

ECE 763

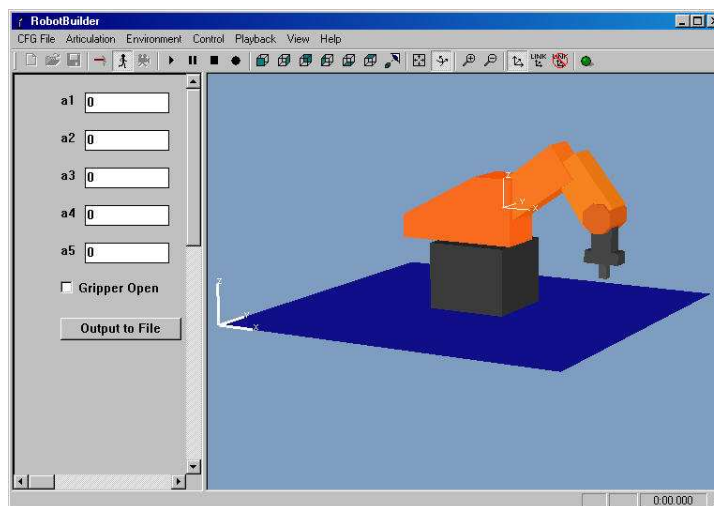
Project #1

The purpose of this project is to use **RobotBuilder** to display different positions and orientations of the Mitsubishi RM-501. **RobotBuilder** is a graphical simulation package that may be used to build models and simulate robot manipulators and legged vehicles. The package was developed by Steven Rodenbaugh for his M.S. thesis in the Dept. of Electrical and Computer Engineering at OSU.

Your task will be to write the C code to convert the Mitsubishi's joint angles from pulse units referenced to the rest position, to units of radians referenced to the Denavit-Hartenberg zero position (see Homework #4). You will also program the forward kinematics equations to compute the homogeneous transform ${}^S T_T$.

The full graphical model of the Mitsubishi has already been developed using **RobotBuilder**. The graphical model for each link uses **XAnimate** file conventions (`.xan` file extension). **XAnimate** is a graphical package, also developed at OSU, that displays 3D objects in **X Windows**. **RobotBuilder** also uses files to configure the simulation (`.cfg` file extension), define the environment (`.env` & `.dat`), and provide the model for the arm (`.dm`). A second graphical model of the Mitsubishi has been built using **RobotModeler** (`.rbm` file extension) but will not be used in this project. **RobotModeler** is an application that comes with **RobotBuilder** that may be used to build graphical models from primitive shapes.

A display of **RobotBuilder** showing the Mitsubishi is given below. For Project #1, you will write the code to show the position and orientation of the Mitsubishi associated with the joint angle values entered in the left window.



A C++ program called `Control` has been written to work with `RobotBuilder` to control the movement of the Mitsubishi and to provide most of the functionality needed for this project. The intended function is that each time joint values are entered (in pulse units), then `RobotBuilder` will move the Mitsubishi to that position. However, to do so, you need to add code to `Control` to convert pulse units to radians, and change the reference from the nest position to the Denavit-Hartenberg zero position. You will also need to add code for the forward kinematics computation.


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

1. Download `RobotBuilder.NET.zip` to a PC equipped with Microsoft Visual Studio .NET. Be sure to use the version of `RobotBuilder.NET.zip` that works for your version of Visual Studio. 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. More information on `RobotBuilder` and `RobotModeler` may be found on your instructor's web page and through the Help menu in these applications.
2. Start `RobotBuilder.exe`, open `mitsu1.cfg` (from the `\RobotBuilder\Projects\Mitsubishi RM-501` directory), and hit the Simulate (⏏) and Play (▶) buttons. Enter some values for a_1, a_2, \dots, a_5 . When functioning properly, `RobotBuilder` will move the Mitsubishi to that position and output these values along with ${}^S T_T$ to a file called `MitsuProject1.txt` if the Output to File button is pressed.
3. Open `Control.sln`, from the `\RobotBuilder\Projects\Mitsubishi RM-501\Control1` directory, with Visual Studio so that you can edit `StudentCode.cpp`. Work with Visual Studio in the Release solution configuration as opposed to the Debug configuration. `StudentCode.cpp` is just a section of code that is linked to `Control` to do the necessary conversions in the joint angles and generate the ${}^S T_T$ homogeneous transform.
4. The initial form for `StudentCode.cpp` is given on the last two pages. The `a` array provides the joint angles in pulse units. The `JointAnglesInRadians` array should be used to store the θ 's according to the Denavit-Hartenberg convention (see Homework #4). The `Transform` array should be used to store the ${}^S T_T$ transform. Note that, as given, the joint angles are not converted and ${}^S T_T$ is set to identity. Also, note that the indices for the `a` array in `StudentCode.cpp` are 0-4 while the indices for the angles from the nest position from Mitsubishi documentation are 1-5.

Change and add to the code so that the angles are properly converted, and ${}^S T_T$ computed.

5. Consider the roll axis of the Mitsubishi to be in the nest position and zero Denavit-Hartenberg position as pictured in Homework #4. Also, pure clockwise rotation, as seen looking at the end-effector from the outside, results when the

pulse units furnished for the wrist (a_4 and a_5) are equal and positive. Note that pure pitching motion occurs when the pulse units are equal and of opposite sign (a_4 positive and a_5 negative for upward movement). Recall that the conversion factor from pulse units to degrees is $3/80$ for both wrist axes.

6. Build Control. It will generate the Control1.dll file in the \RobotBuilder\Projects\Mitsubishi RM-501 directory. Under the Control menu in RobotBuilder, click Select Control DLL ... and choose Control1.dll. (Note that a new Control DLL can only be selected when you are in the Build () mode of RobotBuilder.)
7. Enter several sets of angles through the interface in the left window of RobotBuilder and output these to the MitsuProject1.txt file. Include results for the four positions of Homework #2.

Your report may be relatively brief and should include the following items: a printout of StudentCode.cpp (with your names edited into the comments of the file) and a printout of the results in the MitsuProject1.txt file. Also include the equations used for converting pulse units to Denavit-Hartenberg angles. Any significant points should also be discussed.

```

#include "stdafx.h"
#include <math.h>

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

/**
 *
 * Changes the input array of joint angles from pulse units referenced to
 * the next position, to units of radians referenced to the Denavit-Hartenberg
 * zero position for the Mitsubishi RM-501
 *
 * Parameters:
 * a - an array of 5 floats holding the joint angles inputed (in pulse units)
 * 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) to Control
 *
 */
void TranslatePulseUnitsToRadians (const float a[], float JointAnglesInRadians[])
{
    JointAnglesInRadians[0] = a[0];
    JointAnglesInRadians[1] = a[1];
    JointAnglesInRadians[2] = a[2];
    JointAnglesInRadians[3] = a[3];
    JointAnglesInRadians[4] = a[4];
}

/**
 *
 * Finds the transform  ${}^S T_T$ 
 *
 * Parameters:
 * 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)
 * Transform - the 4x4 of floats that passes back the transform
 *
 */

```

```
void GetTransform (const float JointAnglesInRadians[], float Transform[][4])
{
Transform[0][0] = 1;
Transform[0][1] = 0;
Transform[0][2] = 0;
Transform[0][3] = 0;

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] = 0;

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