```c
#include <tistdtypes.h>
#include <coecsl.h>
#include "user_includes.h"
#include "math.h"

// These two offsets are only used in the main file user_CRSRobot.c  You just need to create them here and find the correct offset and then these offset will adjust the encoder readings
float offset_Enc2_rad = -0.415; //-0.37;
float offset_Enc3_rad = 0.233; //0.27;

// Your global variables.
float px = 0; // x coordinate of end effector
float py = 0; // y coordinate of end effector
float pz = 0; // z coordinate of end effector
float theta1m_IK = 0; // theta1 motor from inverse kinematics
float theta2m_IK = 0; // theta2 motor from inverse kinematics
float theta3m_IK = 0; // theta3 motor from inverse kinematics
long mycount = 0;

// Variables for Omega calculations
float Theta1_old = 0;
float Omega1_old1 = 0;
float Omega1_old2 = 0;
float Omega1 = 0;

float Theta2_old = 0;
float Omega2_old1 = 0;
float Omega2_old2 = 0;
float Omega2 = 0;

float Theta3_old = 0;
float Omega3_old1 = 0;
float Omega3_old2 = 0;
float Omega3 = 0;

//Forward Kinematics
float x, y, z;
// Variables for Velocity calculations
float x_old = 0;
float vx_old1 = 0;
float vx_old2 = 0;
float vx = 0;

float y_old = 0;
float vy_old1 = 0;
float vy_old2 = 0;
float vy = 0;

float z_old = 0;
float vz_old1 = 0;
float vz_old2 = 0;
float vz = 0;

// Constants
float dt = 0.001;
float threshold1 = 0.01;
float threshold2 = 0.06;
float threshold3 = 0.02;
```

```
float theta_dotdot = 0.0;

float x_desired = 0;
float y_desired = 0;
float z_desired = 0;

//Without Mass
float p1 = 0.0300;
float p2 = 0.0128;
float p3 = 0.0076;
float p4 = 0.0753;
float p5 = 0.0298;

float a_theta2;
float a_theta3;

float sintheta2;
float costheta2;
float sintheta3;
float costheta3;
float g = 9.81;

float J1 = 0.0167;
float J2 = 0.03;
float J3 = 0.0128;

//Minimum velocity, positive and negative viscous coefficients, and positive and negative coulomb (static) coefficients for each joint
float Viscouspos1=0.17, Viscouspos2=0.23, Viscouspos3=0.1922;
float Viscousneg1=0.17, Viscousneg2=0.287, Viscousneg3=0.2132;
float Coulombpos1=0.40, Coulombpos2=0.40, Coulombpos3=0.40;
float Coulombneg1=-0.35, Coulombneg2=-0.40, Coulombneg3=-0.50;
float slope1 = 3.6;
float slope2 = 3.6;
float slope3 = 3.6;
float ff = 1;

float a0,a1,a2,a3;
float tau1cur,tau2cur,tau3cur;

float desired_1, desired_2, desired_3;
float desired_dot_1, desired_dot_2, desired_dot_3;
float desired_doubledot_1, desired_doubledot_2, desired_doubledot_3;

//sin and cos of the three joints
float cosq1 = 0;
float sinq1 = 0;
float cosq2 = 0;
float sinq2 = 0;
float cosq3 = 0;
float sinq3 = 0;
//Jacobian Transpose
float JT_11 = 0;
float JT_12 = 0;
float JT_13 = 0;
float JT_21 = 0;
float JT_22 = 0;
float JT_23 = 0;
float JT_31 = 0;
```

```
float JT_32 = 0;
float JT_33 = 0;
//sin and cos used for rotation
float cosz = 0;
float sinz = 0;
float cosx = 0;
float sinx = 0;
float cosy = 0;
float siny = 0;
//Rotational matrix
float R11 = 0;
float R12 = 0;
float R13 = 0;
float R21 = 0;
float R22 = 0;
float R23 = 0;
float R31 = 0;
float R32 = 0;
float R33 = 0;
//Transpose of the rotational matrix
float RT11 = 0;
float RT12 = 0;
float RT13 = 0;
float RT21 = 0;
float RT22 = 0;
float RT23 = 0;
float RT31 = 0;
float RT32 = 0;
float RT33 = 0;

float thetax = 0;
float thetay = 0;
float thetaz = 60;
//Task Space PD Gains
float kpx = 0.5;
float kpy = 0.5;
float kpz = 0.5;
float kdx = 0.025;
float kdy = 0.025;
float kdz = 0.025;

//Gravity Compensation in Z direction
float fzcmd = 0;
float kt = 6;
float grav_comp = 5;

// Gains in N-frame
float Kpx_n = 1.3;
float Kpy_n = 1.3;
float Kpz_n = 1.3;
float Kdx_n = 0.03;
float Kdy_n = 0.03;
float Kdz_n = 0.025;

//Final Project Variables
int mode = 2; //Mode = 1: Impedence Control; Mode = 2: Task Space PD control; Mode = 3: Impedence Control 2

//Desired Trajectory
float xd, yd, zd;
```

```
float xd_dot = 0;
float yd_dot = 0;
float zd_dot = 0;

float a = 0;
float b = 0;
float c = 0;
float d = 0;
float t = 0.0;

float t_total = 0;
float deltax = 0;
float deltay = 0;
float deltaz = 0;

typedef struct point_tag {
    float x; // x position
    float y; // y position
    float z; // z position
    float thz; // rotation about z
    int mode; //Modes for controller
} point;

#define XYZSTIFF 1
#define ZSTIFF 2
#define XZSTIFF 3

#define NUM_POINTS 24
point point_array[NUM_POINTS] = {
        { 6.5, 0, 17, 0, XYZSTIFF},  //0 point 0
        { 10, 0, 20, 0, XYZSTIFF},  //1 point 0 Home Position
        {7.88, 9.03, 18.86, 0, XYZSTIFF},   //2 point 1
        {1.57, 13.98, 8, 0, XYZSTIFF},   //3 point 2
        {1.57, 13.98, 7.1, 0, XYZSTIFF}, //4 point 3   peg hole
        {1.57, 13.98, 5.00, 0, ZSTIFF},  //5 point 4 x and y weak  peg hole
        {1.57, 13.98, 12, 0, ZSTIFF},  //6 point 5 x and y weak   peg hole
        {7.64, 9.64, 12.34, 0, XYZSTIFF},  //7 point 6 peg hole not used
        {8.97, 5.11, 13.64, 0, XYZSTIFF},  //8 point 7  avoid obstacle above egg
        {9, 3.68, 9.7, 0, XYZSTIFF},  //9 point 8 avoid obstacle not used
        {9.14, 0.82, 9.7, 0, XYZSTIFF},  //10 point 9 avoid obstacle not used
        {15.33, 3.90, 8.11, 0, XYZSTIFF},//11 point 10 begin zig zag (Start Point)
        {16.22, 2.50, 8.11, PI/4, XZSTIFF}, //12 point 11  zig zag Before 1st curve
        {15.63, 1.80, 8.11, -PI/4, XZSTIFF},  //13 point 12 zig zag After 1st curve
        {13.04, 2.06, 8.11, PI/4, XZSTIFF},  //14 point 13 zig zag
        {12.70, 1.11, 8.11, 0, XYZSTIFF},  //15 point 14 up
        {15.21, -1.98, 8.11, 0, XYZSTIFF},  //16 point 15 top egg
        {15.21, -1.98, 10.64, 0, XYZSTIFF},  //17 point 16 push egg
        {9.73, 5.52, 12.64, 0, XYZSTIFF},  //18 point 17 push egg
        {9.73, 5.52, 11.11, 0, XYZSTIFF},  //19 point 18 top egg
        {10, 0, 20, 0, XYZSTIFF},  //20 point 19
        { 6.5, 0, 17, 0, XYZSTIFF},  // point 0 not used
        { 6.5, 0, 17, 0, XYZSTIFF},  // point 0 not used
        { 6.5, 0, 17, 0, XYZSTIFF},  // point 0 not used
};


#pragma DATA_SECTION(whattoprint, ".my_vars")
float whattoprint = 0.0;

#pragma DATA_SECTION(whatnottoprint, ".my_vars")
```

```c
float whatnottoprint = 0.0;

#pragma DATA_SECTION(theta1array, ".my_arrs")
float theta1array[100];

#pragma DATA_SECTION(theta2array, ".my_arrs")
float theta2array[100];

long arrayindex = 0;

float printtheta1motor = 0;
float printtheta2motor = 0;
float printtheta3motor = 0;

// Assign these float to the values you would like to plot in Simulink
float Simulink_PlotVar1 = 0;
float Simulink_PlotVar2 = 0;
float Simulink_PlotVar3 = 0;
float Simulink_PlotVar4 = 0;

void inverseKinematics(float px, float py, float pz) {
    // Calculate DH angles from end effector position
    float theta1 = atan2(py, px); // DH theta 1
    float z = pz - 10;
    float beta = sqrt(px*px + py*py);
    float L = sqrt(z*z + beta*beta);
    float theta3 = acos((L*L - 200)/200); // DH theta 3
    float theta2 = -theta3/2 - atan2(z, beta); // DH theta 2


    // Convert DH angles to motor angles
    theta1m_IK = theta1;
    theta2m_IK = (theta2 + PI/2);
    theta3m_IK = (theta3 + theta2m_IK - PI/2);
}

// Omega estimation
void omega(float theta1motor, float theta2motor, float theta3motor) {
    Omega1 = (theta1motor - Theta1_old)/0.001;
    Omega1 = (Omega1 + Omega1_old1 + Omega1_old2)/3.0;

    Theta1_old = theta1motor;

    Omega1_old2 = Omega1_old1;
    Omega1_old1 = Omega1;

    Omega2 = (theta2motor - Theta2_old)/0.001;
    Omega2 = (Omega2 + Omega2_old1 + Omega2_old2)/3.0;

    Theta2_old = theta2motor;

    Omega2_old2 = Omega2_old1;
    Omega2_old1 = Omega2;

    Omega3 = (theta3motor - Theta3_old)/0.001;
    Omega3 = (Omega3 + Omega3_old1 + Omega3_old2)/3.0;

    Theta3_old = theta3motor;

    Omega3_old2 = Omega3_old1;
```

```
    Omega3_old1 = Omega3;
}

//Velocity Updates
void velocity(float x, float y, float z) {
    vx = (x - x_old)/0.001;
    vx = (vx + vx_old1 + vx_old2)/3.0;

    x_old = x;

    vx_old2 = vx_old1;
    vx_old1 = vx;

    vy = (y - y_old)/0.001;
    vy = (vy + vy_old1 + vy_old2)/3.0;

    y_old = y;

    vy_old2 = vy_old1;
    vy_old1 = vy;

    vz = (z - z_old)/0.001;
    vz = (vz + vz_old1 + vz_old2)/3.0;

    z_old = z;

    vz_old2 = vz_old1;
    vz_old1 = vz;
}

// Generation cubic trajectory
//This function is used to calculate the coefficients required to perform a cubic trajectory. The cubic function ranges from times t to t_f, and from positions p_0 to p_1.
//Source: Robot Modeling and Control by Spong, Hutchinson, Vidyasagar. Page 175-176.

float cubic2points(float t, float t_f, float p_0, float p_1){
    a = 2*(p_0 - p_1)/(t_f*t_f*t_f);
    b = -3*(p_0 - p_1)/(t_f*t_f);
    d = p_0;
    return (a*t*t*t + b*t*t + d);
}

void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,float *tau2,float *tau3, int error) {
    //Forward Kinematics
    x = 10*cos(theta1motor)*(cos(theta3motor) + sin(theta2motor));
    y = 10*sin(theta1motor)*(cos(theta3motor) + sin(theta2motor));
    z = 10*cos(theta2motor) - 10*sin(theta3motor) + 10;

    //Coefficients for different time steps
    // trajectory
    t = (mycount%200000)/1000.;

    float t_0 = 0.75; //Homing
    float t_1 = t_0 + 1.0; //Align to the Hole
    float t_2 = t_1 + 1.0; // Going into the Hole
    float t_3 = t_2 + 1.0; //From inside hole to outside hole
    float t_4 = t_3 + 1.0; //Outside hole to above egg
    float t_5 = t_4 + 1.0;//Above egg to entrance of Zigzag
```

```
    float t_6 = t_5 + 0.65; //1st Straight line from point A to B
    float t_7 = t_6 + 0.65; //1st Curve from point A to B
    float t_8 = t_7 + 1.25; //2nd Straight line from point A to B
    float t_9 = t_8 + 0.55; //2nd Curve from point A to B
    float t_10 = t_9 + 0.4; //3rd Straight line from point A to exit
    float t_11 = t_10 + 0.4; //Exit to Up
    float t_12 = t_11 + 0.4; //Up to top of egg
    float t_13 = t_12 + 0.8; //PUSH!!!!!
    float t_14 = t_13 + 2.0; //keep pushing!!!!!
    float t_15 = t_14 + 0.7; //After egg go back to home position

    if (t <= t_0){ // Homing
        xd = cubic2points(t, t_0 , point_array[0].x, point_array[1].x);
        yd = point_array[0].y;
        zd = cubic2points(t, t_0 , point_array[0].z, point_array[1].z);
        mode = 2;
        }
    else if (t_0<t && t<=t_1){ //Aligning to the Hole
        xd = cubic2points(t-t_0, t_1-t_0 , point_array[1].x, point_array[3].x);
        yd = cubic2points(t-t_0, t_1-t_0 , point_array[1].y, point_array[3].y);
        zd = cubic2points(t-t_0, t_1-t_0 , point_array[1].z, point_array[3].z);
        mode = 2;
    }
    else if (t_1<t && t<=t_2/2){ //Going into the Hole
        xd = cubic2points(t-t_1, t_2/2-t_1, point_array[3].x, point_array[5].x);
        yd = cubic2points(t-t_1, t_2/2-t_1 , point_array[3].y, point_array[5].y);
        zd = cubic2points(t-t_1, t_2/2-t_1 , point_array[3].z, point_array[5].z);
        mode = 3;
    }
    else if (t_2/2<t && t<=t_2){ //Staying inside Hole
        xd = cubic2points(t-t_2/2, t_2-t_2/2, point_array[5].x, point_array[5].x);
        yd = cubic2points(t-t_2/2, t_2-t_2/2 , point_array[5].y, point_array[5].y);
        zd = cubic2points(t-t_2/2, t_2-t_2/2 , point_array[5].z, point_array[5].z);
        mode = 3;
    }
    else if (t_2<t && t<=t_3){ //From inside hole to outside hole
        xd = cubic2points(t-t_2, t_3-t_2 , point_array[5].x, point_array[6].x);
        yd = cubic2points(t-t_2, t_3-t_2 , point_array[5].y, point_array[6].y);
        zd = cubic2points(t-t_2, t_3-t_2 , point_array[5].z, point_array[6].z);
        mode = 3;
        }
    else if (t_3<t && t<=t_4){ //Outside hole to above egg
        t_total = t_4 - t_3;
        deltax = point_array[8].x - point_array[6].x;
        deltay = point_array[8].y - point_array[6].y;
        deltaz = point_array[8].z - point_array[6].z;
        xd = point_array[6].x + deltax*(t-t_3)/t_total;
        yd = point_array[6].y + deltay*(t-t_3)/t_total;
        zd = point_array[6].z + deltaz*(t-t_3)/t_total;
        mode = 3;
        }
    else if (t_4<t && t<=t_5){ //Above egg to entrance of Zigzag
        xd = cubic2points(t-t_4, t_5-t_4 , point_array[8].x, point_array[11].x);
        yd = cubic2points(t-t_4, t_5-t_4 , point_array[8].y, point_array[11].y);
        zd = cubic2points(t-t_4, t_5-t_4 , point_array[8].z, point_array[11].z);
        mode = 2;
        }
    else if (t_5<t && t<=t_6){ //1st Straight line from point A to B
        t_total = t_6 - t_5;
        deltax = point_array[12].x - point_array[11].x;
```

```
        deltay = point_array[12].y - point_array[11].y;
        deltaz = point_array[12].z - point_array[11].z;
        xd = point_array[11].x + deltax*(t-t_5)/t_total;
        yd = point_array[11].y + deltay*(t-t_5)/t_total;
        zd = point_array[11].z + deltaz*(t-t_5)/t_total;
        mode = 1;
        }
    else if (t_6<t && t<=t_7){ //1st Curve from point A to B
        xd = cubic2points(t-t_6, t_7-t_6 , point_array[12].x, point_array[13].x);
        yd = cubic2points(t-t_6, t_7-t_6 , point_array[12].y, point_array[13].y);
        zd = cubic2points(t-t_6, t_7-t_6 , point_array[12].z, point_array[13].z);
        mode = 1;
            }
    else if (t_7<t && t<=t_8){ //2nd Straight line from point A to B
        t_total = t_8 - t_7;
        deltax = point_array[14].x - point_array[13].x;
        deltay = point_array[14].y - point_array[13].y;
        deltaz = point_array[14].z - point_array[13].z;
        xd = point_array[13].x + deltax*(t-t_7)/t_total;
        yd = point_array[13].y + deltay*(t-t_7)/t_total;
        zd = point_array[13].z + deltaz*(t-t_7)/t_total;
        mode = 1;
        }
    else if (t_8<t && t<=t_9){ //2nd Curve from point A to B
        xd = cubic2points(t-t_8, t_9-t_8 , point_array[14].x, point_array[15].x);
        yd = cubic2points(t-t_8, t_9-t_8 , point_array[14].y, point_array[15].y);
        zd = cubic2points(t-t_8, t_9-t_8 , point_array[14].z, point_array[15].z);
        mode = 1;
        }
    else if (t_9<t && t<=t_10){ //3rd Straight line from point A to exit
        t_total = t_10 - t_9;
        deltax = point_array[16].x - point_array[15].x;
        deltay = point_array[16].y - point_array[15].y;
        deltaz = point_array[16].z - point_array[15].z;
        xd = point_array[15].x + deltax*(t-t_9)/t_total;
        yd = point_array[15].y + deltay*(t-t_9)/t_total;
        zd = point_array[15].z + deltaz*(t-t_9)/t_total;
        mode = 1;
            }
    else if (t_10<t && t<=t_11){ //Exit to Up
        t_total = t_11 - t_10;
        deltax = point_array[17].x - point_array[16].x;
        deltay = point_array[17].y - point_array[16].y;
        deltaz = point_array[17].z - point_array[16].z;
        xd = point_array[16].x + deltax*(t-t_10)/t_total;
        yd = point_array[16].y + deltay*(t-t_10)/t_total;
        zd = point_array[16].z + deltaz*(t-t_10)/t_total;
        mode = 2;
                }
    else if (t_11<t && t<=t_12){ //Up to top of egg
        xd = cubic2points(t-t_11, t_12-t_11 , point_array[17].x, point_array[18].x);
        yd = cubic2points(t-t_11, t_12-t_11 , point_array[17].y, point_array[18].y);
        zd = cubic2points(t-t_11, t_12-t_11 , point_array[17].z, point_array[18].z);
        mode = 2;
                }
    else if (t_12<t && t<=t_13){ //PUSH!!!!!
        xd = cubic2points(t-t_12, t_13-t_12 , point_array[18].x, point_array[19].x);
        yd = cubic2points(t-t_12, t_13-t_12 , point_array[18].y, point_array[19].y);
        zd = cubic2points(t-t_12, t_13-t_12 , point_array[18].z, point_array[19].z);
        mode = 3;
```

```
                }

    else if (t_13<t && t<=t_14){ //keep pushing!!!!!
        xd = cubic2points(t-t_13, t_14-t_13 , point_array[19].x, point_array[19].x);
        yd = cubic2points(t-t_13, t_14-t_13 , point_array[19].y, point_array[19].y);
        zd = cubic2points(t-t_13, t_14-t_13 , point_array[19].z, point_array[19].z);
        mode = 3;
                }

    else if (t_14<t && t<=t_15){ //After egg go back to home position
        xd = cubic2points(t-t_14, t_15-t_14 , point_array[19].x, point_array[20].x);
        yd = cubic2points(t-t_14, t_15-t_14 , point_array[19].y, point_array[20].y);
        zd = cubic2points(t-t_14, t_15-t_14 , point_array[19].z, point_array[20].z);
        mode = 3;
                }
    else{
        xd = point_array[20].x;
        yd = point_array[20].y;
        zd = point_array[20].z;
    }
    // Calculate/Update omegas/velocities
    omega(theta1motor, theta2motor, theta3motor);
    velocity(x, y, z);

    // position error
    float x_error = xd - x;
    float y_error = yd - y;
    float z_error = zd - z;
    float xd_error = xd_dot - vx;
    float yd_error = yd_dot - vy;
    float zd_error = zd_dot - vz;

    if(mode == 1){  //Zigzag mode
        // Gains in N-frame
        Kpx_n = 1.0;
        Kpy_n = 1.0;
        Kpz_n = 1.3;
        Kdx_n = 0.02;
        Kdy_n = 0.02;
        Kdz_n = 0.025;
        cosq1 = cos(theta1motor);
        sinq1 = sin(theta1motor);
        cosq2 = cos(theta2motor);
        sinq2 = sin(theta2motor);
        cosq3 = cos(theta3motor);
        sinq3 = sin(theta3motor);
        JT_11 = -10*sinq1*(cosq3 + sinq2);
        JT_12 = 10*cosq1*(cosq3 + sinq2);
        JT_13 = 0;
        JT_21 = 10*cosq1*(cosq2 - sinq3);
        JT_22 = 10*sinq1*(cosq2 - sinq3);
        JT_23 = -10*(cosq3 + sinq2);
        JT_31 = -10*cosq1*sinq3;
        JT_32 = -10*sinq1*sinq3;
        JT_33 = -10*cosq3;

        cosz = cos(thetaz);
        sinz = sin(thetaz);
        cosx = cos(thetax);
        sinx = sin(thetax);
```

```
        cosy = cos(thetay);
        siny = sin(thetay);

        RT11 = R11 = cosz*cosy-sinz*sinx*siny;
        RT21 = R12 = -sinz*cosx;
        RT31 = R13 = cosz*siny+sinz*sinx*cosy;
        RT12 = R21 = sinz*cosy+cosz*sinx*siny;
        RT22 = R22 = cosz*cosx;
        RT32 = R23 = sinz*siny-cosz*sinx*cosy;
        RT13 = R31 = -cosx*siny;
        RT23 = R32 = sinx;
        RT33 = R33 = cosx*cosy;

        *tau1 = (JT_11*R11 + JT_12*R21 + JT_13*R31)*(Kdx_n*R11*xd_error + Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error + Kpx_n*R31*z_error) + (JT_11*R12 + JT_12*R22 + JT_13*R32)*(Kdy_n*R12*xd_error + Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error + Kpy_n*R32*z_error) + (JT_11*R13 + JT_12*R23 + JT_13*R33)*(Kdz_n*R13*xd_error + Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error + Kpz_n*R33*z_error);
        *tau2 = (JT_21*R11 + JT_22*R21 + JT_23*R31)*(Kdx_n*R11*xd_error + Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error + Kpx_n*R31*z_error) + (JT_21*R12 + JT_22*R22 + JT_23*R32)*(Kdy_n*R12*xd_error + Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error + Kpy_n*R32*z_error) + (JT_21*R13 + JT_22*R23 + JT_23*R33)*(Kdz_n*R13*xd_error + Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error + Kpz_n*R33*z_error);
        *tau3 = (JT_31*R11 + JT_32*R21 + JT_33*R31)*(Kdx_n*R11*xd_error + Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error + Kpx_n*R31*z_error) + (JT_31*R12 + JT_32*R22 + JT_33*R32)*(Kdy_n*R12*xd_error + Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error + Kpy_n*R32*z_error) + (JT_31*R13 + JT_32*R23 + JT_33*R33)*(Kdz_n*R13*xd_error + Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error + Kpz_n*R33*z_error);
          }
    else if(mode == 2){//Task Space PD controller

        float fx = kpx*(xd - x) + kdx*(xd_dot - vx);
        float fy = kpy*(yd - y) + kdy*(yd_dot - vy);
        float fz = kpz*(zd - z) + kdz*(zd_dot - vz);

        //Compute Jacobian Transpose
        // Jacobian Transpose
        cosq1 = cos(theta1motor);
        sinq1 = sin(theta1motor);
        cosq2 = cos(theta2motor);
        sinq2 = sin(theta2motor);
        cosq3 = cos(theta3motor);
        sinq3 = sin(theta3motor);
        JT_11 = -10*sinq1*(cosq3 + sinq2);
        JT_12 = 10*cosq1*(cosq3 + sinq2);
        JT_13 = 0;
        JT_21 = 10*cosq1*(cosq2 - sinq3);
        JT_22 = 10*sinq1*(cosq2 - sinq3);
        JT_23 = -10*(cosq3 + sinq2);
        JT_31 = -10*cosq1*sinq3;
        JT_32 = -10*sinq1*sinq3;
        JT_33 = -10*cosq3;
        //
        float x_grav_comp = 0;
```

```
        float y_grav_comp = 0.0254*JT_23*fzcmd/kt;
        float z_grav_comp = 0.0254*JT_33*(fzcmd + grav_comp)/kt;
        *tau1 = JT_11*fx + JT_12*fy + x_grav_comp;
        *tau2 = JT_21*fx + JT_22*fy + JT_23*fz + y_grav_comp;
        *tau3 = JT_31*fx + JT_32*fy + JT_33*fz + z_grav_comp;

    }

    else if(mode == 3){//Simple Impedance Control
        Kpx_n = 1.0;
        Kpy_n = 1.0;
        Kpz_n = 1.3;
        Kdx_n = 0.03;
        Kdy_n = 0.03;
        Kdz_n = 0.025;
        //Compute Jacobian Transpose
        // Jacobian Transpose
        cosq1 = cos(theta1motor);
        sinq1 = sin(theta1motor);
        cosq2 = cos(theta2motor);
        sinq2 = sin(theta2motor);
        cosq3 = cos(theta3motor);
        sinq3 = sin(theta3motor);
        JT_11 = -10*sinq1*(cosq3 + sinq2);
        JT_12 = 10*cosq1*(cosq3 + sinq2);
        JT_13 = 0;
        JT_21 = 10*cosq1*(cosq2 - sinq3);
        JT_22 = 10*sinq1*(cosq2 - sinq3);
        JT_23 = -10*(cosq3 + sinq2);
        JT_31 = -10*cosq1*sinq3;
        JT_32 = -10*sinq1*sinq3;
        JT_33 = -10*cosq3;

        cosz = cos(thetaz);
        sinz = sin(thetaz);
        cosx = cos(thetax);
        sinx = sin(thetax);
        cosy = cos(thetay);
        siny = sin(thetay);

        RT11 = R11 = cosz*cosy-sinz*sinx*siny;
        RT21 = R12 = -sinz*cosx;
        RT31 = R13 = cosz*siny+sinz*sinx*cosy;
        RT12 = R21 = sinz*cosy+cosz*sinx*siny;
        RT22 = R22 = cosz*cosx;
        RT32 = R23 = sinz*siny-cosz*sinx*cosy;
        RT13 = R31 = -cosx*siny;
        RT23 = R32 = sinx;
        RT33 = R33 = cosx*cosy;

        *tau1 = (JT_11*R11 + JT_12*R21 + JT_13*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_11*R12 + JT_12*R22 + JT_13*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_11*R13 + JT_12*R23 + JT_13*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
        *tau2 = (JT_21*R11 + JT_22*R21 + JT_23*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_21*R12 + JT_22*R22 + JT_23*R32)*(Kdy_n*R12*xd_error +
```

```
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_21*R13 + JT_22*R23 + JT_23*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
        *tau3 = (JT_31*R11 + JT_32*R21 + JT_33*R31)*(Kdx_n*R11*xd_error +
Kpx_n*R11*x_error + Kdx_n*R21*yd_error + Kpx_n*R21*y_error + Kdx_n*R31*zd_error +
Kpx_n*R31*z_error) + (JT_31*R12 + JT_32*R22 + JT_33*R32)*(Kdy_n*R12*xd_error +
Kpy_n*R12*x_error + Kdy_n*R22*yd_error + Kpy_n*R22*y_error + Kdy_n*R32*zd_error +
Kpy_n*R32*z_error) + (JT_31*R13 + JT_32*R23 + JT_33*R33)*(Kdz_n*R13*xd_error +
Kpz_n*R13*x_error + Kdz_n*R23*yd_error + Kpz_n*R23*y_error + Kdz_n*R33*zd_error +
Kpz_n*R33*z_error);
    }//


     //  Friction Compensation
    //If omega greater than 0.1 (Max Velocity) for joint 1
    if (Omega1 > 0.1) {
        *tau1 = *tau1 + 0.6*(Viscouspos1 * Omega1 + Coulombpos1) ;
        }
    //If omega greater than -0.1 (Min Velocity) for joint 1
    else if (Omega1 <-0.1) {
        *tau1 = *tau1 +0.6*(Viscousneg1 * Omega1 + Coulombneg1);
    }
    //If omega between min and max velocity for joint 1
    else {
        *tau1 = *tau1 + 0.6*(slope1*Omega1);
    }

    //If omega greater than 0.05(Max Velocity) for joint 2
    if (Omega2 > 0.05) {
        *tau2 = *tau2 + 0.6*(Viscouspos2 * Omega2 + Coulombpos2);
    }
    //If omega greater than -0.05 (Min Velocity) for joint 2
    else if (Omega2 <-0.05) {
        *tau2 = *tau2 + 0.6*(Viscousneg2 * Omega2 + Coulombneg2);
    }
    //If omega between min and max velocity for joint 2
    else {
        *tau2 = *tau2 + 0.6*(slope2*Omega2);
    }
    //If omega greater than -0.05 (Max Velocity) for joint 3
    if (Omega3 > 0.05) {
        *tau2 = *tau2 + 0.6*(Viscouspos3 * Omega3 + Coulombpos3) ;
    }
    //If omega greater than -0.05 (Min Velocity) for joint 3
    else if (Omega3 <-0.05) {
        *tau2 = *tau2 + 0.6*(Viscousneg3 * Omega3 + Coulombneg3);
    }
    //If omega between min and max velocity for joint 3
    else {
        *tau2 = *tau2 + 0.6*(slope3*Omega3);
    }


    //Limits torque to 5N-m
    if (*tau1 >= 5) {
        *tau1 = 5;
    } else if (*tau1 < -5) {
        *tau1 = -5;
    }
```

```

    if (*tau2 >= 5) {
        *tau2 = 5;
    } else if (*tau2 < -5) {
        *tau2 = -5;
    }

    if (*tau3 >= 5) {
        *tau3 = 5;
    } else if (*tau3 < -5) {
        *tau3 = -5;
    }


    // save past states
    if ((mycount%50)==0) {

        theta1array[arrayindex] = theta1motor;
        theta2array[arrayindex] = theta2motor;

        if (arrayindex >= 100) {
            arrayindex = 0;
        } else {
            arrayindex++;
        }

    }

    if ((mycount%500)==0) {
        if (whattoprint > 0.5) {
            serial_printf(&SerialA, "I love robotics\n\r");
        } else {
            printtheta1motor = theta1motor;
            printtheta2motor = theta2motor;
            printtheta3motor = theta3motor;
            SWI_post(&SWI_printf); //Using a SWI to fix SPI issue from sending too many floats.
        }

        GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Blink LED on Control Card
        GpioDataRegs.GPBTOGGLE.bit.GPIO60 = 1; // Blink LED on Emergency Stop Box
    }



    Simulink_PlotVar1 = desired_1;  //yellow
    Simulink_PlotVar2 = theta1motor; //blue
    Simulink_PlotVar3 = theta2motor; //orange
    Simulink_PlotVar4 = theta3motor; //green
    mycount++;
}

void printing(void){
    // Printing (theta1, theta2, theta3, , py, pz) and then on a new line (theta1_IK, theta2_IK, theta3_IK)
    serial_printf(&SerialA, "px: %.2f, py: %.2f, pz: %.2f \n\r", x, y, z);
}
```