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 a progress update for our 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

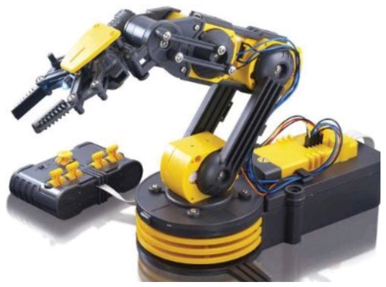

Figure 1.1: *OWI ARM*

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].

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

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.

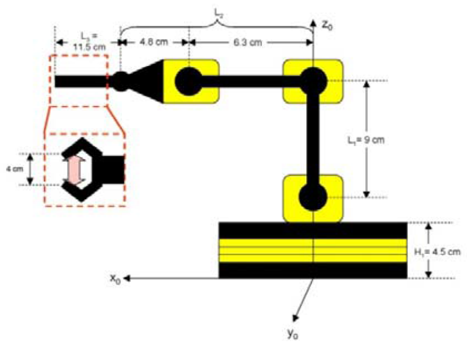

Figure 3.1: *Dimensions of the OWI-535 robot arm manipulator[3]*

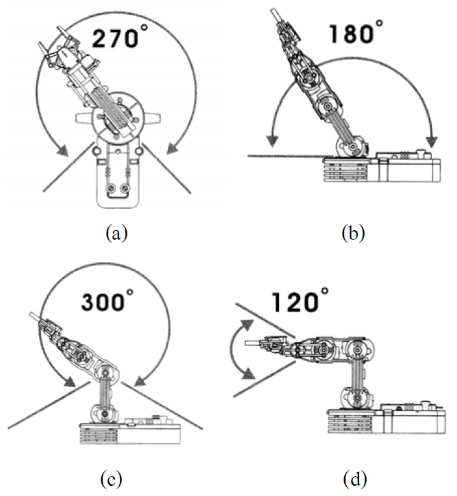

Figure 3.2: *Limitation of angular rotation of each joint[3]*

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 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.

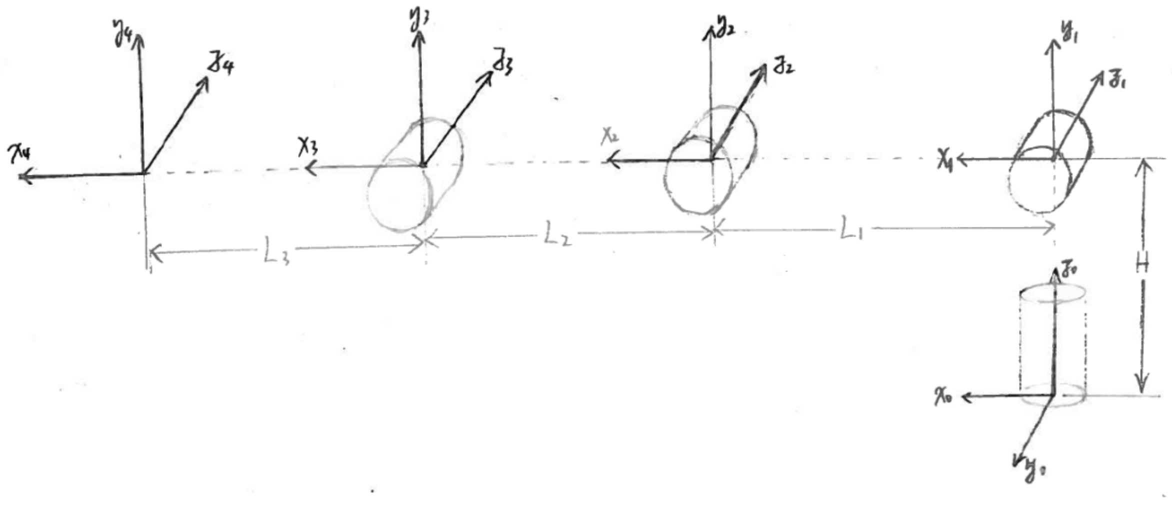

Figure 3.3: *DH coordinate frame assignment for the OWI-535 robot arm manipulator*

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

Link	a	α	d	θ
1	0	90	H	θ_1
2	L_1	0	0	θ_2
3	L_2	0	0	θ_3
4	L_3	0	0	θ_4

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).

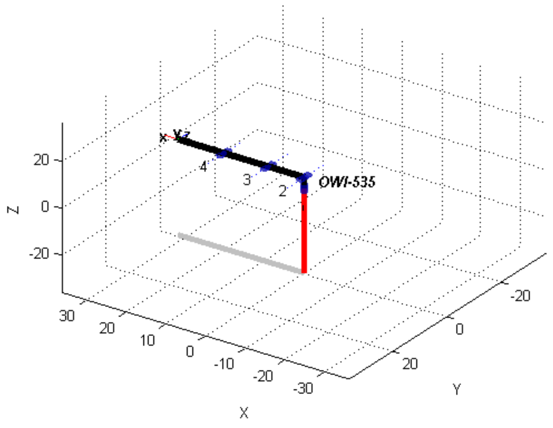

Figure 3.4: *OWI-535 robot arm at its initial position*

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

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:

$$\theta = [0 \quad 90^\circ \quad 0 \quad 0]$$

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.

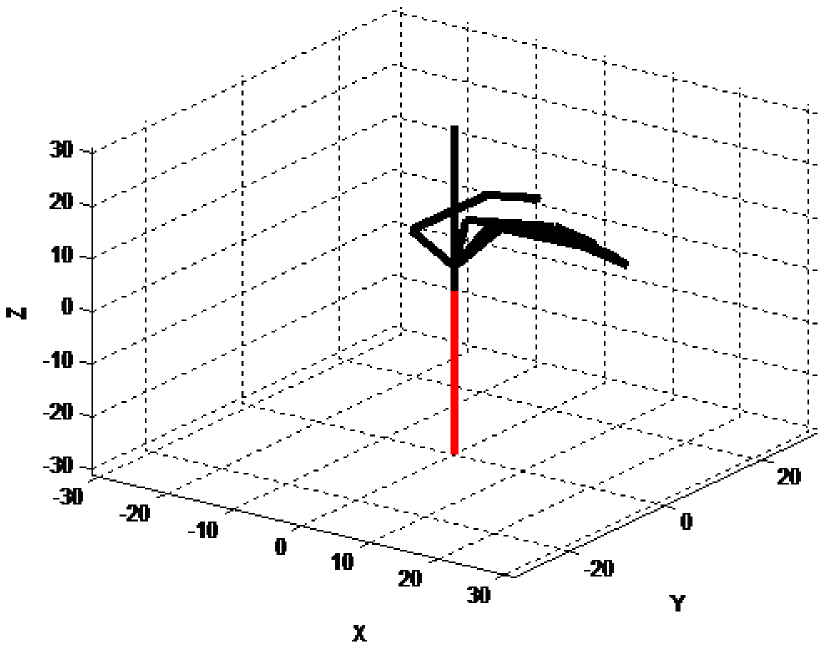

Figure 3.5: *OWI-535 robot RMRC Simulation*

3.4. Hardware Modification

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 Texas Instrument MSP430 Launchpad with a EMGRobotics Six Pack, Figure 3.6. The six pack includes 3 FAN82000 Dual H-Bridges. Code was implemented to allow an RS-232 Serial interface with the computer. This allows control of each motor. Currently, the software only allows full on/off control of the motors. To

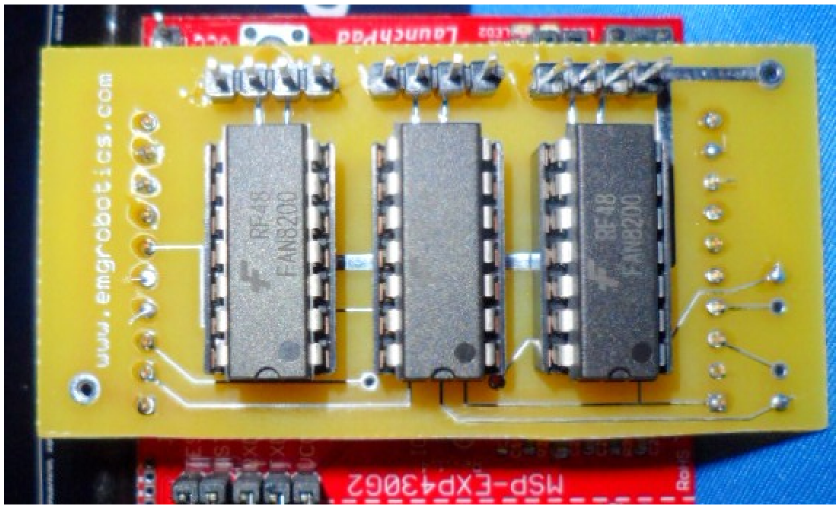

Figure 3.6: *EMGRobotics Six Pack*

implement the RMRC method we will modify the software to allow Pulse Width Modulation (PWM) control.

4. Tasks Remaining

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.

4.0.1. Summary of Tasks Remaining

- Modify Algorithm to incorporate Workspace
- Add Feedback Control to OWI
- Update MSP430 C Code to include PWM and positional feedback
- Simulate updated RMRC algorithm for completing a task
- Test on OWI Robotic Arm

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. Matlab Code

```
close all
clear all
clc
%startup_rvc
% Inverse Kinematic
% DH parameters
%         |a |alpha |d   |theta
% Link 1 |0 |  90  |H   |theta1
% Link 2 |L1|  0   |0   |theta2
% Link 3 |L2|  0   |0   |theta3
% Link 4 |L3|  0   |0   |theta4

%% The General Translation Matrix
syms theta alpha a d

A = [cos(theta) -sin(theta)*cos(alpha) sin(theta)*sin(alpha) a*cos(theta);
    sin(theta) cos(theta)*cos(alpha) -cos(theta)*sin(alpha) a*sin(theta);
    0 sin(alpha) cos(alpha) d;
    0 0 0 1];

a90 = sym(pi/2);

syms H L1 L2 L3 t1 t2 t3 t4

A10 = subs(A,[theta alpha a d],[t1 a90 0 H]);
A21 = subs(A,[theta alpha a d],[t2 0 L1 0]);
A32 = subs(A,[theta alpha a d],[t3 0 L2 0]);
A43 = subs(A,[theta alpha a d],[t4 0 L3 0]);

A20 = simplify (A10*A21);
A30 = simplify (A20*A32);
A40 = simplify (A30*A43);

% Limitation on theta
%% Link definition (theta d a alpha)
L(1)=Link([0,4.5,0,pi/2]);
L(2)=Link([0,0,9,0]);
L(3)=Link([0,0,11.1,0]);
L(4)=Link([0,0,11.5-5,0]);

owi535=SerialLink(L,'name','OWI-535');
% initial theta
Init_theta = [0 pi/2 0 0];
T_init = owi535.fkine(Init_theta);
Pos_init = T_init(1:3,4);

% Initial position
figure
owi535.plot(Init_theta);

% % desired position
Pos_desire = [15;15;3];
Current_theta = Init_theta;
Current_Pos = Pos_init;

% % position difference
epsilon = 0.2;
step = 0.1;
K = 2;
Spd_desire=[0.2;0.01;0.05];

% parameters to adjusting the singularity
```

```
w0 = 0.01;
k0 = 0.001;

% joint velocity saturation limit
deg = pi/180;

limit_theta_dot_p = 0.1*[270*deg/step;180*deg/step;300*deg/step;120*deg/step];
limit_theta_dot_n = -0.1*[270*deg/step;180*deg/step;300*deg/step;120*deg/step];

while (sqrt(sum((Pos_desire - Current_Pos).^2)) > epsilon)

    A10_real = double(subs (A10,[t1 a90 0 H],[Current_theta(1) pi/2 0 4.5]));
    A21_real = double(subs(A21,[t2 0 L1 0],[Current_theta(2) 0 9 0]));
    A32_real = double(subs(A32,[t3 0 L2 0],[Current_theta(3) 0 11.1 0]));
    A43_real = double(subs(A43,[t4 0 L3 0],[Current_theta(4) 0 6.5 0]));

    A20_real = A10_real*A21_real;
    A30_real = A20_real*A32_real;
    A40_real = A30_real*A43_real;
%
    z00 = [0;0;1];
    z10 = A10_real(1:3,3);
    z20 = A20_real(1:3,3);
    z30 = A30_real(1:3,3);

    o00 = [0;0;0];
    o10 = A10_real(1:3,4);
    o20 = A20_real(1:3,4);
    o30 = A30_real(1:3,4);
    o40 = A40_real(1:3,4);

    Pn = o40;

    Jw (:,1) = z00;
    Jw (:,2) = z10;
    Jw (:,3) = z20;
    Jw (:,4) = z30;

    Jv (:,1) = cross (z00,(Pn - o00));
    Jv (:,2) = cross (z10,(Pn - o10));
    Jv (:,3) = cross (z20,(Pn - o20));
    Jv (:,4) = cross (z30,(Pn - o30));

    % Jacobian with only velocity
    J = [Jv];

    % check whether there is singularity
    w = sqrt(det(J*J'));
    if (w < w0)
        k1 = k0*(1-w/w0)^2
    else
        k1 = 0
    end

    % pesudo jacobian calculation
    J_pesudo = double(J'*inv(J*J'+k1*eye(size(J*J'))));

    % get theta dot
    theta_dot = J_pesudo * (Spd_desire + K*(Pos_desire-Current_Pos));

    % saturation limit on theta_dot
    theta_dot = min(theta_dot,limit_theta_dot_p);
    theta_dot = max(theta_dot,limit_theta_dot_n)

    % get current theta
    Current_theta = double(Current_theta + theta_dot' * step);

    % get current end effector position
    % current tranmission matrix
    T_current = owi535.fkine(Current_theta);

    % assign this to the current position
    Current_Pos = T_current(1:3,4)

%   plot(Current_Pos)
    owi535.plot(Current_theta);
    hold on
end

hold off
```