ME446 Lab 4: Task Space PD control, Impedance Control

Kevin Gim and Jasvir Virdi

April 26, 2019

1 Source Code Submission

```
// ME446 Lab 4 C Code
// Kevin Gim and Jasvir Virdi


#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 = -0.36;
float offset_Enc3_rad = 0.27;

// Your global varialbes.
long mycount = 0;
#pragma DATA_SECTION(whattoprint, ".my_vars")
float whattoprint = 0.0;
#pragma DATA_SECTION(theta1array, ".my_arrs")
float theta1array[100];
long arrayindex = 0;
float printtheta1motor = 0;
float printtheta2motor = 0;
float printtheta3motor = 0;

float DHtheta1 = 0;
float DHtheta2 = 0;
float DHtheta3 = 0;

float x = 0;
float y = 0;
float z = 0;

float IKtheta1DH = 0;
float IKtheta2DH = 0;
float IKtheta3DH = 0;
float IKthetam1 = 0;
float IKthetam2 = 0;
float IKthetam3 = 0;

// Inverse Kinematics Parameters
```

```
float r1 = 0;
float r2 = 0;
float l1 = 10;
float l2 = 10;
float l3 = 10;

// Feedforward value
float J1 = 0.0167;
float J2 = 0.03;
float J3 = 0.0128;

// Controller Mode: 0 = Impedence Control , 1 = PD Control
int mode = 0;

// Joint Angle Error
float error_1 = 0.0;
float error_2 = 0.0;
float error_3 = 0.0;


// PD Controller Gain
float KP_1 = 150;
float KP_2 = 500;
float KP_3 = 500;
float KD_1 = 0.7;
float KD_2 = 2.0;
float KD_3 = 1.3;

// 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;

// Velocity Variables
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;
float error1_old = 0;

// Time derivative of the position error
float derror1_old1 = 0;
float derror1_old2 = 0;
float derror1 = 0;
float error2_old = 0;
float derror2_old1 = 0;
float derror2_old2 = 0;
float derror2 = 0;
```

```
float error3_old = 0;
float derror3_old1 = 0;
float derror3_old2 = 0;
float derror3 = 0;

float theta1_des = 0;
float theta2_des = 0;
float theta3_des = 0;
float Omega1_des = 0;
float Omega2_des = 0;
float Omega3_des = 0;
float Omega1d_des = 0;
float Omega2d_des = 0;
float Omega3d_des = 0;

float error1_ee = 0.0;
float error2_ee = 0.0;
float error3_ee = 0.0;

float a = 0;
float b = 0;
float c = 0;
float d = 0;
float t = 0.0;
float x_f = 0.0;
float y_f = 0.0;
float z_f = 0.0;
float x_0 = 6.00;
float y_0 = 0;
float z_0 = 16.78;

// For friction compensation

float u_fric_1 = 0;
float u_fric_2 = 0;
float u_fric_3 = 0;
float u_fric = 0;

float min_vel_1 = 0.09;
float vis_pos_1 = 0.26;
float vis_neg_1 = 0.25;
float cmb_pos_1 = 0.4;
float cmb_neg_1 = -.38;

float min_vel_2 = 0.049;
float vis_pos_2 = 0.22;
float vis_neg_2 = 0.275;
float cmb_pos_2 = 0.43;
float cmb_neg_2 = -.43;
float min_vel_3 = 0.049;
float vis_pos_3 = 0.29;
float vis_neg_3 = 0.29;
float cmb_pos_3 = 0.4;
float cmb_neg_3 = -.4;
float slope_1 = 3.6;
float slope_2 = 3.6;
float slope_3 = 3.6;
```

```
float cosq1 = 0;
float sinq1 = 0;
float cosq2 = 0;
float sinq2 = 0;
float cosq3 = 0;
float sinq3 = 0;
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;
float cosz = 0;
float sinz = 0;
float cosx = 0;
float sinx = 0;
float cosy = 0;
float siny = 0;
float thetaz = 0;
float thetax = 0;
float thetay = 0;
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;
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;


//// Gain of Impedence Control
float KP_x = 0.1;
float KP_y = 0.1;
float KP_z = 0.1;
float KD_x = 0.025;
float KD_y = 0.025;
float KD_z = 0.025;

float x_des = 0;
float y_des = 0;
float z_des = 0;

```

```
// Parameters for Trajectory Generation
float t_f;
float p_0;
float p_1;

// Friction Compensation
float fric_comp(float Omega, float min_vel, float vis_pos, float cmb_pos, float vis_neg, float cmb_neg, float slope){
    if (Omega > min_vel) {
        u_fric = vis_pos*Omega + cmb_pos ;
    } else if (Omega < -min_vel) {
        u_fric = vis_neg*Omega + cmb_neg;
    } else {
        u_fric = slope*Omega;
    }
    return u_fric;
}

// Generation cubic trajectory between two points
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);
}


// Velocity Calculation

float velcalc(float thetamotor, float Theta_old, float Omega, float Omega_old1, float Omega_old2){
    Omega = (thetamotor - Theta_old)/0.001;
    Omega = (Omega + Omega_old1 + Omega_old2)/3.0;
    Theta_old = thetamotor;
    Omega_old2 = Omega_old1;
    Omega_old1 = Omega;
    return Omega;
}


// This function is called every 1 ms
void lab(float theta1motor,float theta2motor,float theta3motor,float *tau1, float *tau2,float *tau3, int error) {
    // Rotation zxy and its Transpose
    // Define cos and sin function to save computation resource
    cosz = cos(thetaz);
    sinz = sin(thetaz);
    cosx = cos(thetax);
    sinx = sin(thetax);
    cosy = cos(thetay);
    siny = sin(thetay);
    cosq1 = cos(theta1motor);
    sinq1 = sin(theta1motor);
    cosq2 = cos(theta2motor);
    sinq2 = sin(theta2motor);
    cosq3 = cos(theta3motor);
    sinq3 = sin(theta3motor);
```

```
    // Rotation Matrix
    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;

    // Jacobian Transpose
    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;

    // save past states
    if ((mycount%50)==0) {
        theta1array[arrayindex] = theta1motor;
        if (arrayindex >= 100) {
            arrayindex = 0;
        } else {
            arrayindex++;
        }
    }

    if ((mycount%50)==0) {
        if (whattoprint > 0.5) {
            serial_printf(&SerialA, "I love robotics\n\r");
        } else {
            printtheta1motor = theta1motor;
            printtheta2motor = theta2motor;
            printtheta3motor = theta3motor;
            DHtheta1 = theta1motor;
            DHtheta2 = theta2motor-PI*0.5;
            DHtheta3 = theta3motor-theta2motor+PI*0.5;
            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
    }


    //Forward Kinematics
    x_f = 10*cosq1*(cosq3+sinq2);
    y_f = 10*sinq1*(cosq3+sinq2);
    z_f = 10*(1+cosq2-sinq3);

```

```
    //Inverse Kinematics
    x = x_des;
    y = y_des;
    z = z_des;
    r1 = z-l1;
    r2 = sqrt(r1*r1 + x*x + y*y);
    IKtheta1DH = atan2(y,x);
    IKtheta3DH = PI - acos((l2*l2+l3*l3-r2*r2)/(2*l2*l3));
    IKtheta2DH = -(IKtheta3DH)/2 - asin((r1/r2));
    IKthetam1 = IKtheta1DH;
    IKthetam2 = IKtheta2DH +(PI/2);
    IKthetam3 = IKtheta3DH + IKtheta2DH;
    theta1_des = IKthetam1;
    theta2_des = IKthetam2;
    theta3_des = IKthetam3;

    // Simulink Plot
    Simulink_PlotVar1 = theta3_des;
    Simulink_PlotVar2 = theta3motor;
    Simulink_PlotVar3 = theta2_des;
    Simulink_PlotVar4 = theta2motor;

    // Motor Angle Time Derivative
    Omega1 = velcalc(theta1motor, Theta1_old, Omega1, Omega1_old1, Omega1_old2);
    Omega2 = velcalc(theta2motor, Theta2_old, Omega2, Omega2_old1, Omega2_old2);
    Omega3 = velcalc(theta3motor, Theta3_old, Omega3, Omega3_old1, Omega3_old2);

    // Motor Angle Error
    error_1 = theta1_des - theta1motor;
    error_2 = theta2_des - theta2motor;
    error_3 = theta3_des - theta3motor;

    // End effector Position Error
    error1_ee = x_des - x_f;
    error2_ee = y_des - y_f;
    error3_ee = z_des - z_f;

    // End Effector Position Error Time Derivative
    derror1 = velcalc(error1_ee, error1_old, derror1, derror1_old1,
        derror1_old2);
    derror2 = velcalc(error2_ee, error2_old, derror2, derror2_old1,
        derror2_old2);
    derror3 = velcalc(error3_ee, error3_old, derror3, derror3_old1,
        derror3_old2);

    //Friction Compensation
    u_fric_1 = fric_comp(Omega1, min_vel_1, vis_pos_1, cmb_pos_1, vis_neg_1,
        cmb_neg_1, slope_1);
    u_fric_2 = fric_comp(Omega2, min_vel_2, vis_pos_2, cmb_pos_2, vis_neg_2,
        cmb_neg_2, slope_2);
    u_fric_3 = fric_comp(Omega3, min_vel_3, vis_pos_3, cmb_pos_3, vis_neg_3,
        cmb_neg_3, slope_3);
```

```
// Part 1: Task space PD controller controlling at one point in space
    // Desired EE Position
        x_des = 10;
        y_des = 0;
        z_des = 20;

// Part 2: Impedance Control
    // (a) Weak in one World Coordinate Axis

        // Weak in only Z axis
        KP_x = 0.1;
        KP_y = 0.1;
        KP_z = 0.0001;
        KD_x = 0.025;
        KD_y = 0.025;
        KD_z = 0.001;

    // (b) Weak in one World Coordinate Axis

        // Weak in Y and Z axis
        KP_x = 0.1;
        KP_y = 0.0001;
        KP_z = 0.0001;
        KD_x = 0.025;
        KD_y = 0.001;
        KD_z = 0.001;

    // (c) Weak in one World Coordinate Axis

        // Rotation Angle of the Impedance Frame
        thetax = 0.5;
        thetay = 0.3;

        // Weak in Y and Z axis
        KP_x = 0.1;
        KP_y = 0.0001;
        KP_z = 0.0001;
        KD_x = 0.025;
        KD_y = 0.001;
        KD_z = 0.001;


// Part 3: Following a Straight Line from One point to a Second Point

    // Straight Line Trajectory from one point to a second point
    t = (mycount%200000)/1000.; // Timer
    float t_0 = 1; // Homing
    float t_1 = t_0 + 4; // Following a straight line

    if (t < t_0){ // Homming
        x_des = cubic2points(t, t_0 , x_0, 10);
        y_des = cubic2points(t, t_0 , y_0, -10);
        z_des = cubic2points(t, t_0 , z_0, 10);
        mode = 0;
    }
    else if (t_0<t && t<t_1){ // Move to y direction with 20 distance
        x_des = 10;
```

```
        y_des = cubic2points(t, t_1-t_0 , y_0, 10);
        z_des = 10;
        mode = 0;
    }

    // (a) All axis Stiff, no rotation
        KP_x = 0.1;
        KP_y = 0.1;
        KP_z = 0.1;
        KD_x = 0.025;
        KD_y = 0.025;
        KD_z = 0.025;

    // (b) Second make the direction perpendicular to the line weak and the direction along the line stiff.
    //     To make things a bit easier here, keep the line in a plane parallel to the World XY plane.

        // Rotation Angle of the Impedance Frame
        thetaz = atan(0.5); // x travel: 10, y travel; 20

        // Weak in x axis
        KP_x = 0.0001;
        KP_y = 0.1;
        KP_z = 0.1;
        KD_x = 0.001;
        KD_y = 0.025;
        KD_z = 0.025;



// Impedance Controller
    *tau1 = (JT_11*R11 + JT_12*R21 + JT_13*R31)*(KD_x*R11*derror1 + KD_x*R21*derror2 + KD_x*R31*derror3 + KP_x*R11*error1_ee + KP_x*R21*error2_ee + KP_x*R31*error3_ee)
        + (JT_11*R12 + JT_12*R22 + JT_13*R32)*(KD_y*R12*derror1 + KD_y*R22*derror2 + KD_y*R32*derror3 + KP_y*R12*error1_ee + KP_y*R22*error2_ee + KP_y*R32*error3_ee)
        + (JT_11*R13 + JT_12*R23 + JT_13*R33)*(KD_z*R13*derror1 + KD_z*R23*derror2 + KD_z*R33*derror3 + KP_z*R13*error1_ee + KP_z*R23*error2_ee + KP_z*R33*error3_ee)
        + 0.9*u_fric_1;
    *tau2 = (JT_21*R11 + JT_22*R21 + JT_23*R31)*(KD_x*R11*derror1 + KD_x*R21*derror2 + KD_x*R31*derror3 + KP_x*R11*error1_ee + KP_x*R21*error2_ee + KP_x*R31*error3_ee)
        + (JT_21*R12 + JT_22*R22 + JT_23*R32)*(KD_y*R12*derror1 + KD_y*R22*derror2 + KD_y*R32*derror3 + KP_y*R12*error1_ee + KP_y*R22*error2_ee + KP_y*R32*error3_ee)
        + (JT_21*R13 + JT_22*R23 + JT_23*R33)*(KD_z*R13*derror1 + KD_z*R23*derror2 + KD_z*R33*derror3 + KP_z*R13*error1_ee + KP_z*R23*error2_ee + KP_z*R33*error3_ee)
        + 0.9*u_fric_2;
    *tau3 = (JT_31*R11 + JT_32*R21 + JT_33*R31)*(KD_x*R11*derror1 + KD_x*R21*derror2 + KD_x*R31*derror3 + KP_x*R11*error1_ee + KP_x*R21*error2_ee + KP_x*R31*error3_ee)
        + (JT_31*R12 + JT_32*R22 + JT_33*R32)*(KD_y*R12*derror1 + KD_y*R22*derror2 + KD_y*R32*derror3 + KP_y*R12*error1_ee + KP_y*R22*error2_ee
```

```
            + KP_y*R32*error3_ee)
        + (JT_31*R13 + JT_32*R23 + JT_33*R33)*(KD_z*R13*derror1 + KD_z*R23*
            derror2 + KD_z*R33*derror3 + KP_z*R13*error1_ee + KP_z*R23*error2_ee
            + KP_z*R33*error3_ee)
        + 0.9*u_fric_3;

    mycount++;

}
void printing(void){
    serial_printf(&SerialA, "x: %.2f    y: %.2f    z: %.2f     ", x_f, y_f, z_f
       ); // Display Forward Kinematics Result
    serial_printf(&SerialA, "x_d: %.2f  y_d: %.2f  z_d: %.2f  t: %.2f\n\r",
       x_des, y_des, z_des,t); // Display Forward Kinematics Result
}

}
```