REFERENCES

[1] Allotta, B.; Giacalone, G.; Rinaldi, L., "A hand-held drilling tool for orthopedic surgery," *Mechatronics, IEEE/ASME Transactions on*, vol.2, no.4, pp.218,229, Dec.1997

[2] Bouazza-Marouf, K., I. Browbank, and J. R. Hewit. "Robot-assisted invasive orthopedic surgery." *Mechatronics* 6.4 (1996): 381-397.

[3] Wen-Yo Lee; Ching-Long Shih; Shih-Tseng Lee, "Force control and breakthrough detection of a bone-drilling system," *Transactions on Mechatronics, IEEE/ASME*, vol.9, no.1, pp.20,29, March 2004.

[4] Jang-Ho Cho; Hoeryong Jung; Kyungno Lee; Doo Yong Lee; Hyung Soo Ahn, "Haptic Rendering of Drilling into Femur Bone with Graded Stiffness," *Frontiers in the Convergence of Bioscience and Information Technologies*, 2007. FBIT 2007 , vol., no.05, pp.525,530, 11-13 Oct. 2007.

[5] Ying Hu; Haiyang Jin; Liwei Zhang; Peng Zhang; Jianwei Zhang, "State Recognition of Pedicle Drilling With Force Sensing in a Robotic Spinal Surgical System," *Transactions on Mechatronics, IEEE/ASME*, vol.19, no.1, pp.357,365, Feb.2014

[6] Kotev, V.; Boiadjiev, G.; Kawasaki, H.; Mouri, T.; Delchev, K.; Boiadjiev, T., "Design of a hand-held robotized module for bone drilling and cutting in orthopedic surgery," System Integration (SII), 2012 IEEE/SICE International Symposium on , vol., no., pp.504,509, 16-18 Dec. 2012

[7] Lee, JuEun, B. Arda Gozen, and O. Burak Ozdoganlar. "Modeling and experimentation of bone drilling forces." *Journal of biomechanics* 45.6 (2012): 1076-1083.

[8] Taha, Zahari, A. Salah, and J. Lee. "Bone breakthrough detection for orthopedic robot-assisted surgery." *APIEMS 2008 Proceedings of the 9th Asia Pacific Industrial Engineering and Management Systems Conference*. 2008.

[9] Wiggins, K. L., and S. Malkin. "Drilling of bone." *Journal of biomechanics* 9.9 (1976): 553-559.

[10] Hobkirk, J. A., and K. Rusiniak. "Investigation of variable factors in drilling bone." *Journal of oral surgery (American Dental Association: 1965)* 35.12 (1977): 968-973.

[11] B. Allotta, F. Belmonte, L. Bosio, and P. Dario, "Study on a mechatronic tool for drilling in the osteosynthesis of long bones: tool/bone interaction, modeling,and experiments", *Mechatronics*, Vol. 6, No. 4, 1996, pp. 447-459.

[12] Zhengyi, Yang, and Y. Chen. "Haptic rendering of milling." *Mechanical Engineering Department, The University of Hong Kong, Hong Kong China.*

[13] Yonghua Chen; Xuejian He, "Haptic simulation of bone drilling based on hybrid 3D part representation," *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2013 IEEE International Conference on*, vol., no., pp.78,81, July 2013.

[14] J Armstrong-Helouvry B., Dupont P. and De Wit C., 1994. "A survey of models, analysis tools and compensation method for the control of machines with friction", Automatica, Vol. 30, No.7 (1994) pp. 10831138.

[15] K. Ohnishi, M. Shibata, T. Murakami: "Motion control for advancedmechatronics," Mechatronics, IEEE/ASME Transactions on, vol.1, no.1, pp.56-67, Mar 1996.

[16] Katsura, S.; Irie, K.; Ohishi, K.; "Wideband Force Control by Position-Acceleration Integrated Disturbance Observer," *Industrial Electronics,IEEE Transactions on*, vol.55, no.4, pp.1699-1706, April 2008.

[17] T.Murakami, K.Ohnishi, (1993) "Disturbance Observer Based Motion Control- Application to Robust Control and Parameter Identification", IEEE Transaction on Industrial Electronics.

[18] Katsura.;Y.Matsumoto,K.Ohnishi:"Modelingofforcesensingandvalidationof disturbanceobserverforforcecontrol,"*IndustrialElectronicsSociety,2003. IECON'03.The29thAnnualConferenceoftheIEEE*,vol.1,no.,pp.291-296vol.1,2-6Nov.2003.

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

[20] Asif Sabanovic, Kouhei Ohnishi (2011) *Motion control systems*, IEEE press John Willey and sons (Asia) pte Ltd, (First Edition).

[21] A.M Harsha S.Abeykoon, Kouhei Ohnishi: "Improvement of Tactile Sensation of a Bilateral Forceps Robot by a Switching Virtual Model," *Transaction on Advanced Robotics*", Vol. 8, pp. 789-806 (18), 2008.

[22] A.M Harsha S.Abeykoon, Kouhei Ohnishi: "Virtual tool for bilaterally controlled forceps robot-for minimally invasive surgery," *Transaction on international Journal of Medical Robotics and Computer Assisted Surgery*", ISSN 1478-5951, VOL 3; No 3, pp. 271-280, 2007.

[23] www.mbed.org

[24] https://aptinex.com/

APPENDIX A: INERTIA ESTIMATION

```
#include "mbed.h"
#include "rtos.h"
#include "qeihw.h"
#include "SDFileSystem.h"
#include "math.h"
//#define QEI_RESET_POSOnIDX        QEI_CON_RESPI

QEIHW qei_s(QEI_DIRINV_NONE, QEI_SIGNALMODE_QUAD,
QEI_CAPMODE_4X, QEI_INVINX_NONE );
Serial pc(USBTX, USBRX);
#define G1 1.0//100.0
#define G2 1.0 //500.0

// Safety for mbed unused pins
AnalogIn   current_m1(p16);
AnalogIn   current_s1(p20);
AnalogIn   M_encorder(p18);
AnalogIn   current_s(p19);
AnalogIn   current_m(p17);
//DigitalIn safety_19(p19);
//DigitalIn safety_25(p25);
//DigitalIn safety_26(p26);
float currentvalue1=0;
float M_position=0;
float t1=0.0,t2 =0.0;
float g=10;


float q11=0.0,q12 =0.0; // current sensor old master filter
float g21=10;


float r11=0.0,r12 =0.0; // current sensor new master filter
float g31=10;

float t11=0.0,t12 =0.0; // current sensor old slave filter
float g1=10;

float j11=0.0,j12 =0.0; // current sensor motor drive slave filter
float gg1=10;
```

```
float M_pos=0.0;
PwmOut pwm_s(p23);  // clockwise rotation pwm pin for SLAVE
PwmOut pwm_m(p24);
DigitalOut  EnableS(p25);
DigitalOut  EnableM(p26);
DigitalOut  anticlk_M(p21);
DigitalOut  clk_M(p22);
DigitalOut  clk_S(p13);
DigitalOut  anticlk_S(p14);


//AnalogIn current_sensor_s_p(p17); // current sensor input for SLAVE positive
//AnalogIn current_sensor_s_n(p18); // current sensor input for SLAVE negative

DigitalOut led1(LED1);
DigitalOut led3(LED3);

//DigitalIn signal(p20);  //external signal to close the file

Timer timer;                // For the controller
FILE *fp;
Ticker ticker;
//Mutex stdio_mutex;

int dt_us= 300;             // define main loop time in us
float dt = dt_us/1000000.0;     //loop time in seconds for calculations
float rpm = 0.0, ramp_time = 0.0;
float delta_v = 0.0;
float M_cur =0.0;
int counter_time;
int counter =0;
int counter_old =0;

//Current Sensor Directions

int Slave_Direction = 0;

// Encoder Constants

float const encoder_pulses_s = 800.0;

#define PI 3.141592653
#define Gd 0.5 //1200.0 //cutoff frequency of the DOB
//#define Gv 0.0
```

```
// PID parameters for Current - Loop

float const Ikp_s = 0.04, Iki_s = 0.00001, Ikd_s = 0.001;
float const Ikp_m = 0.45, Iki_m = 1.0, Ikd_m = 1.34;
// PID parameters for velocity  - Loop

float I_ref_s = 0.0, I_err_s = 0.0, I_res_s = 0.0, I_tmp_s = 0.0, tem_I_s = 0.0, d_I_s =
0.0,I1_act_s=0.0;

float I_ref_m = 0.0, I_err_m = 0.0, I_res_m = 0.0, I_tmp_m = 0.0, tem_I_m = 0.0,
d_I_m = 0.0,I1_act_m=0.0;
float v_ref_s = 0.0, v_err_s = 0.0, v_res_s = 0.0,v_res_s_rpm = 0.0, v_tmp_s = 0.0,
tem_v_s = 0.0, d_v_s = 0.0;
float duty_s = 0.0;
float  vkp_s = 01.000, vki_s = 1.0000000, vkd_s =1.0;
float  d_v_err_s = 0.0, prev_v_err_s=0.0,I_v_err_s=0.0;


float dde_s = 0.0, v_res_s_prv=0.0,accelaration_s=0.00,temp_s2=0.0,temp_s3
=0.0,acce_res_s =0.0;

  // Low pass filter gain for Current - loop
float const G_filcon_I_s = 1.0;
float const G_filcon_I_m = 1.0;
  // Low pass filter gain for Current - loop
float const G_filcon_I1_s = 100.0;


   // Storing actual current flow
float I_act_s = 0.0;
float I_act_m = 0.0;
    // Parameters to calculate current rotational speed
float encoder_s_prv = 0.0;
float encoder_s_now = 0.0;

// Motor Constant and Inertia
float const J_const_s = 0.0000800; //0.0000268;
float const Kt_const_s = 0.134;
float const Kt_constinv_s = 1.0/0.134;


float tmp_s = 0.0,ob_sum_s = 0.0,ob_sum_s1=0.0;

float i_com_s = 0.0;
float fric_m = 0.0,fric_s = 0.0,i_rto_m = 0.0,i_rto_s = 0.0;
```

```
float x_res_s = 0.0;
float ve_sum_s = 0.0;
float pwm_I_S= 0.0;
float pid_V_I_S= 0.0;

float point =0.0;
float de_s =0.0, temp_s1=0.0, v_res_s1 = 0.0, temp_s= 0.0, ddx_s = 0.0;

float pos_ref=0.0, pos_err=0.0, p_pos_err=0.0, i_pos_err=0.0;
float tem_d_pos_err=0.0, tmp_d_pos_err=0.0, d_pos_err=0.0, PWM_pos=0.0;

float position = 0;
float duty_m = 0.0, pwm_I_M= 0.0, pid_V_I_M= 0.0;

 float Int_tau = 0.0,JB = 0.0, theta_1 = 0.0, theta_2 = 0.0, a = 0.0,G_filcon_v_s = 0.0
;
 float t = 0.0 ,cur = 0.0, tau_dob = 0.0 ,integral_dob = 0.0, Gdr = 55.0;
 float Kt = 0.151, JN1 = 0.0 ;
float S_cur=0.0;
float S_cur1=0.0;
float M_cur1=0.0;
float Ires_s2 = 0.0; // Current sensor S2
float Ires_s3 = 0.0; // Current sensor S2
float I_res_m1 =0.0;
float I_res_m2 =0.0;
/*
 void pwm_init(void) {

    pwm_s_anticlk.period_us(5);
    pwm_s_clk.period_us(5);


    // Set the ouput duty-cycle, specified as a percentage (float)

    pwm_s_anticlk.write(0.0f);
    pwm_s_clk.write(0.0f);

   //ENABLE RUNNING MODE  (H BRIDGE ENABLE)

    Reset_AB_S = 1;
    Reset_CD_S = 1;
}*/
void pwm_init(void) {

     pwm_m.period_us(50);
```

```
    pwm_s.period_us(50);


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

    pwm_s.write(0.0f);


   // Reset_AB_M = 1;              //ENABLE RUNNING MODE  (H BRIDGE
ENABLE)
   EnableS=1;
   EnableM=1;
}

/*
void position_calc()  {

qei_s.SetDigiFilter(480UL);
qei_s.SetMaxPosition(0xFFFFFFFF);

int main{

position_control(int abs_position){



while(1){

position = qei_s.GetPosition();

if (position_old == position){
I_ref += 0.00000001;
counter = I_ref;
position_old= position;
current_pid_s();

}

else (position_old != position) {
cout << counter;
position_old++;
}

if (position_old != position){
position_control(position_old);
```

```
}

}
}*/

float kp_pos= 0.01;//0.00006;
float ki_pos = 0.008;//0.0008;
float kd_pos =0.000002; //0.00649999999;
float pos_filcon = 100;

void position_com_s() {
pos_ref = 800;
position = qei_s.GetPosition();

}
/*
void position_pid_s() {

int32_t position = 0;
qei_s.SetDigiFilter(480UL);
qei_s.SetMaxPosition(0xFFFFFFFF);

position = qei_s.GetPosition();

pos_err = pos_ref - position;
p_pos_err = kp_pos*pos_err;
i_pos_err += pos_err*dt;
tem_d_pos_err += tmp_d_pos_err*dt;
tmp_d_pos_err = (position - tem_d_pos_err)*pos_filcon;
d_pos_err = kd_pos*tmp_d_pos_err;

PWM_pos = p_pos_err ;//+ ki_pos*i_pos_err + d_pos_err;

}*/



/*
void position_pid_m() {
pos_ref = v_res_s;
//rpm = (5000/0.48)*(M_pos-0.20);
position = rpm;
pos_err = pos_ref - position;
p_pos_err = kp_pos*pos_err*2;
i_pos_err += pos_err*dt;
tem_d_pos_err += tmp_d_pos_err*dt;
```

```
tmp_d_pos_err = (position - tem_d_pos_err)*pos_filcon;
d_pos_err = kd_pos*tmp_d_pos_err;

pwm_I_M = p_pos_err ; //+ ki_pos*i_pos_err + d_pos_err;

}
*/
int compare(const void *a, const void *b) {
   return (int)(*(float*)a - *(float*)b);
}

float median(float *samples, int n) {
   qsort(samples, n, sizeof(float), compare);
   return samples[(n - 1)/2];
}
void velocity_command(){

  //float sampleanalog[5];
 // for (int jj=0; jj<5 ; jj++){

   //   sampleanalog[jj]=M_encorder.read();
    //  }
 /*
 //M_position= median(sampleanalog,5);
M_position = M_encorder.read();
//currentvalue1= ( 0.5- currents1.read())* 27.03 ;
//currentvalue1= currents1.read();
t1=t1+g*M_position*dt;
t2=t2+g*M_pos*dt;
M_pos = t1-t2;

  //rpm = (64 - M_pos*100)*100;
  rpm = (5000/0.48)*(M_pos-0.20);
  //rpm = 160;
//if(counter<=100000){
 delta_v = 0.5;
 if(abs(v_ref_s) < abs(rpm)){
 v_ref_s += delta_v;}
 else {
 v_ref_s=rpm;}
 //}
 */

if (t <= ramp_time){
          rpm = +8000.0;
          theta_1 = rpm*0.104719755;
```

```
            }
            else if (t >= (ramp_time +0.01)){
            rpm = -8000.0;
            theta_2 = rpm*0.104719755;
            }
            ramp_time = 3.0;
            delta_v = (rpm*dt)/ramp_time;
           //(v_ref_s - rpm)*100) >
            if (abs((v_ref_s - rpm))> 0.18){
            //if (abs(dthe_cmd) < abs(rpm)){
            v_ref_s += delta_v;
            }
            else{
            v_ref_s = rpm;
            }

 // v_ref_s = 500;
}
void velocity_pid_s() {

    int32_t position = 0;
    qei_s.SetDigiFilter(480UL);
    qei_s.SetMaxPosition(0xFFFFFFFF);
    point = qei_s.GetIndex();
    position = qei_s.GetPosition();
    encoder_s_now = position * 2.0 * PI / encoder_pulses_s;
    de_s = encoder_s_now - encoder_s_prv;
    encoder_s_prv = encoder_s_now;
    //v_res_s = ((de_s/dt_us)*(1000000.0 * 60.0))/(2.0*PI);
    ddx_s=((de_s/dt_us)*(1000000.0 * 60.0))/(2.0*PI);
    temp_s += G1*ddx_s*dt;
    temp_s1 += G2*v_res_s*dt;
    v_res_s = temp_s - temp_s1;
          /* V_err = dthe_cmd - j[2].dthe; //V_res;
           //V_err= dthe_cmd - V_res;
           D_V_err = GLPF*(V_err- Prev_V_err)*ST;
           Prev_V_err += D_V_err;
           I_V_err += V_err * ST;
           cur = Kp*V_err*1.0 + Kd*D_V_err*100.0 + Ki*I_V_err*50.0;*/
    v_err_s = v_ref_s - v_res_s;
    //d_v_err_s = (v_err_s - prev_v_err_s) * G_filcon_v_s;
   /* d_v_err_s = G_filcon_v_s*(v_err_s - prev_v_err_s) *dt_us ;
    prev_v_err_s += d_v_err_s;*/
     d_v_err_s = (v_err_s - prev_v_err_s) /dt ;
    prev_v_err_s = v_err_s;
    I_v_err_s += v_err_s * dt;
```

```
   //cur = vkp_s * v_err_s*0.035 + vkd_s * d_v_err_s*0.001 + vki_s *
I_v_err_s*.009;
    cur = vkp_s * v_err_s*2700 + vkd_s * d_v_err_s*102+ vki_s * I_v_err_s*500;
   //pid_V_I_S= vkp_s * v_err_s + vkd_s * d_v_err_s + vki_s * I_v_err_s;
   //PWM_pos = pid_V_I_S;
 pwm_I_S=cur/1000000;
}

void dob_rotary()
{
   tau_dob = integral_dob - v_res_s*0.104719755*JN1*Gdr;
   integral_dob += ((I_act_s*Kt + v_res_s*JN1*Gdr*0.104719755) -
integral_dob)*Gdr*dt;
   return;
}



void Inertia_s() {
 /*** Inertia Estimation****/
       if ((t >=(ramp_time+0.01)) && (((3*ramp_time)+0.01)>= t )){

           Int_tau += tau_dob*dt;
           JB = Int_tau/(theta_2 - theta_1);

           }

}


/*

float B=0.0;

float F=0.0215;
float T_ref_s=0.0;

 void I_com(){

   //T_ref_s = (B*v_res_s + F) ;
  // I_ref_s = T_ref_s/ Kt_const_s;
 I_ref_s =5;
 I_ref_m =0.02;
 }
*/
 void current_pid_s(){
```

```
   S_cur1 =  current_s1.read();
    j11=j11+gg1*10*S_cur1*dt;
j12=j12+gg1*10*Ires_s2*dt;
Ires_s2 = j11-j12;
Ires_s3 =Ires_s2 *3.3/0.140;

   I_res_s =  current_s.read();
    //Slave_Direction = S_Dir.read();
    if(I_res_s <0.757){                                  //slave clockwise
        //I_res_s =  current_sensor_s_p.read();
      //  I1_act_s = -3.0*((0.757-I_res_s) / 0.1);
       I1_act_s = Ires_s3;
    }else if(I_res_s >0.757){
        //I_res_s =  current_sensor_s_n.read();              //slave anticlockwise
       // I1_act_s = 3*((I_res_s-0.757) / 0.1);
       I1_act_s = -1*Ires_s3;
    }


     I_act_s += G_filcon_I1_s*(I1_act_s-I_act_s)*dt;

    I_err_s =  cur - I_act_s;
    I_tmp_s += Iki_s * dt * I_err_s;
    tem_I_s += dt * d_I_s;
    d_I_s = (I_act_s - tem_I_s) * G_filcon_I_s;

  // pwm_I_S=((I_err_s * Ikp_s) + I_tmp_s + (d_I_s * Ikd_s));

    //pwm_I_S=I_ref_s;
}


/*
void current_pid_m(){

  // Master_Direction = M_Dir.read();
    M_cur =  current_m.read();
    t11=t11+g1*10*M_cur*dt;
t12=t12+g1*10*I_res_m*dt;
I_res_m = t11-t12;

   if(I_res_m <0.76){                                   //master clockwise
        //I_res_s =  current_sensor_s_p.read();
        I1_act_m = -1.0*((0.76-I_res_m) / 0.10);
    }else if(I_res_m >0.76){
        //I_res_s =  current_sensor_s_n.read();              //master anticlockwise
```

```
      I1_act_m = ((I_res_m-0.76) / 0.10);
   }



   I_act_m = I1_act_m;

   I_err_m = I_ref_m - I_act_m;
   I_tmp_m += (Iki_m * dt * I_err_m);

   tem_I_m += G_filcon_I_m*d_I_m*dt;
   d_I_m = I_err_m - tem_I_m;

   pwm_I_M=((I_err_m * Ikp_m) + I_tmp_m + (d_I_m * Ikd_m));
 }
*/
/*
void PWM_Generator_m() {
duty_m =  -pwm_I_M;
//duty_m = 0.5;
   if (duty_m> 0.0) {
      if (duty_m > 0.95) {
         duty_m = 0.98;
      }
      pwm_m_clk = 0.0;
      pwm_m_anticlk = duty_m;
   }

   if (duty_m < 0.0) {
      if (duty_m < -0.95) {
         duty_m = -0.98;
      }
      pwm_m_anticlk = 0.0;
      pwm_m_clk = -1.0 * duty_m;
   }
}
*/



void PWM_Generator_s(){
duty_s = pwm_I_S;

//duty_s = pwm_I_S+ i_com_s;//current pid
  // duty_s = 0.65;
   if (duty_s > 0.0) {
```

```
      if (duty_s > 0.98) {
         duty_s = 0.98;
      }
       clk_S=1;
   anticlk_S=0;

      pwm_s = duty_s;
   }

   if (duty_s < 0.0) {
      if (duty_s < -0.98) {
         duty_s = -0.98;
      }
    clk_S=0;
   anticlk_S=1;
      pwm_s = -1.0*duty_s;
   }

}



 int cleanup_module(void){

   pwm_m = 0.0;         // pwm cleanup module

   pwm_s = 0.0;


  // Reset_AB_M = 0;      //RESET H BRIDGE
   EnableM=0;
   EnableS=0;
   led1=0;
   led3=0;

   return 0;
}



void Control_body() {
                        // Control Part - main code
     velocity_command(); // ---------slave
      velocity_pid_s();// ---------slave
      Inertia_s();
     // I_com();
```

```
      current_pid_s ();
    // current_pid_m ();
      // position_pid_m();//----------master
       //position_com_s();
       //position_pid_s();
       dob_rotary();
       PWM_Generator_s();
       //PWM_Generator_m();
      // Disob();
        t += dt;
        led1=!led1;
         counter++;
   }


void thread_2(void const *argument){
 //pc.printf("Pulses is: %f\n", M_encorder.read() );
 //  wait(0.2);

/*
        SDFileSystem sd(p5, p6, p7, p8, "sd");
         FILE *fp = fopen("/sd/v100.dat", "w");
 if(fp == NULL) {
         for(int i=0;i<5;i++){
        led3=!led3;
         wait(1.0);
            }
         }
*/
 while(1){//counter<10000
   if(counter>=(counter_old+100)){
   //M_position= M_encorder.read();

 // rpm = M_position*400;
        //fprintf(fp,"%d %f %f %f \n",timer.read_us(),I_act_s,I_ref_s, i_com_s );
         // fprintf(fp,"%d %f \n",timer.read_us(),position);
  // pc.printf("current m: %f\n", I_act_s );
   // pc.printf("cur: %f\n", cur );
 //pc.printf("V_ref: %f\n", v_ref_s );
 // pc.printf("V_res: %f\n", v_res_s );
     //  pc.printf("%f %f\n", v_res_s,t );
    // // pc.printf("delta_v: %f\n", delta_v );
     // pc.printf("t : %f\n", t );
    // pc.printf(" %f %f\n", t, JN1+JB );
     pc.printf(" %f %f %f %f %f %f\n", t, JN1+JB,I_act_s,tau_dob ,v_ref_s,v_res_s);
    //  pc.printf("PWM S: %f\n", pwm_I_S );
```

```
      //pc.printf("currentmn: %f\n", current_mn.read() );
   //wait(0.5);
        counter_old=counter;
       // led3=!led3;
          }
        }


      fclose(fp);
      timer.stop();
      cleanup_module();
      ticker.detach ();


   }

int main() {
    //pc.baud(115200);
    dt =dt_us/1000000.0;

    pwm_init();
    timer.start();


    Thread thread2(thread_2, NULL, osPriorityNormal, (DEFAULT_STACK_SIZE *
4));

    ticker.attach_us(&Control_body, dt_us);
    // pc.printf("Pulses is: %d\n", position);

    //wait(1.0);

   }
```

APPENDIX B: HAPTIC BASED ORTHOPEDIC DRILL

```
#include "mbed.h"
#include "rtos.h"
#include "qeihw.h"
#include "SDFileSystem.h"
#include "math.h"
//#define QEI_RESET_POSOnIDX        QEI_CON_RESPI


//y = 0.192x + 0.115 torque graph vs pwm
// force factor 1/m kc
// 1/0.019

float Kc = 5.263;

QEIHW qei_s(QEI_DIRINV_NONE, QEI_SIGNALMODE_QUAD,
QEI_CAPMODE_4X, QEI_INVINX_NONE );
Serial pc(USBTX, USBRX);
#define G1 1.0//100.0
#define G2 1.0 //500.0

// Safety for mbed unused pins

AnalogIn   v_pos(p15);
AnalogIn   current_m1(p16);
AnalogIn   current_s1(p20);
AnalogIn   M_encorder(p18);
AnalogIn   current_s(p19);
AnalogIn   current_m(p17);
float vrtcl_pos=0;
float vrtclmm=0;
float currentvalue1=0;
float M_position=0;
float t1=0.0,t2 =0.0;
float g=10;
float q11=0.0,q12 =0.0; // current sensor old master filter
float g21=10;
float f_s= 0.110;
float B_s=0.00002;

float r11=0.0,r12 =0.0; // current sensor new master filter
float g31=10;
```

```
float t11=0.0,t12 =0.0; // current sensor old slave filter
float g1=10;

float j11=0.0,j12 =0.0; // current sensor motor drive slave filter
float gg1=10;

float M_pos=0.0;
float G_filcon_v_s=100;

float theta =0;
float Rtob_S=0;
PwmOut pwm_s(p23);  // clockwise rotation pwm pin for SLAVE
PwmOut pwm_m(p24);
DigitalOut  EnableS(p25);
DigitalOut  EnableM(p26);
DigitalOut  anticlk_M(p21);
DigitalOut  clk_M(p22);
DigitalOut  clk_S(p13);
DigitalOut  anticlk_S(p14);
//DigitalIn M_Dir(p9);
//DigitalIn S_Dir(p10);

//DigitalIn Index(p12);

//AnalogIn current_sensor_s_p(p17); // current sensor input for SLAVE positive
//AnalogIn current_sensor_s_n(p18); // current sensor input for SLAVE negative

DigitalOut led1(LED1);
DigitalOut led3(LED3);

//DigitalIn signal(p20);  //external signal to close the file

Timer timer;                // For the controller
FILE *fp;
Ticker ticker;
//Mutex stdio_mutex;

int dt_us= 250;             // define main loop time in us
float dt = dt_us/1000000.0;     //loop time in seconds for calculations
float rpm = 0.0, ramp_time = 0.0;
float delta_v = 0.0;
float M_cur =0.0;
int counter_time;
int counter =0;
int counter_old =0;
```

```
//Current Sensor Directions

int Slave_Direction = 0;

// Encoder Constants

float const encoder_pulses_s = 800.0;

#define PI 3.141592653
#define Gd 0.5 //1200.0 //cutoff frequency of the DOB
//#define Gv 0.0


// PID parameters for Current - Loop

//float const Ikp_s = 4.5, Iki_s = 10.0, Ikd_s = 0.1;

float const Ikp_s = 1.0, Iki_s = 1.0, Ikd_s = 1.0;
float const Ikp_m = 0.45, Iki_m = 1.0, Ikd_m = 1.34;
// PID parameters for velocity  - Loop

float I_ref_s = 0.0, I_err_s = 0.0, I_res_s = 0.0, I_tmp_s = 0.0, tem_I_s = 0.0, d_I_s =
0.0,I1_act_s=0.0;

float I_ref_m = 0.0, I_err_m = 0.0, I_res_m = 0.0, I_tmp_m = 0.0, tem_I_m = 0.0,
d_I_m = 0.0,I1_act_m=0.0;
float v_ref_s = 60.0, v_err_s = 0.0, v_res_s = 20.0,v_res_s_rpm = 0.0, v_tmp_s =
0.0, tem_v_s = 0.0, d_v_s = 0.0;
float duty_s = 0.0;
float  vkp_s = 0.0002, vki_s = 0.00011, vkd_s =0.00056;
float  d_v_err_s = 0.0, prev_v_err_s=0.0,I_v_err_s=0.0;


float dde_s = 0.0, v_res_s_prv=0.0,accelaration_s=0.00,temp_s2=0.0,temp_s3
=0.0,acce_res_s =0.0;

  // Low pass filter gain for Current - loop
float const G_filcon_I_s = 1.0;
float const G_filcon_I_m = 1.0;
  // Low pass filter gain for Current - loop
float const G_filcon_I1_s = 100.0;


   // Storing actual current flow
float I_act_s = 0.0;
```

```
float I_act_m = 0.0;
    // Parameters to calculate current rotational speed
float encoder_s_prv = 0.0;
float encoder_s_now = 0.0;

// Motor Constant and Inertia
float const J_const_s = 0.0000800; //0.0000268;
float const Kt_const_s = 0.134;
float const Kt_constinv_s = 1.0/0.134;


float tmp_s = 0.0,ob_sum_s = 0.0,ob_sum_s1=0.0;

float i_com_s = 0.0;
float fric_m = 0.0,fric_s = 0.0,i_rto_m = 0.0,i_rto_s = 0.0;


float x_res_s = 0.0;
float ve_sum_s = 0.0;
float pwm_I_S= 0.0;
float pid_V_I_S= 0.0;

float point =0.0;
float de_s =0.0, temp_s1=0.0, v_res_s1 = 0.0, temp_s= 0.0, ddx_s = 0.0;

float pos_ref=0.0, pos_err=0.0, p_pos_err=0.0, i_pos_err=0.0;
float tem_d_pos_err=0.0, tmp_d_pos_err=0.0, d_pos_err=0.0, PWM_pos=0.0;

float position = 0;
float duty_m = 0.0, pwm_I_M= 0.0, pid_V_I_M= 0.0;


 float Int_tau = 0.0,JB = 0.0, theta_1 = 0.0, theta_2 = 0.0, a = 0.0;
 float t = 0.0 ,cur = 0.0, tau_dob = 0.0 ,integral_dob = 0.0, Gdr = 55.0;
 float Kt = 0.151, JN1 = 0.000260; //660-46
float S_cur=0.0;
float S_cur1=0.0;
float M_cur1=0.0;
float Ires_s2 = 0.0; // Current sensor S2
float Ires_s3 = 0.0; // Current sensor S2
float I_res_m1 =0.0;
float I_res_m2 =0.0;
 int Master_Direction=0;
/*
 void pwm_init(void) {
```

```
    pwm_s_anticlk.period_us(5);
    pwm_s_clk.period_us(5);

    // Set the ouput duty-cycle, specified as a percentage (float)

    pwm_s_anticlk.write(0.0f);
    pwm_s_clk.write(0.0f);

   //ENABLE RUNNING MODE  (H BRIDGE ENABLE)

    Reset_AB_S = 1;
    Reset_CD_S = 1;
}*/
void pwm_init(void) {

    pwm_m.period_us(50);
    pwm_s.period_us(50);


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

    pwm_s.write(0.0f);


   // Reset_AB_M = 1;             //ENABLE RUNNING MODE  (H BRIDGE
ENABLE)
   EnableS=1;
   EnableM=1;


}

/*
void position_calc()  {

qei_s.SetDigiFilter(480UL);
qei_s.SetMaxPosition(0xFFFFFFFF);

int main{

position_control(int abs_position){



while(1){
```

```
position = qei_s.GetPosition();

if (position_old == position){
I_ref += 0.00000001;
counter = I_ref;
position_old= position;
current_pid_s();

}

else (position_old != position) {
cout << counter;
position_old++;
}

if (position_old != position){
position_control(position_old);
}

}
}*/

float kp_pos= 0.01;//0.00006;
float ki_pos = 0.008;//0.0008;
float kd_pos =0.000002; //0.00649999999;
float pos_filcon = 100;

void position_com_s() {
pos_ref = 1000;
position = qei_s.GetPosition();

}
/*
void position_pid_s() {

int32_t position = 0;
qei_s.SetDigiFilter(480UL);
qei_s.SetMaxPosition(0xFFFFFFFF);

position = qei_s.GetPosition();

pos_err = pos_ref - position;
p_pos_err = kp_pos*pos_err;
i_pos_err += pos_err*dt;
tem_d_pos_err += tmp_d_pos_err*dt;
```

```
tmp_d_pos_err = (position - tem_d_pos_err)*pos_filcon;
d_pos_err = kd_pos*tmp_d_pos_err;

PWM_pos = p_pos_err ;//+ ki_pos*i_pos_err + d_pos_err;

}*/

void position_pid_m() {
//pos_ref = v_res_s;
/*pos_ref = v_res_s*.48/5000+ 0.3;
//rpm = (5000/0.48)*(M_pos-0.20);
position = M_pos-0.3;
pos_err = pos_ref - position;
p_pos_err = kp_pos*pos_err*2;
i_pos_err += pos_err*dt;
tem_d_pos_err += tmp_d_pos_err*dt;
tmp_d_pos_err = (position - tem_d_pos_err)*pos_filcon;
d_pos_err = kd_pos*tmp_d_pos_err;*/

    M_cur =  current_m.read();
    q11=q11+g21*10*M_cur*dt;
q12=q12+g21*10*I_res_m*dt;
I_res_m = q11-q12;

   if(I_res_m <0.7609){                                    //master clockwise
       //I_res_s =  current_sensor_s_p.read();
       I1_act_m = -1.0*((0.7609-I_res_m) / 0.10);
    }else if(I_res_m >0.769){
       //I_res_s =  current_sensor_s_n.read();                 //master anticlockwise
       I1_act_m = ((I_res_m-0.7609) / 0.10);
    }


    //-------------------------------------Current sensor motor board
      M_cur1 =  current_m1.read();
    r11=r11+g31*10*M_cur1*dt;
    r12=r12+g31*10*I_res_m1*dt;
I_res_m1 = r11-r12;
 I_res_m2 = I_res_m1 *3.3/0.110;


pwm_I_M = -1*(0.0002*v_res_s*0.104719755+0.8*Rtob_S); //p_pos_err ; //+
ki_pos*i_pos_err + d_pos_err;

}
```

```
int compare(const void *a, const void *b) {
   return (int)(*(float*)a - *(float*)b);
}

float median(float *samples, int n) {
   qsort(samples, n, sizeof(float), compare);
   return samples[(n - 1)/2];
}
void velocity_command(){

  //float sampleanalog[5];
 // for (int jj=0; jj<5 ; jj++){

   //   sampleanalog[jj]=M_encorder.read();
    //  }

 //M_position= median(sampleanalog,5);
M_position = M_encorder.read();
//currentvalue1= ( 0.5- currents1.read())* 27.03 ;
//currentvalue1= currents1.read();
t1=t1+g*M_position*dt;
t2=t2+g*M_pos*dt;
M_pos = t1-t2;

// rpm = (5000/0.48)*(M_pos-0.30);
 rpm = 5000;
 // theta = (M_pos-0.30)*2;
 // rpm = theta;
//if(counter<=100000){
 delta_v = 0.5;
 if(abs(v_ref_s) < abs(rpm)){
 v_ref_s += delta_v;}
 else {
 v_ref_s=rpm;}
 //}
}
void velocity_pid_s() {

   int32_t position = 0;
   qei_s.SetDigiFilter(480UL);
   qei_s.SetMaxPosition(0xFFFFFFFF);
   point = qei_s.GetIndex();
   position = qei_s.GetPosition();
   encoder_s_now = position * 2.0 * PI / encoder_pulses_s;
   de_s = encoder_s_now - encoder_s_prv;
   encoder_s_prv = encoder_s_now;
```

```
   //v_res_s = ((de_s/dt_us)*(1000000.0 * 60.0))/(2.0*PI);
   ddx_s=((de_s/dt_us)*(1000000.0 * 60.0))/(2.0*PI);
   temp_s += G1*ddx_s*dt;
   temp_s1 += G2*v_res_s*dt;
   v_res_s = temp_s - temp_s1;
   v_err_s = v_ref_s - v_res_s;
   //d_v_err_s = (v_err_s - prev_v_err_s) * G_filcon_v_s;
  /*d_v_err_s = G_filcon_v_s*(v_err_s - prev_v_err_s) *dt_us ;
   prev_v_err_s += d_v_err_s;*/
   d_v_err_s = (v_err_s - prev_v_err_s) /dt ;
   prev_v_err_s = v_err_s;
   I_v_err_s += v_err_s * dt;
   //pid_V_I_S= vkp_s * v_err_s + vkd_s * d_v_err_s + vki_s * I_v_err_s;
     pid_V_I_S=  v_err_s *900 + d_v_err_s *60+ I_v_err_s*500;
   PWM_pos = pid_V_I_S/1000000;

    //  cur = vkp_s * v_err_s*2700 + vkd_s * d_v_err_s*102+ vki_s *
I_v_err_s*500;
   //pid_V_I_S= vkp_s * v_err_s + vkd_s * d_v_err_s + vki_s * I_v_err_s;
   //PWM_pos = pid_V_I_S;


}

float B=0.0;

float F=0.0215;
float T_ref_s=0.0;

 void I_com(){

   //T_ref_s = (B*v_res_s + F) ;
  // I_ref_s = T_ref_s/ Kt_const_s;
//I_ref_s =5;
I_ref_m =0.02;
 }

 void current_pid_s(){
 // S_cur1=current_s1.read()*3.3/0.140;
 vrtcl_pos=v_pos.read()*100;
    if(vrtcl_pos <20){                                  //slave clockwise

        vrtclmm =vrtcl_pos;
   }else if(vrtcl_pos >20){
     vrtclmm =vrtcl_pos-5.5;
   }
```

```
  //----------------------------------sensor old
    S_cur =  current_s.read();
   t11=t11+g1*10*S_cur*dt;
t12=t12+g1*10*I_res_s*dt;
I_res_s = t11-t12;

  //---------------------------------------Sensor motor board
     S_cur1 =  current_s1.read();
   j11=j11+gg1*10*S_cur1*dt;
j12=j12+gg1*10*Ires_s2*dt;
Ires_s2 = j11-j12;
Ires_s3 =Ires_s2 *3.3/0.140;

   //I_res_s =  current_s.read();
    //Slave_Direction = S_Dir.read();


    if(I_res_s <0.744){                                  //slave clockwise
       //I_res_s =  current_sensor_s_p.read();
       ///I1_act_s = -2.5*((0.744-I_res_s) / 0.1);
         I1_act_s =Ires_s3;
    }else if(I_res_s >0.744){
       //I_res_s =  current_sensor_s_n.read();                 //slave anticlockwise
      //  I1_act_s = 2.5*((I_res_s-0.744) / 0.1);
          I1_act_s =  -1*Ires_s3;
    }

    I_act_s=I1_act_s;
   // I_ref_s = (pid_V_I_S/10000000 + tau_dob/Kt);

    // I_act_s += G_filcon_I1_s*(I1_act_s-I_act_s)*dt;
    /*
    I_err_s = I_ref_s - I_act_s;
    I_tmp_s += Iki_s * dt * I_err_s;
    tem_I_s += dt * d_I_s;
    d_I_s = (I_act_s - tem_I_s) * G_filcon_I_s;

   pwm_I_S=((I_err_s * Ikp_s) + I_tmp_s + (d_I_s * Ikd_s));
   */
  /*
   I_err_s = I_ref_s - I_act_s;
    I_tmp_s +=  I_err_s*dt;
    d_I_s = I_err_s/dt;

   pwm_I_S=((I_err_s * Ikp_s*4) + (d_I_s * Ikd_s*2) + I_tmp_s*Iki_s );
```

```
   //pwm_I_S=I_ref_s;*/
}

void dob_rotary()
{
   tau_dob = integral_dob - v_res_s*0.104719755*JN1*Gdr ;

   integral_dob += ((I_act_s*Kt + v_res_s*JN1*Gdr*0.104719755) -
integral_dob)*Gdr*dt;

  Rtob_S = tau_dob - f_s - B_s*v_res_s;

   return;
}


/*
void current_pid_m(){

  // Master_Direction = M_Dir.read();
   M_cur =  current_m.read();
   t11=t11+g1*10*M_cur*dt;
t12=t12+g1*10*I_res_m*dt;
I_res_m = t11-t12;

  if(I_res_m <0.76){                                  //master clockwise
      //I_res_s =  current_sensor_s_p.read();
      I1_act_m = -1.0*((0.76-I_res_m) / 0.10);
   }else if(I_res_m >0.76){
      //I_res_s =  current_sensor_s_n.read();                 //master anticlockwise
      I1_act_m = ((I_res_m-0.76) / 0.10);
   }



   I_act_m = I1_act_m;

   I_err_m = I_ref_m - I_act_m;
   I_tmp_m += (Iki_m * dt * I_err_m);

   tem_I_m += G_filcon_I_m*d_I_m*dt;
   d_I_m = I_err_m - tem_I_m;

   pwm_I_M=((I_err_m * Ikp_m) + I_tmp_m + (d_I_m * Ikd_m));
 }
*/
```

```
void PWM_Generator_m() {
duty_m =  pwm_I_M;
//duty_m = 0.5;
   if (duty_m> 0.0) {
      if (duty_m > 0.95) {
         duty_m = 0.98;
      }
       clk_M=1;
   anticlk_M=0;
      pwm_m = duty_m;
   }

   if (duty_m < 0.0) {
      if (duty_m < -0.95) {
         duty_m = -0.98;
      }
        clk_M=0;
   anticlk_M=1;
      pwm_m = -1.0 * duty_m;
   }
}


void PWM_Generator_s(){

duty_s = PWM_pos;
//duty_s = 0.2;//current pid
  // duty_s = 0.65;
   if (duty_s > 0.0) {
      if (duty_s > 0.98) {
         duty_s = 0.6;
      }
        clk_S=1;
   anticlk_S=0;

      pwm_s = duty_s;
   }

   if (duty_s < 0.0) {
      if (duty_s < -0.98) {
         duty_s = -0.6;
      }
     clk_S=0;
   anticlk_S=1;
```

```
        pwm_s = -1.0*duty_s;
    }

}


 int cleanup_module(void){

    pwm_m = 0.0;         // pwm cleanup module

    pwm_s = 0.0;


   // Reset_AB_M = 0;      //RESET H BRIDGE
    EnableM=0;
    EnableS=0;
    led1=0;
    led3=0;

    return 0;
}

//RTOS

void Control_body() {
     //  M_cur =  current_m.read();                    // Control Part - main code
      velocity_command(); // ---------slave
       velocity_pid_s();// ---------slave
     //  I_com();
   current_pid_s ();
   // current_pid_m ();
   dob_rotary();//----------dob
     position_pid_m();//-----------master
       //position_com_s();
       //position_pid_s();

       PWM_Generator_s();
      PWM_Generator_m();
      // Disob();


        led1=!led1;
         counter++;
   }
```

```
void thread_2(void const *argument){
 //pc.printf("Pulses is: %f\n", M_encorder.read() );
 //  wait(0.2);

/*
        SDFileSystem sd(p5, p6, p7, p8, "sd");
         FILE *fp = fopen("/sd/v100.dat", "w");
 if(fp == NULL) {
         for(int i=0;i<5;i++){
        led3=!led3;
         wait(1.0);
            }
         }*/

 while(1){//counter<10000
   if(counter>=(counter_old+100)){
  //M_position= M_encorder.read();

 // rpm = M_position*400;
     //  fprintf(fp,"%d %f %f %f \n",timer.read_us(),I_act_s,I_ref_s, i_com_s );
         // fprintf(fp,"%d %f \n",timer.read_us(),position);
         //   pc.printf(" %f %f \n",tau_dob,I_act_s );
  //pc.printf("10" );
  pc.printf("%f %f %f %f %f %f  %d\n", vrtclmm,Rtob_S,
v_res_s,tau_dob,I_act_s,I_res_s,timer.read_us());

   // pc.printf(" %f %f %f \n",I_act_s,v_ref_s,v_res_s);
    //  pc.printf("PWM: %f\n", pid_V_I_S );
          // pc.printf("rpm: %f\n", rpm );
       //  pc.printf("rpm actual: %f\n", v_res_s );
     //  pc.printf("I_v_err_s: %f\n", I_v_err_s );
     //  pc.printf("current act: %f\n", I_act_s );
     //pc.printf("current sens %f\n", M_cur );
       //pc.printf("toq S: %f\n", pwm_I_M );
      //pc.printf("currentmn: %f\n", current_mn.read() );
  //wait(0.5);
        counter_old=counter;
       // led3=!led3;
          }
        }


      fclose(fp);
      timer.stop();
      cleanup_module();
```

```
    ticker.detach ();


  }

int main() {
  //pc.baud(115200);
  dt =dt_us/1000000.0;
  pwm_init();
  timer.start();


  Thread thread2(thread_2, NULL, osPriorityNormal, (DEFAULT_STACK_SIZE *
4));

  ticker.attach_us(&Control_body, dt_us);
  // pc.printf("Pulses is: %d\n", position);

  //wait(1.0);

  }
```