

ECE 763

Project #3

The purpose of this project is to use `RobotBuilder` to simulate the dynamics of DC motor control for the Mitsubishi RM-501 as it draws a circle on a horizontal surface (see Project #2). In particular, you will implement PD control for the motors of the Mitsubishi so that it closely follows the desired circular trajectory that you generate. A C++ program called `Control` has been written to work with `RobotBuilder` to control the Mitsubishi. You will be writing the code for `StudentCode3.cpp`, which is linked to `Control`, to (1) generate the desired circular trajectory (as in Project #2), (2) implement PD control for the five DC motors, and (3) compare the desired and actual 3D position of the gripper as it draws the circle.

Write the program code and generate the results, using the following steps:

1. Open the `Control` project, from the `\RobotBuilder\Projects\Mitsubishi RM-501 \Control3` directory, with Microsoft Visual Studio so that you can edit `StudentCode3.cpp`. Change and add to the code to:

- (a) Generate the ${}^S T_T$ homogeneous transform for the circular trajectory as a function of time (Project #2),
- (b) Use `InverseKinematics` to generate the corresponding desired set of joint angles (in pulse units) along the circular trajectory, and
- (c) Implement PD control for each of the five motors:

$$V_i = K_1(a_{di} - a_i) - K_2\dot{a}_i \quad i = 0, \dots, 4$$

where V_i is the input voltage for motor i , K_1 is the proportional gain, a_{di} is the desired joint angle (pulse units), a_i is the actual joint angle (pulse units), K_2 is the derivative gain, and \dot{a}_i is the actual joint velocity (pulse units / second).

- (d) Compute the actual gripper position from the actual joint positions which are furnished (in pulse units). (See Project #1.) Pass back the actual gripper position and desired position so that `Control` can output your results.

2. For this project, assume that each of the motors of the Mitsubishi has parameters close to that of the U12M4T so that its values are used in the simulation. Further, assume that the mass and inertia of the links of the Mitsubishi present negligible loading to the motors. Therefore, the gains of the PD controllers may be set using the U12M4T characteristics after appropriate transformation between pulse units and radians. What values of the gains, K_1 (volts/pulse unit) and K_2 (volts/(pulse units/second)), are needed for critical damping with a double pole at -60 ? Hint: each individual wrist motor should use a factor of $\left(\frac{3}{40} \frac{\pi}{180}\right)$, in the PD control equation, to convert from pulse units to radians.

3. Build Control. It will generate the Control3.dll file in the \RobotBuilder\Projects\Mitsubishi RM-501 directory. Start RobotBuilder.exe and open mitsu3rbm.cfg (from the \RobotBuilder\Projects\Mitsubishi RM-501 directory).

Under the Control menu in RobotBuilder, click Select Control DLL ... and choose Control3.dll. Under the CFG File menu, select Edit Simulation Properties ... so that you can change the parameters of the simulation. Note that these selections can only be accomplished when you are in the Build mode of RobotBuilder.

Set the Control Step Size to 0.001 so that new points in the trajectory and the control voltages will be computed every 1 ms. Set the Display Update Period to 5 to graphically display a new point every 5 ms. Select Runge-Kutta (4th Order) integration to simulate the dynamics of the Mitsubishi. Use an appropriate value for the integration Step Size.

4. Hit the Simulate and Play buttons to obtain your results. Note that RobotBuilder will output the time, desired gripper position, and actual position to the MitsuProject3.txt file, so that you can check your results.

Your report may be relatively brief and should include the following items: a printout of StudentCode3.cpp, with your names edited into the comments of the file, and a printout of the results in the MitsuProject3.txt file. Also include a short derivation to determine the gains for critical damping on the motors. Use MatLab or Excel to plot the desired and actual gripper positions over the time interval for which results are available, and include these in your report. Any significant points should also be discussed.

```

#include "stdafx.h"
#include "Control.h"

/**
 *   Student Name #1
 *   Student Name #2
 */

/**
 *
 * Routine to (1) compute the joint angles to generate a desired circular motion
 * for the Mitsubishi RM-501, (2) implement PD control for the DC motors of
 * the Mitsubishi to follow the trajectory, and (3) output the desired and
 * actual positions of the gripper.
 *
 * Parameters:
 * SimulationTimeInSeconds - total simulation time in seconds.
 * ActualJointAnglesInPulseUnits - an array of 5 ints that is an input
 * containing the actual positions of the joints in pulse units.
 * ActualJointVelocitiesInPulseUnitsPerSecond - an array of 5 ints that is
 * an input containing the actual angular velocities of the joints in
 * pulse units per second.
 * InputVoltage - an array of 5 floats that is an output and will pass back
 * the DC motor voltage to achieve the desired joint angles.
 * DesiredJointAnglesInPulseUnits - an array of 5 ints that is the output
 * and will pass back the desired joint values (in pulse units).
 * DesiredGripperPosition - an array of 3 floats that is an output and will
 * pass back the desired position of the gripper.
 * ActualGripperPosition - an array of 3 floats that is an output and will
 * pass back the actual position of the gripper.
 */
void ComputeInputVoltage (double SimulationTimeInSeconds,
                          const int ActualJointAnglesInPulseUnits[],
                          const int ActualJointVelocitiesInPulseUnitsPerSecond[],
                          float InputVoltage[], int DesiredJointAnglesInPulseUnits[],
                          float DesiredGripperPosition[], float ActualGripperPosition[])
{
    float DesiredTransform[4][4];

    // Setup the initial desired transform (the x and y position components
    // will be adjusted later).

```

```

DesiredTransform[0][0] = -1;
DesiredTransform[0][1] = 0;
DesiredTransform[0][2] = 0;
DesiredTransform[0][3] = 275;

DesiredTransform[1][0] = 0;
DesiredTransform[1][1] = 1;
DesiredTransform[1][2] = 0;
DesiredTransform[1][3] = 0;

DesiredTransform[2][0] = 0;
DesiredTransform[2][1] = 0;
DesiredTransform[2][2] = -1;
DesiredTransform[2][3] = 54;

DesiredTransform[3][0] = 0;
DesiredTransform[3][1] = 0;
DesiredTransform[3][2] = 0;
DesiredTransform[3][3] = 1;

// Get the joint angles for that transform
InverseKinematics (DesiredTransform, DesiredJointAnglesInPulseUnits);

// Return the gripper position
DesiredGripperPosition[0] = 0;
DesiredGripperPosition[1] = 0;
DesiredGripperPosition[2] = 0;

int i;
// Compute the DC motor voltages to implement PD control
for (i = 0; i < 5; i++)
{
    InputVoltage[i] = 0;
}

// Find the actual gripper position
ActualGripperPosition[0] = 0;
ActualGripperPosition[1] = 0;
ActualGripperPosition[2] = 0;
}

```