p22);// anti-clockwise rotation pwm pin MASTER
PwmOut pwm_s_clk(p23);    // clockwise rotation pwm pin for SLAVE
PwmOut pwm_s_anticlk(p24);//anti-clockwise rotation pwm pin SLAVE

// Motor H bridge pins
DigitalOut  Reset_AB_M(p27);
DigitalOut  Reset_CD_M(p28);
DigitalOut  Reset_AB_S(p29);
DigitalOut  Reset_CD_S(p30);

DigitalIn M_Dir(p9);
DigitalIn S_Dir(p10);

//Current sensor inputs
AnalogIn current_sensor_m_p(p15);//current sensor input MASTER +ve
AnalogIn current_sensor_m_n(p16);// current sensor input MASTER -ve
AnalogIn current_sensor_s_p(p17);// current sensor input SLAVE +ve
AnalogIn current_sensor_s_n(p18);// current sensor input SLAVE -ve

//LED output for testing code
DigitalOut led1(LED1);
```

```
DigitalOut led3(LED3);

//Current Sensor Directions
int Master_Direction=0;
int Slave_Direction = 0;

// Encoder Constants
float const encoder_pulses_s = 2400.0;

// Motor Constant and Inertia
float const J_const_m = 0.000910;
float const J_const_s = 0.000910;
float const Kt_const_m = 0.135;
float const Kt_const_s = 0.134;
float const Kt_constinv_m = 7.407407407;
float const Kt_constinv_s = 7.462586567;

//the main function variables
Timer timer;
Timer timer1;
FILE *fp;
Ticker ticker;
int counter=0;                     //data writing loop counter
int counter_old=0;
int counter_time;
int dt_us= 150, ramp_time=0.0;     // define main loop time in us
float dt;                          //loop time in seconds for
calculations

//velocity controller variables
float x_res_m = 0.0;
float x_res_s = 0.0;
float dx_res_m = 0.0;
float dx_res_s = 0.0;
float dx_e_sum_m = 0.0;
float dx_e_sum_s = 0.0;
float const G_filcon_v_m = 2.0;//Low pass filter gain velocity
float const G_filcon_v_s = 2.0;

//current controller variables
float const G_filcon_I1_m = 100.0;
float const G_filcon_I1_s = 100.0;
float const G_filcon_I_m = 100.0;
float const G_filcon_I_s = 100.0;
float  I1_act_m=0.0;
float  I1_act_s=0.0;
float I_act_m = 0.0;
float I_act_s = 0.0;
float I_ref_m = 0.0;
float I_ref_s = 0.0;
float I_res_m = 0.0;
float I_res_s = 0.0;
float I_err_m = 0.0;
float I_err_s = 0.0;
float I_tmp_m = 0.0;
float I_tmp_s = 0.0;
float tem_I_m = 0.0;
float tem_I_s = 0.0;
```

```
float d_I_m = 0.0;
float d_I_s = 0.0;
float pwm_I_M= 0.0;
float pwm_I_S= 0.0;
float const Ikp_m = 25.0, Iki_m =1.50, Ikd_m = 0.015;
float const Ikp_s = 25.0, Iki_s = 1.50,Ikd_s = 0.015;

//DOB and RTOB variables
float tmp_m = 0.0;
float tmp_s = 0.0;
float ob_sum_m = 0.0;
float ob_sum_s = 0.0;
float ob_sum_m1 = 0.0;
float ob_sum_s1 = 0.0;
float i_com_m = 0.0;
float i_com_s = 0.0;
float fric_m = 0.0;
float fric_s = 0.0;
float i_rto_m = 0.0;
float i_rto_s = 0.0;
float torque_dob_m=0.0;
float torque_dob_s=0.0;

//controller variables
float ddx_ref_m=0.0;
float ddx_ref_s=0.0;
float I_ref_m1=0.0;
float I_ref_s1=0.0;
float tem_x_m=0.0;
float tem_x_s=0.0;
float x_err_m=0.0;
float x_err_s=0.0;
float torque_error=0.0;
float DOB_out_m=0.0;
float DOB_out_s=0.0;

//PWM generator variables
float duty_m = 0.0;                  // PWM duty for master
float duty_s = 0.0;

//Force controller variables
float const Kp_FC=650.0,Kd_FC=1.10, Ki_FC=60.0;
float G_filcon_FC=10.0;
float FC_error=0.0;
float dF_e_cmd_sum=0.0;
float dF_e_cmd=0.0;
float dF_e_res_sum=0.0;
float dF_e_res=0.0;
float FC_out=0.0;
float F_m=0.0;
float F_m1=0.0;
float F_s=0.0;
float x_eq=0.0;

//Spring controller variables
float const Kp_SC=550.0,Kd_SC=10.0,Ki_SC=5.0,k_sp= 15.0;
float G_filcon_SC=20.0;
float SC_error=0.0;
```

```
float dS_e_sum=0.0;
float dS_e=0.0;
float SC_out=0.0;
float SC_tmp=0.0;
float VC_out=0.0;
float  dx_e_VC_m=0.0;
float dx_res_VC=0.0;
float dx_VC_m=0.0;

int a=0, b=0, c=0;
float FC_S=0.0;
float FC_M=0.0;

 void pwm_init(void) {
    pwm_m_clk.period_us(10);
    pwm_m_anticlk.period_us(10);
    pwm_s_anticlk.period_us(10);
    pwm_s_clk.period_us(10);

    pwm_m_clk.write(0.0f);// Set the ouput duty-cycle, specified as
a percentage (float)
    pwm_m_anticlk.write(0.0f);
    pwm_s_anticlk.write(0.0f);
    pwm_s_clk.write(0.0f);

    Reset_AB_M = 1;        //ENABLE RUNNING MODE  (H BRIDGE ENABLE)
    Reset_CD_M = 1;
    Reset_AB_S = 1;
    Reset_CD_S = 1;
}
void velocity_m(){
        int size2 = eth.receive();
        if(size2 > 0) {
        eth.read(buf, size2);
        memcpy(&recv_m_angle, buf, sizeof(float));
        x_res_m = recv_m_angle;
        }
     dx_e_sum_m += dx_res_m*dt;
     dx_res_m =G_filcon_v_m*( x_res_m-dx_e_sum_m);
}
void velocity_s() {
    int32_t position = 0;
    qei_s.SetDigiFilter(480UL);
    qei_s.SetMaxPosition(0xFFFFFFFF);
    position = qei_s.GetPosition();
    x_res_s = -1.0*position * 2.0 * PI / encoder_pulses_s;

    dx_e_sum_s += dx_res_s*dt;
    dx_res_s = G_filcon_v_s*(x_res_s-dx_e_sum_s);
}
void current_pid(){
   Master_Direction = M_Dir.read();
   if(Master_Direction == 0){                  //master clockwise
        I_res_m = current_sensor_m_p.read();
        I1_act_m = -1.0*((I_res_m*3.3/0.74787687701613) );
      }
   else if(Master_Direction == 1) {      //master anticlockwise
        I_res_m =  current_sensor_m_n.read();
```

```
    I1_act_m = 1.0*((I_res_m*3.3)/0.713239227822580);  }

    I_act_m =I_act_m_pre + G_filcon_I1_m*(I1_act_m-I_act_m)*dt;
    I_act_m_pre=I_act_m;
    I_err_m = I_ref_m - I_act_m;
    I_tmp_m += (Iki_m * dt * I_err_m);
    tem_I_m += d_I_m*dt;
    d_I_m = G_filcon_I_m*(I_err_m - tem_I_m);
    pwm_I_M=((I_err_m * Ikp_m) + I_tmp_m + (d_I_m * Ikd_m));

    Slave_Direction = S_Dir.read();
    if(Slave_Direction == 0){     //slave clockwise
        I_res_s =  current_sensor_s_p.read();
        I1_act_s = -1.0*((I_res_s*3.3)/0.717075441532258 );
    }
   else if (Slave_Direction == 1){
        I_res_s =  current_sensor_s_n.read(); //slave anticlockwise
        I1_act_s = 1.0*((I_res_s*3.3)/0.724138445564516);}

    I_act_s =I_act_s_pre + G_filcon_I1_s*(I1_act_s-I_act_s)*dt;
    I_act_s_pre=I_act_s;
    I_err_s = I_ref_s - I_act_s;
    I_tmp_s +=  (Iki_s* dt * I_err_s);
    tem_I_s += d_I_s*dt;
    d_I_s = G_filcon_I_s*(I_err_s - tem_I_s);
    pwm_I_S=((I_err_s * Ikp_s) + I_tmp_s + (d_I_s * Ikd_s));
    }

void Disob() {
    tmp_m = Gd*J_const_m*dx_res_m;
    ob_sum_m1  = Kt_const_m*I_act_m+tmp_m;
    ob_sum_m  += Gd*(ob_sum_m1-ob_sum_m)*dt;
    DOB_out_m = (ob_sum_m - tmp_m);
    i_com_m  = (ob_sum_m - tmp_m)*Kt_constinv_m;    //read current
    fric_m = 0.0;
    torque_dob_m= DOB_out_m-fric_m;
    i_rto_m = torque_dob_m*Kt_constinv_m;

    tmp_s      = Gd*J_const_s*dx_res_s;
    ob_sum_s1  = Kt_const_s*I_act_s+tmp_s;
    ob_sum_s  += Gd*(ob_sum_s1-ob_sum_s)*dt;
    DOB_out_s = (ob_sum_s - tmp_s);
    i_com_s  = (ob_sum_s - tmp_s)*Kt_constinv_s;    //read current
    fric_s = 0.0;//0.011;
    torque_dob_s = DOB_out_s-fric_s;
    i_rto_s = torque_dob_s*Kt_constinv_s;
    }

int Controller(void) {
if (((torque_dob_s*-1.0) <F_limit)|((x_eq-
x_res_m)<0.0&(torque_dob_m<0.0))){
    a=0;
    x_err_m=x_res_s-x_res_m;
    tem_x_m += (Ki* dt* x_err_m);
    ddx_ref_m = Kp*x_err_m +  Kv*(dx_res_s-dx_res_m)+tem_x_m;
    x_err_s=x_res_m-x_res_s;
    tem_x_s += (Ki* dt* x_err_s);
    ddx_ref_s = Kp* x_err_s +  Kv*(dx_res_m-dx_res_s)+tem_x_s;
```

```
    F_m=ddx_ref_m*J_const_m-torque_error;
    F_s=ddx_ref_s*J_const_s-torque_error;
    torque_error= torque_dob_m+torque_dob_s;
    I_ref_s1 = Kt_constinv_s*F_s;
    I_ref_s  =I_ref_s1 + DOB_out_s*Kt_constinv_s;
    I_ref_m =  Kt_constinv_m*F_m+DOB_out_m*Kt_constinv_m;
    timer1.stop();
    timer1.reset();
    FC_S=0.0;
    FC_M=0.0;
    }
if (((torque_dob_s*-1.0) >=(0.90*F_limit))&((torque_dob_s*-
1.0)<=F_limit)){//
        FC_S=I_ref_s;
        FC_M=F_m;}

if((torque_dob_s*-1.0) >= F_limit){
    a=1;
    x_eq=x_res_s;
    I_ref_s1 = Kt_constinv_s*FC_S*1.0;
    I_ref_s  =FC_S; //I_ref_s1+ DOB_out_s*Kt_constinv_s;
    timer1.start();

if(((x_eq-x_res_m)>=0.0)|(torque_dob_m>=0.0)){
    if(timer1.read()<0.500){
        if(c%100==0){
            b=!b;
            }
            c++;

        x_err_m=x_eq+0.05*b-x_res_m;
        dx_e_VC_m += dx_res_VC*dt;
        dx_res_VC =G_filcon_v_m*(x_err_m-dx_e_VC_m);
        tem_x_m += (Ki* dt* x_err_m);
        ddx_ref_m = Kp*x_err_m+  Kv*(dx_res_VC)+tem_x_m;

        VC_out=ddx_ref_m*J_const_m;}
    else{VC_out=0.0;}

    SC_error=k_sp*(x_eq-x_res_m)+VC_out+F_limit-torque_dob_m;
    SC_tmp += (Ki_SC * dt * SC_error);
    dS_e_sum += dS_e*dt;
    dS_e = G_filcon_SC*(SC_error-dS_e_sum);
    SC_out=(Kp_SC*SC_error+Kd_SC*(dS_e)+SC_tmp);   }

    I_ref_m =
Kt_constinv_m*J_const_m*(SC_out+VC_out)+DOB_out_m*Kt_constinv_m;
    }
    return 0;
}

void PWM_Generator() {
      duty_m =  pwm_I_M;

    if (duty_m> 0.0) {
        if (duty_m > 0.9) {
            duty_m = 0.9;
        }
```

```
            pwm_m_clk = 0.0;
            pwm_m_anticlk = duty_m;
        }

        if (duty_m < 0.0) {
            if (duty_m < -0.9) {
                duty_m = -0.9;
            }
            pwm_m_anticlk = 0.0;
            pwm_m_clk = -1.0 * duty_m;
        }

           duty_s = pwm_I_S;

        if (duty_s > 0.0) {
            if (duty_s > 0.9) {
                duty_s = 0.9;
            }
            pwm_s_clk = 0.0;
            pwm_s_anticlk = duty_s;
        }
        if (duty_s < 0.0) {
            if (duty_s < -0.9) {
                duty_s = -0.9;
            }
            pwm_s_anticlk = 0.0;
            pwm_s_clk = -1.0 * duty_s;
        }
}

void cleanup_module(void){
    pwm_m_clk = 0.0;                // pwm cleanup module
    pwm_m_anticlk = 0.0;
    pwm_s_anticlk = 0.0;
    pwm_s_clk = 0.0;

    Reset_AB_M = 0;                 //Reset H bridge
    Reset_CD_M = 0;
    Reset_AB_S = 0;
    Reset_CD_S = 0;

    led1=0;
    led3=0;
}

//RTOS
void Control_body() {            // Control Part - main code
        velocity_m();
        velocity_s ();
        Disob();
        Controller();
        current_pid();
        PWM_Generator();
        counter++;
   }

void thread_2(void const *argument){
    led1=1;
```

```
    SDFileSystem sd(p5, p6, p7, p8, "sd");
    FILE *fp = fopen("/sd/BCG.csv", "w");

    if(fp == NULL) {
         for(int i=0;i<5;i++){
            led3=!led3;
            wait(1.0);
            }
    }

   while(counter<300000){
        if(counter>=(counter_old+100)){
        fprintf(fp, "%d %f %f %f %f
\n",timer.read_us(),torque_dob_m,torque_dob_s,x_res_m,x_res_s);
            counter_old=counter;
            led3=!led3;
            }
        }

    fclose(fp);
    timer.stop();
    cleanup_module();
    ticker.detach ();
    wait(1.0);
}
void ethernet_init(){
    eth.set_link(Ethernet::HalfDuplex100);
    wait_ms(1000);                              // Needed after startup.

    if(eth.link()) {
         for(int i=0;i<3;i++){
            led3=!led3;
            wait(1.0);
            }
    }
}

int main() {
    ethernet_init();
    pwm_init();
    timer.start();
    dt=dt_us/1000000.0;

    ticker.attach_us(&Control_body, dt_us);
    Thread
thread(*thread_2,NULL,osPriorityAboveNormal,DEFAULT_STACK_SIZE*10.0,
NULL);
}
```