EECS567 Midterm Report

Resolved Motion Rate Control Algorithm: Simulation and Implementation

Kris Schilling, Ran Lin, and Yang Xu
University of Michigan
Mechanical Engineering Department

1. Introduction

This paper is intended to give the details of our EECS567 semester project.

1.1. Motivation

Many industrial applications require the end-effector of a robot to move along a desired path at a set velocity. The ability to easily adapt a robotic manipulator for various tasks gives the manufacture a cost benefit of purchasing multiple manipulators and allows for Lean Manufacturing. As opposed to calculating the inverse kinematic equation geometrically, we will review the Resolved Motion Rate Control (RMRC) discussed in [1]. The benefits the method can provide apply to open loop and closed loop control. For example, a human using a joystick to control a robot expects control in the direction specified at a speed proportional to the input. In this project, our team wants to review the RMRC method, overview the benefits and limitations, and get experience implementing the algorithm on physical hardware.

Figure 1.1: *OWI ARM*

1.2. Objective

The object of the project will be to implement a Rate Motion Control algorithm using a low-cost commercial off the shelf (COTS) robotic manipulator. The mathematics derivations of the Resolved Motion Rate Control will first be derived. Following this, we will define the DH parameters for the OWI robotic arm and the Jacobian Matrix. Graphical Simulations of the Rate Motion Control will be completed using the Robotics Tool Box [2]. Finally, we will retrofit the robotic arm with position feedback, h-bridge motor controllers, and a computer control interface. The robotic manipulator selected is an OWI Robotic Arm Edge, Figure 1.1. The simulated RMRC will be implemented with closed loop feed back control on the OWI arm allowing a user to complete tasks such as writing on a piece of paper.

2. Resolved Motion Rate Control

Resolved motion rate control utilizes the inverse Jacobian matrix to convert translation velocities to joint angular velocities using the form:

$$\dot{\theta} = J^{-1}\dot{x} \tag{2.1}$$

In order for the results of this equation to converge on the desired location, we can use a error relationship between the desired location and the current location. This can be combined with the desired translation velocities to give

$$\dot{\theta} = J^{-1}(\dot{x_d} + K(x_d - x))[3] \tag{2.2}$$

This equation can be used with Euler integration and time step Δt to iterate and converge to the final location. The gain factor K is used to control the rate of convergence.

However, we know that the Jacobian can become singular is some configurations, such as in the initial configuration of the OWI arm shown in Figure 3.4. To avoid these singularities, we can modify the inverse jacobian to use a singular robust inverse discussed in Nakamura [4].

$$J^* = J^T(JJ^T + k_I I)^{-1}[3]H \tag{2.3}$$

Here k_I is used to prevent singularities. These equations can be used to generate the motion trajectories.

3. OWI-535 Robot Arm

3.1. Forward Kinematics

The OWI-535 robot arm consists four revolute joints and an end-effector, which has a four bar linkage to grab an object. The dimension of each link of the robot arm is presented below in Figure 3.1. For each revolute joint, there is certain limitations of angular rotation. The limit of each revolute joint is presented in Figure 3.2.

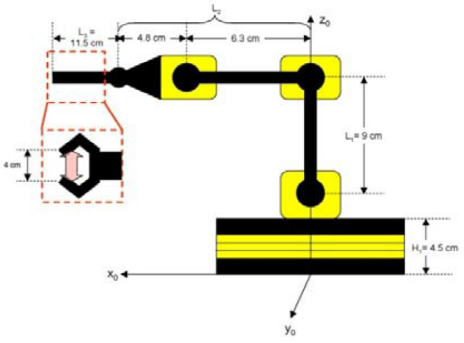


Figure 3.1: *Dimensions of the OWI-535 robot arm manipulator[3]*

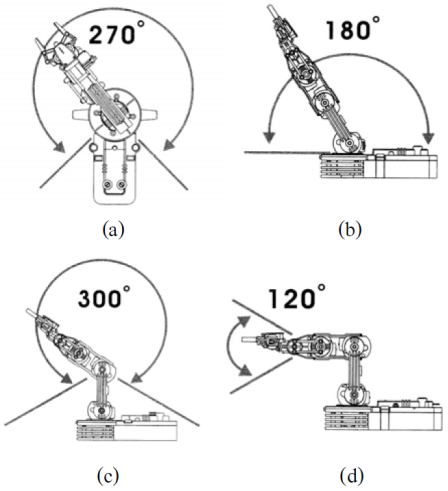


Figure 3.2: *Limitation of angular rotation of each joint[3]*

With the dimensions known, coordinate frames from 0 to 4 are assigned to each link of the robot manipulator. The OWI-535 robot is represented symbolically by figure 3.3 and the frame assignment is also represented on

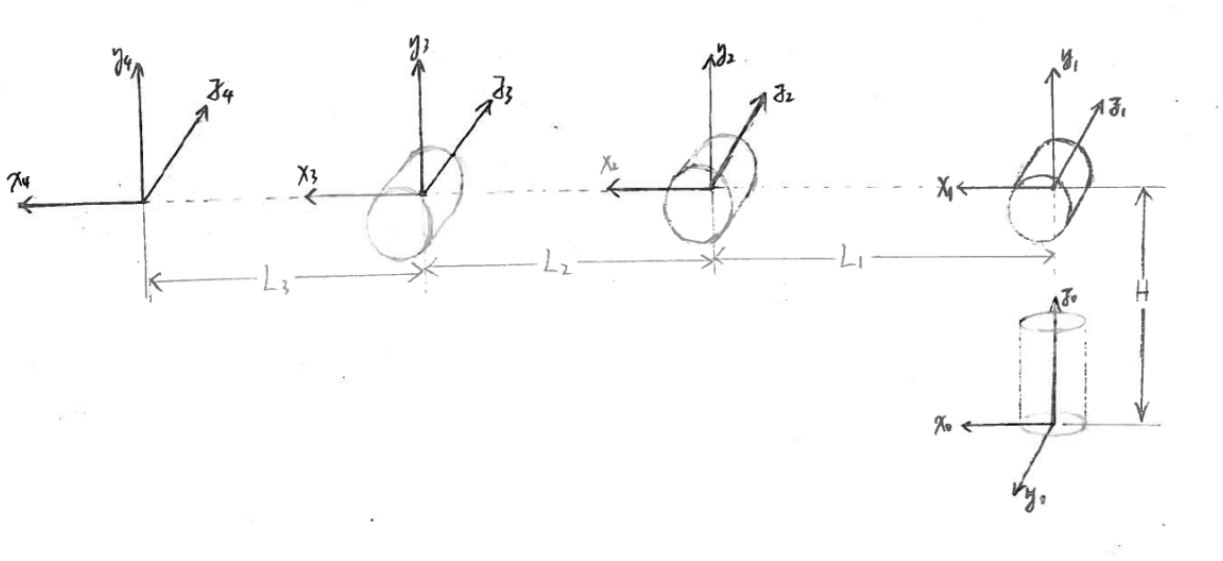


Figure 3.3: *DH coordinate frame assignment for the OWI-535 robot arm manipulator*

Link	a	α	d	θ
1	0	90°	$4.5cm$	θ_1
2	$9cm$	0	0	θ_2
3	$11.1cm$	0	0	θ_3
4	$11.5cm$	0	0	θ_4

Table 3.1: *DH parameters for OWI-535 robot arm manipulator*

the same figure. The z axis of frame 0 is pointing upward, while the z axes of frame 1 to 4 are parallel and pointing out from the paper.

Following the Denavit-Hartenberg convention, the DH parameters for each link are shown in table 3.1, where $H1$, $L1$, $L2$, $L3$ are represented in Figure 3.3.

The A matrices for joint 1 to 4 are derived from the table above with equation (3.1).

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \tag{3.1}$$

The forward kinematics from frame 0 to frame 4 can be then computed by the transform matrix $T_4^0 = A_1 A_2 A_3 A_4$, which denote the position and orientation of the end-effector of the robot manipulator. The A matrices and the transform matrix are given in Appendix A.

A MATLAB® model of the OWI-535 robot arm is generated and the graph of the robot arm at its initial position is presented in the Figure 3.4.

3.2. Velocity Kinematics and the Jacobian

At a certain configuration, the velocity relationship between the end-effector and the joint positions are determined by the Jacobian. The Jacobian matrix relates the end-effector linear velocity and angular velocity with the speed of four revolute joints as shown in Equation (3.2).

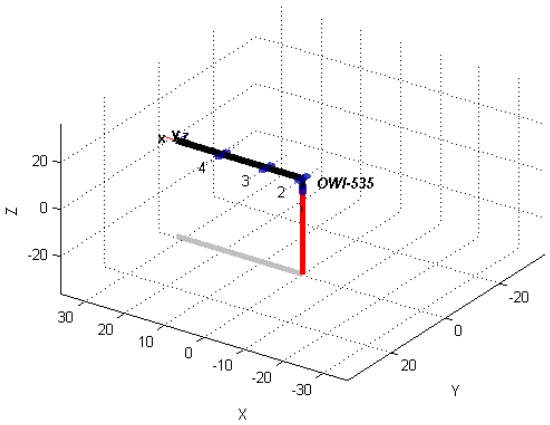


Figure 3.4: *OWI-535 robot arm at its initial position*

$$\begin{bmatrix} V_x \\ V_y \\ V_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \partial x/\partial\theta_1 & \partial x/\partial\theta_2 & \partial x/\partial\theta_3 & \partial x/\partial\theta_4 \\ \partial y/\partial\theta_1 & \partial y/\partial\theta_2 & \partial y/\partial\theta_3 & \partial y/\partial\theta_4 \\ \partial z/\partial\theta_1 & \partial z/\partial\theta_2 & \partial z/\partial\theta_3 & \partial z/\partial\theta_4 \\ \partial\psi/\partial\theta_1 & \partial\psi/\partial\theta_2 & \partial\psi/\partial\theta_3 & \partial\psi/\partial\theta_4 \\ \partial\theta/\partial\theta_1 & \partial\theta/\partial\theta_2 & \partial\theta/\partial\theta_3 & \partial\theta/\partial\theta_4 \\ \partial\phi/\partial\theta_1 & \partial\phi/\partial\theta_2 & \partial\phi/\partial\theta_3 & \partial\phi/\partial\theta_4 \end{bmatrix} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \\ \dot{\theta_4} \end{bmatrix} \tag{3.2}$$

For revolute joints, the Jacobian matrix can be determined by the following equations

$$J = \begin{bmatrix} J_{v1} & \cdots & J_{vn} \\ J_{\omega 1} & \cdots & J_{\omega n} \end{bmatrix} where \tag{3.3}$$

$$J_{vi} = z_{i-1} \times (o_n - o_{i-1}) \quad and \quad J_{\omega i} = z_{i-1} \tag{3.4}$$

This way the Jacobian for a configuration $(\theta_1, \theta_2, \theta_3, \theta_4)$ is determined and the entries of the Jacobian matrix are listed in Appendix B.

3.3. Simulation

Using the equations developed in Section 2, we simulated moving the robotic arm from:

$$\theta = [0 \quad 90° \quad 0 \quad 0]$$

We set the desired position and speed as

$$x_d = [15 \quad 15 \quad 3]cm$$

and

$$\dot{x_d} = [0.2 \quad 0.01 \quad 0.05]cm/s$$

respectively.

As can be seen in Figure 3.5, we see that the arm approaches the desired location at a fixed speed. The code used to complete the simulation can be found in Appendix C.

3.4. Workspace Limits

In order to complete the project, we need to modify our algorithm to stay within the workspace of the robot. After this is complete we will implement the RMRC OWI Robotic ARM.

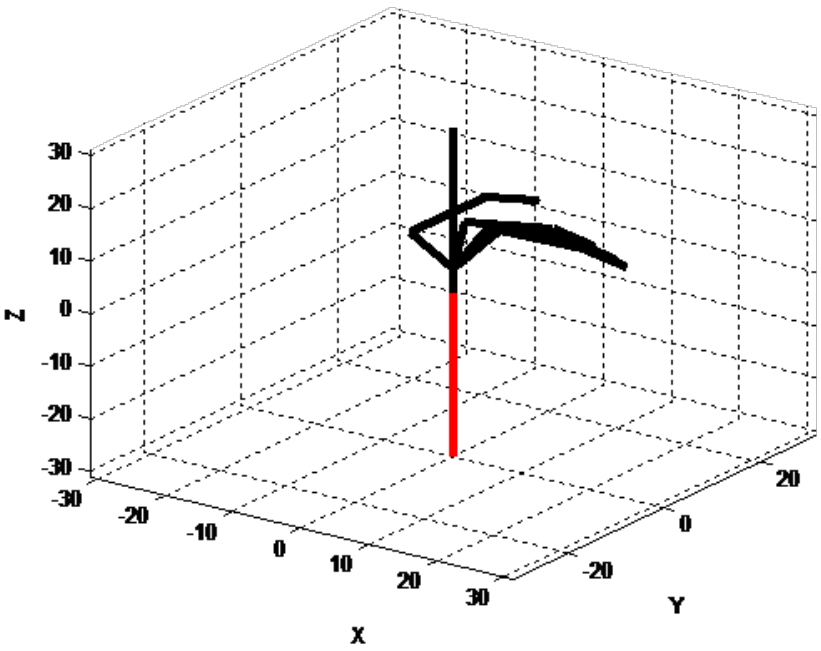
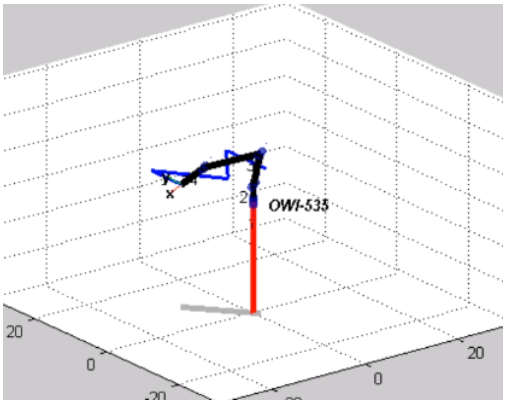


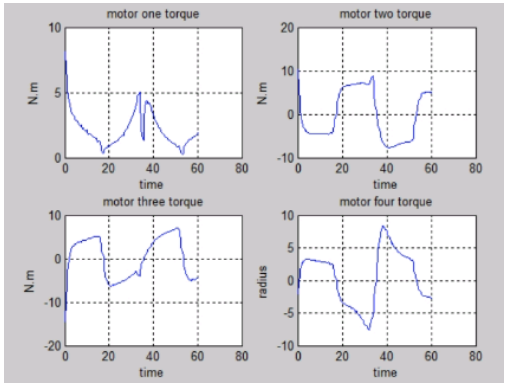
Figure 3.5: *OWI-535 robot RMRC Simulation*

3.5. Dynamics

****** DYNAMICS SECTION *******



(a) *Simulation drawing M*



(b) *Torques during simulation*

Figure 3.6: *Dynamic Simulation*

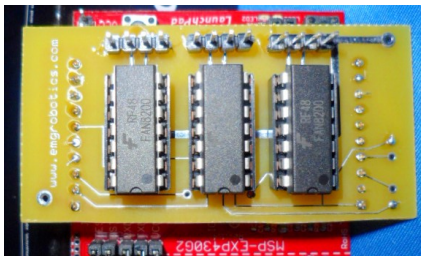
4. Implementation

This section gives an overview of the implementation of the RMRC on the OWI robot.

4.1. Hardware Modifications

The stock OWI robot is controlled by a controller with toggle switches. No method of automatic controller or position feedback is provided. In order to implement the RMRC method, we had to modify the robot with 5

H-Bridge motor controllers and add absolute positional feedback. The motor control is completed with a Arduino Uno with a EMGRobotics Six Pack, Figure 4.1. The six pack includes 3 FAN82000 Dual H-Bridges.



(a) *EMG Robtoics Six Pack*

(b) *Arduino Uno*

Figure 4.1: *Electronics Hardware*

In order to get positional feedback we had to design a way to mount potentiometers to each joint. Each potentiometer is 0-10KΩ, and is used to give the absolute position back to the Arduino. As seen in Figure 4.2, all potentionmeters are were located on the axis of the joint to allow measuring the full motion of each joint.

Currently, the software only allows full on/off control of the motors. To implement the RMRC method we will modify the software to allow Pulse Width Modulation (PWM) control.

4.2. Software Implementation

The motor control was implemented using a Proportional Integral (PI) loop on the Arduino. When the Arduino boots up, it captures the current joint positions of each arm and set this postion as the current target. The Arduino then waits for serial commands to set the newest joint location. Each joint can be commanded to a position using a command structure in HEX

$$\text{COMMAND} = 0x55\ 0xAA\ 0x0J\ 0x\#\#$$

After receiving the command, the Arduino will move the joint to this location. All joints can operate concurrently and joint positions can be changed rapidly (sub 1 ms).

In MATLAB®, we setup a serial port class that automatically handles receiving the postional data from the Arduino. The setup of this class is important for recreating the work or similar tasks so it is included in Listing 1.

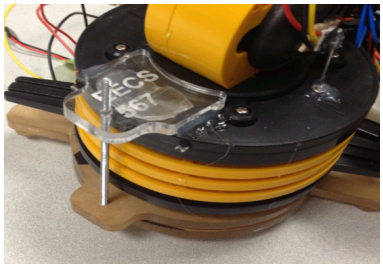
After Serial communication is established, the user can send commands to Arduino using the function setJointPostion shown in Listing 2. This function takes the desired joint and angle that matches the DH convention for the robot, formats it, and sends it to the Arduino.

Using the functions created above, we created a program in MATLAB® that runs our RMRC algorithm and sends the positions to the Arduino. It sends the first four joint locations, can constantly calculates the max error of all joints. When the max error falls below a given threshold, the program sends the next command sequence.

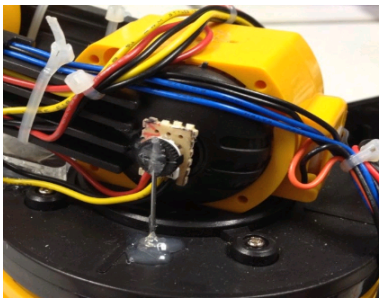
5. Testing

After building the hardware and completing the software, our team ran tests using the modified OWI robot. We set out to complete two objectives. The first was to complete a task, by defining the the way points at each several positions and having the robot move through each. The task was setup to have the robot move a washer from one location to another similar to a pick and place machine. This task was selected as it allowed us to understand the control algorithm tuning and verify the robot was ready to test RMRC. Figure 5.1a shows the robot picking up the washer.

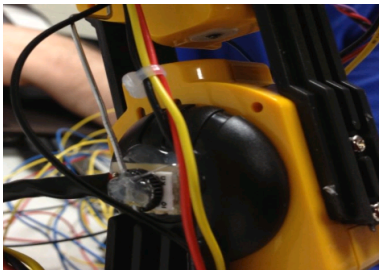
The second objective was to test the RMRC algorithm on the robot. We decided to have the robot paint a line on a piece of paper. This task was completed by setting up a line in the work space using MATLAB® and specifying the desired speed in World Coordinates. Figure 5.1 shows the robot painting a line.



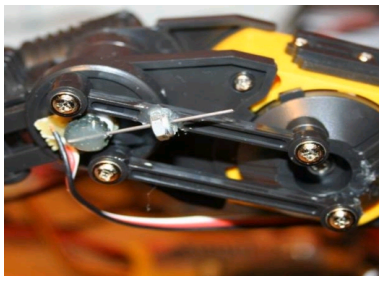
(a) *Base*



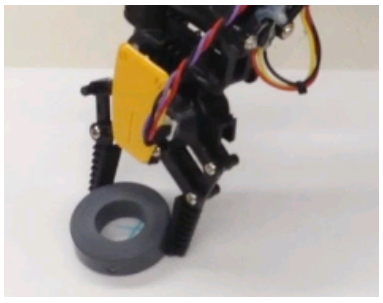
(b) *Joint 1*



(c) *Joint 2*

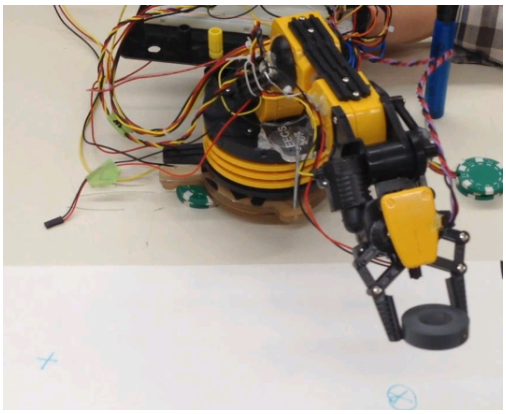


(d) *Joint 3*

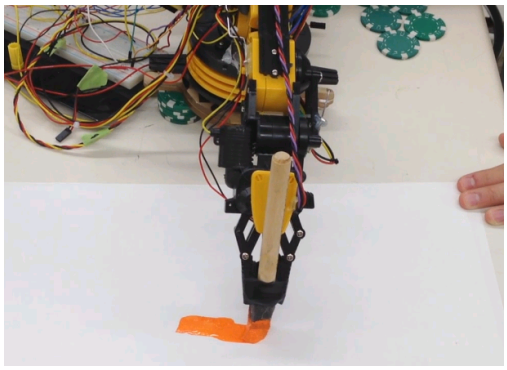


(e) *Joint 4*

Figure 4.2: *Potentiometer Mountings*



(a) *Completing Task*



(b) *Following Path*

Figure 5.1: *Physical Testing*

5.1. Challenges

The testing showed the limitations of the hardware we were using. Although the simulation operated smoothly, the robot movement was jerky with limited resolution of the encoder and low power motors operating high friction joints.

In future implementations it is recommended to use high resolution encoders so speed control is possible and investigate higher power motor with lower gear lash. As well, once speed control is available, it is recommended that the full algorithm be implemented on a microprocessor to allow stand alone operation using a fixed point math.

Listing 1: *Main Loop*

```
%% EECS567 Serial Setup
% Included for completeness.
COMPORT = 'COM5' %Differs on each PC.
delete(instrfind) %Deletes all instrument objects
s = serial(COMPORT); %Creates the Serial Port Object
s.BytesAvailableFcnMode = 'terminator'; %Serial Stream of sends 5 Joints
% encoder counts followed by CR/LF
s.Terminator = 'CR/LF'
s.BytesAvailableFcn = @mycallback %Function Called when CR/LF is seen
%Setup Data Storage Locations (Can create others as well)
s.UserData.isNew = 0;
s.UserData.newData = 0;
 %Opens the Serial Communcation if currently not open
 if strcmp(s.Status,'closed'), fopen(s); end
```

Listing 2: *Serial Port Setup*

```
    function setJointPos(Obj,Joint,Position)
%setJointPos Serial object, Joint Num (0-4), Desired Postion(Degrees)
%   Includes bounds checking for angle.
```

1

6. Conclusion

Overall, the RMRC method allows for simple iterative method to follow a path. The advantage is that it could be used in real time on a controller using lookup tables for sin and cosine and pointer matrix math. Ideal situations would be on mobile robots that include manipulators that are human controlled live.

Our implementation was not in real time but did add both joint limitation considerations and investigated dynamics beyond the standard RMRC algorithms we studied.

References

[1] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *Man-Machine Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 47–53, 1969.

[2] P. Corke, "A robotics toolbox for matlab," *Robotics & Automation Magazine, IEEE*, vol. 3, no. 1, pp. 24–32, 1996.

[3] P. Prempraneerach and P. Kulvanit, "Implementation of resolved motion rate controller with5-axis robot manipulator arm," in *The First TSME International Conference on Mechanical Engineering*, 2010.

[4] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *ASME, Transactions, Journal of Dynamic Systems, Measurement, and Control*, vol. 108, pp. 163–171, 1986.

A.
Denavit-Hartenberg Matrices

$$A_1 = \begin{bmatrix} \cos_1 & 0 & \sin_1 & 0 \\ \sin_1 & 0 & -\cos_1 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} \cos_2 & -\sin_2 & 0 & L_1 \cos_2 \\ \sin_2 & \cos_1 & 0 & L_1 \sin_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$A_3 = \begin{bmatrix} \cos_3 & -\sin_3 & 0 & L_2 \cos_3 \\ \sin_3 & \cos_3 & 0 & L_2 \sin_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_4 = \begin{bmatrix} \cos_4 & -\sin_4 & 0 & L_3 \cos_4 \\ \sin_4 & \cos_4 & 0 & L_3 \sin_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$T_4^0 = \begin{bmatrix} \cos_1 \cos_{234} & -\cos_1 \sin_{234} & \sin_1 & (L_1 \cos_2 + L_2 \cos_{23} + L_3 \cos_{234}) \cos_1 \\ \sin_1 \cos_{234} & -\sin_1 \sin_{234} & -\cos_1 & (L_1 \cos_2 + L_2 \cos_{23} + L_3 \cos_{234}) \sin_1 \\ \sin_{234} & \cos_{234} & 0 & L_1 \cos_2 + L_2 \cos_{23} + L_3 \cos_{234} + H \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

B.
Jacobian Matrix Entries

$$\partial x/\partial\theta_1 = -(L_1C_2 + L_2C_{23} + L_3C_{234})S_1$$
$$\partial x/\partial\theta_2 = -(L_1S_2 + L_2S_{23} + L_3S_{234})C_1$$
$$\partial x/\partial\theta_3 = -(L_2S_{23} + L_3S_{234})C_1$$
$$\partial x/\partial\theta_4 = -(L_2S_{23} + L_3S_{234})C_1$$
$$\partial y/\partial\theta_1 = (L_1C_2 + L_2C_{23} + L_3C_{234})C_1$$
$$\partial y/\partial\theta_2 = -(L_1S_2 + L_2S_{23} + L_3S_{234})S_1$$
$$\partial y/\partial\theta_3 = -(L_2S_{23} + L_3S_{234})S_1$$
$$\partial y/\partial\theta_4 = -(L_3S_{234})S_1$$
$$\partial z/\partial\theta_1 = 0$$
$$\partial z/\partial\theta_2 = (L_1C_2 + L_2C_{23} + L_3C_{234})$$
$$\partial z/\partial\theta_3 = (L_2C_{23} + L_3C_{234})$$
$$\partial z/\partial\theta_3 = L_3C_{234}$$
$$\partial\psi/\partial\theta_1 = 0$$
$$\partial\psi/\partial\theta_i = S_1 \text{ where i} = 2, 3, 4$$

$$\partial\theta/\partial\theta_1 = 0$$
$$\partial\theta/\partial\theta_i = -C_1 \text{ where i} = 2, 3, 4$$

$$\partial\phi/\partial\theta_1 = 1$$
$$\partial\phi/\partial\theta_i = 0 \text{ where i} = 2, 3, 4$$

C.
Source Code

The code for this project was created using Arduino IDE V1.1 and MATLAB® R2011b. The project utilizes open source Arduino Libraries and the code has been setup for future use.

The MATLAB® files included customizable functions for receiving Serial Data and sending Serial Data via RS-232 with the Arudino Uno or other similar RS-232 device.

All source code and MATLAB® files can be found on GitHub repository at

https://github.com/kjschill/OWI_ARM

As well, the project files have been uploaded to CTools site for EECS567 W13 class projects.

For access to the project, please contact:

Kris Schilling
kjschill@umich.edu
University of Michigan