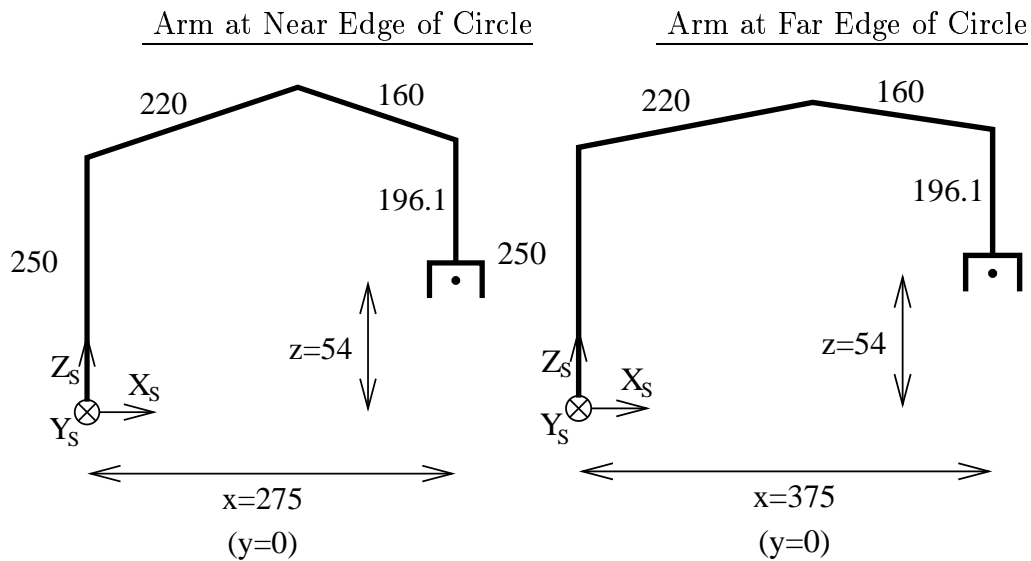


ECE 763

Project #2

The purpose of this project is to use RobotBuilder to produce a graphical simulation of the Mitsubishi RM-501 drawing a circle on a horizontal surface. In particular, you are to write the C code to plan the specified motion and call a procedure for inverse kinematics (`InverseKinematics`) so that the corresponding set of joint angles may be determined. A C++ program called `Control` has been written to work with `RobotBuilder` to simulate the motion specified. You will be writing the code for `StudentCode2.cpp`, which is linked to `Control`, to generate the joint angles for the circular motion.

Let the center of the circle be at (325,0,54) (in millimeters) in inertial reference coordinates. Let the radius for the circle be 50 mm.



Let the gripper start on the near edge of the circle, traverse the circle clockwise at 100 mm/sec (1/4 of the rated speed) while holding its initial orientation, and then continue tracing the circle by repeating the motion.

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

1. Download `RobotBuilder.NET.zip` as needed to a PC equipped with Microsoft Visual Studio .NET. Extract the files while maintaining the full path for each file. A new directory called `\RobotBuilder` will be created with all of the files for the package in it.

2. Start `RobotBuilder.exe`, open `mitsu2.cfg` (from the `\RobotBuilder\Projects\Mitsubishi RM-501` directory), and hit the `Simulate` and `Play` buttons. When functioning properly, the Mitsubishi RM-501 will draw a circle on a horizontal surface while outputting the position data for the motion in the left window for the first π seconds of the motion.
3. Under the `CFG File` menu, select `Edit Simulation Properties ...` so that you can change the parameters of the simulation. Note that this can only be accomplished when you are in the `Build` mode of `RobotBuilder`. Set the `Control Step Size` to 0.01 so that new points in the trajectory will be computed every 10 ms. Set the `Display Update Period` to 5 to graphically display a new point every 50 ms. Note that the `Placement` mode of the "integrator" is selected so that the Mitsubishi is placed at each point computed along the trajectory. Also, be sure the `Slow simulation to real-time` box is checked so that the motion will be generated at real-time rates.
4. Open `Control.sln`, from the `\RobotBuilder\Projects\Mitsubishi RM-501\Control2` directory, with Visual Studio so that you can edit `StudentCode2.cpp`. Change and add to the code to generate the ${}^S T_T$ homogeneous transform and corresponding joint angles along the circular trajectory. Also, be sure to output the position of the gripper over time.
5. Build `Control`. It will generate the `Control2.dll` file in the `\RobotBuilder\Projects\Mitsubishi RM-501` directory. Under the `Control` menu in `RobotBuilder`, click `Select Control DLL ...` and choose `Control2.dll`. (Note that a new `control.dll` can only be selected when you are in the `Build` mode of `RobotBuilder`.) Simulate the Mitsubishi again to obtain your results. Note that `Control` will output the time, gripper position, and joint angles for the first π seconds of the simulation, to the `MitsuProject2.txt` file, so that you can check your results.

Your report may be relatively brief and should include the following items: a printout of `StudentCode2.cpp` and a printout of the results in the `MitsuProject2.txt` file. Also include a short derivation of the form of the ${}^S T_T$ matrix used in the program (as it varies over time). Any significant points should also be discussed.

```

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

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

/**
 *
 * Routine to compute the joint angles to generate a circular motion for the
 * Mitsubishi RM-501.
 *
 * Parameters:
 * SimulationTimeInSeconds - total simulation time in seconds.
 * JointAnglesInPulseUnits - an array of 5 floats that is the output and will
 * pass back the translated joint values (in pulse units) to Control.
 * Position - an array of 3 floats that is an output and will pass back
 * the position of the gripper.
 *
 */
void ComputeJointAngles (double SimulationTimeInSeconds, int JointAnglesInPulseUnits[],
    float Position[])
{
    // Calculate transform for the current time
    float Transform[4][4];

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

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

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

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

    // Get the joint angles for that transform
    InverseKinematics (Transform, JointAnglesInPulseUnits);

    // Return the tool position
    Position[0] = Transform[0][3];

```

```

Position[1] = Transform[1][3];
Position[2] = Transform[2][3];
}

/**
 *
 * Changes the input array of joint angles from pulse units referenced to
 * the nest position, to units of radians referenced to the Denavit-Hartenberg
 * zero position for the Mitsubishi RM-501
 *
 * Parameters:
 * @param JointAnglesInPulseUnits - an array of 5 ints holding the joint angles (in pulse units)
 * @param JointAnglesInRadians - an array of 5 floats that is the output and will
 *     pass back the translated joint values (in radians referenced to the
 *     Denavit-Hartenberg zero position)
 */
void TranslatePulseUnitsToRadians (const int JointAnglesInPulseUnits[], float JointAnglesInRadians[])
{
JointAnglesInRadians[0] = (float) JointAnglesInPulseUnits[0];
JointAnglesInRadians[1] = (float) JointAnglesInPulseUnits[1];
JointAnglesInRadians[2] = (float) JointAnglesInPulseUnits[2];
JointAnglesInRadians[3] = (float) JointAnglesInPulseUnits[3];
JointAnglesInRadians[4] = (float) JointAnglesInPulseUnits[4];
}

```