Week 6: Joint Velocity

Author: Keith Chester

E-mail: kchester@wpi.edu

Table of Contents

Introduction

In the sixth week of *RBE 500 - Foundations of Robotics*, we explore velocity kinematics and the Jacobian, as well as introducing ourselves to Mathwork's MATLAB Robotics Toolbox. To do this, we again visit the Denso HSR Robotic Arm that we've used for numerous examples in prior weeks.

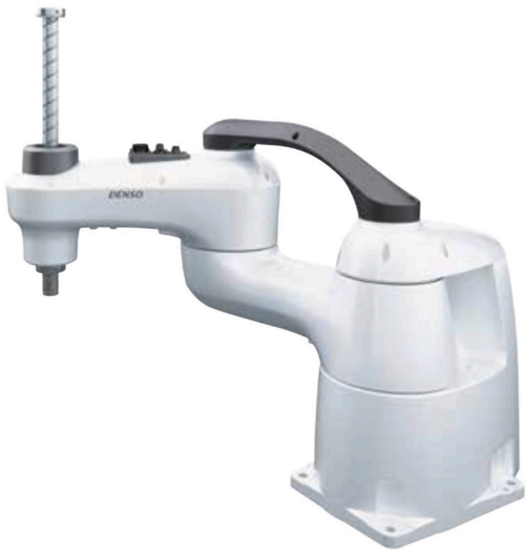


Figure 1: Denso HSR Robot from product catalog

Our first task, **VI-1**, will consist of applying the Denavit–Hartenberg (DH) Convention to the HSR arm. We then utilize the resulting DH table to build the arm within the robotics toolbox, and simulate a set of movement over predefined ranges for joints 1 and 2. While simultaing the movement, we will generate an animation of the arm. After simultaing the movement, we will take a look at the trapezoidal velocity profile for the change in joint thetas as per the robotics toolbox.

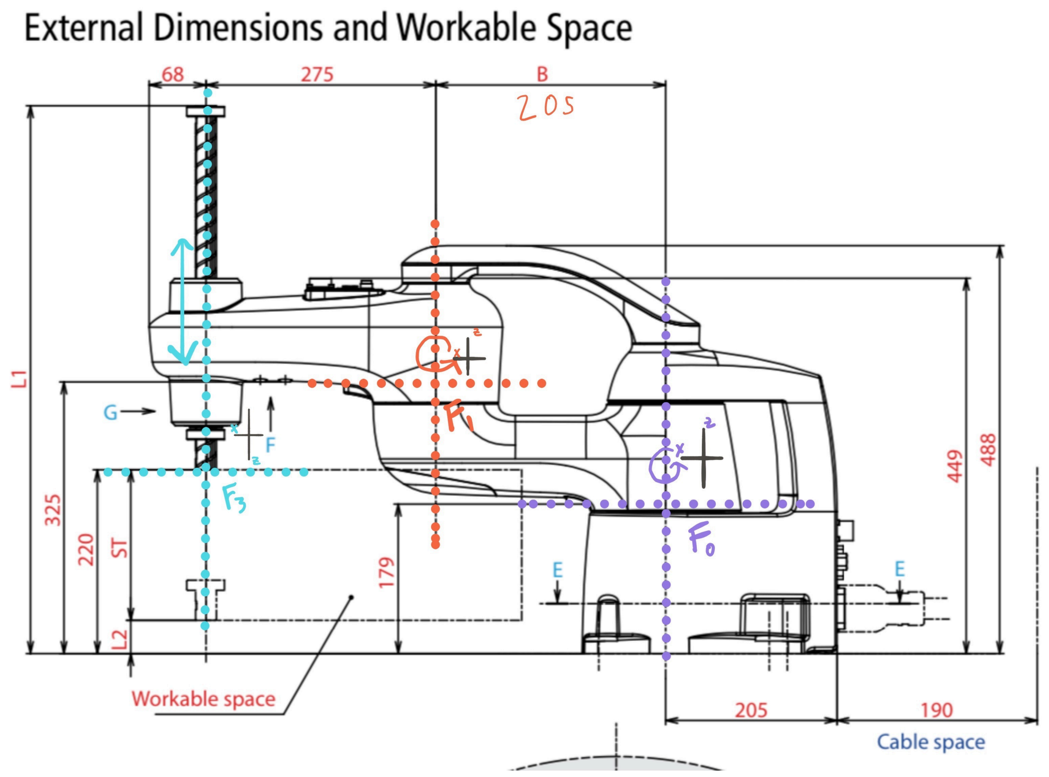
For **VI-2**, we will then use the DH table to manually generate a Jacobian matrix as per our readings and lessons this week. We will compare our calculated Jacobian to the toolbox provided Jacobian to note the differences. Finally, using the Jacobian and qdot as calculated by the trapezoidal velocity functions of the toolbox, we will calculate and map the linear and angular velocities of the actuator tool.

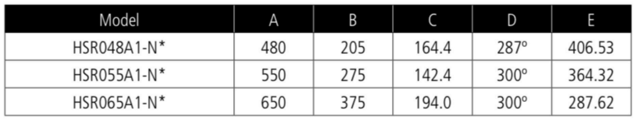
Methods

n this section, we discuss how we approached and performed each problem.

VI-1

To start, we shall define our DH axis and generate a DH table for the Denso Arm. First, we take note of the specified arm measurements from the catalog:





Model	A	B	C	D	E
HSR048A1-N*	480	205	164.4	287°	406.53
HSR055A1-N*	550	275	142.4	300°	364.32
HSR065A1-N*	650	375	194.0	300°	287.62

Figure 2: The Denso HSR Robot Arm Schematics

With these measurements in hand, we can place our axes for each of the frames according to DH convention:

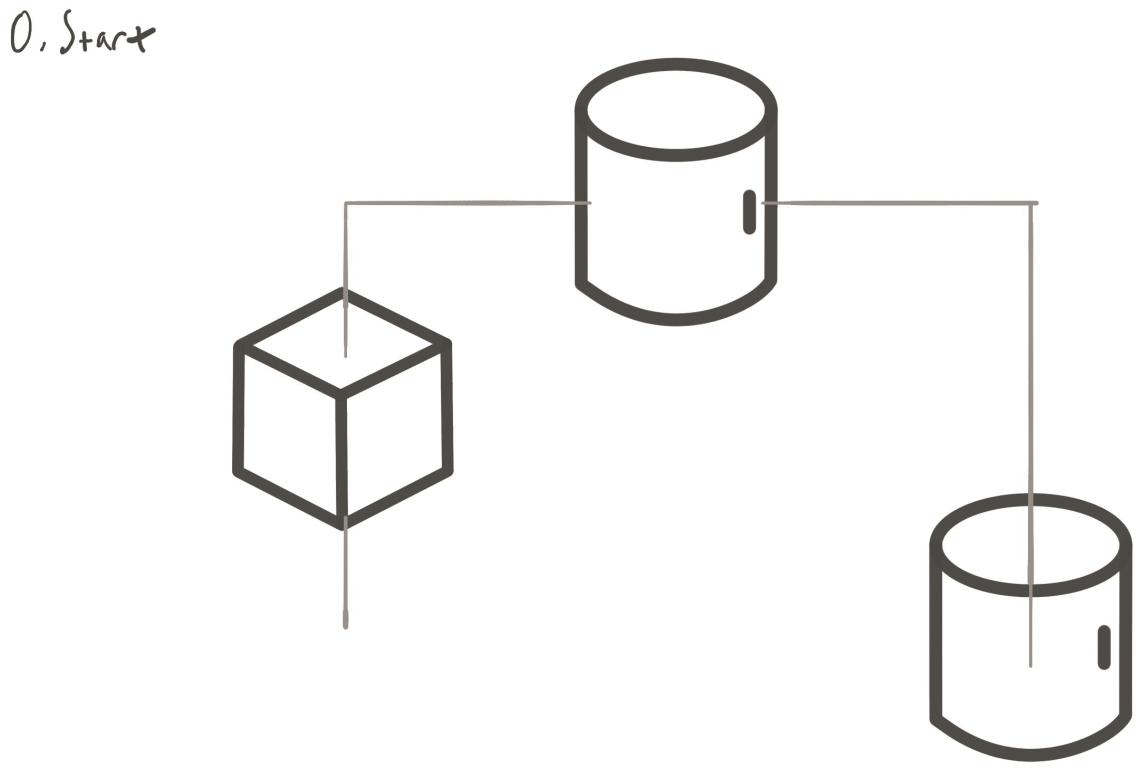


Figure 3: Applying DH Convention to drawing axis for the Denso HSR

1. For frame o_0, we place the z-axis in line with the axis of rotation itself. We place the x-axis perpendicular to the z-axis frame, pointing towards the next frame. The y-axis is placed according to the right-hand rule.
2. For frame o_1, we place the z-axis in line with the axis of rotation itself. We orient it in the same manner as the prior frame to simplify the math around the transformations. We place the x-axis perpendicular to the z-axis frame, pointing towards the next frame along the 275mm link. The y-axis is placed according to the right-hand rule.
3. For frame o_2, we place the z-axis in line with the prismatic joint's movement, oriented in the direction of its movement. Most notably this means a 180 degree rotation on the x to point down vs previous frame's z-axis pointing up. We place the x-axis perpendicular to the z-axis frame, pointing in the same direction past x-axis have. The y-axis is placed according to the right-hand rule.
4. For frame o_3, we perform no rotations or displacements from the prior frame, save for the displacement along the z-axis for the prismatic joint between its base and the tip of the actuator. This distance (specified by **d1**) is the sole change in the frame.

Once we have the DH axes specified, and, using the diagram of measurements for the HSR arm, we can setup the DH table as such:

Link	Θ	d	a	α
1	phi	146	205	0
2	theta	-41	275	180
3	0	d1	0	0

Figure 4: The reslting DH table

Once we have the DH table, we can make use of the robotics toolbox to build the arm.

VI-2

From running the arm simulation in **VI-1**, we have two important generated values. We have, first, the step of the rotation of each joint (ϕ and θ, for us). We'll make use of these later.

For the first stage of **VI-2**, we will calculate the Jacobian to compare it to the toolbox provided Jacboian function *geometricJacobian*. To calculate the Jacobian, we will:

1. Consider the robot in a given configuration for its moment - IE the set joint angles for each joint.
2. For the given configuration, acquire the transformation matrix for H_i^0, where *i* is a given joint's frame.
3. Using these transformations, establish the origin for each coordinate frame to the base/world coordinate frame.
4. Using the DH table coordinate axes specified, create z vectors that represent that rotation of the z axis relative to the base frame.
5. Create the Jacobian - one column of 6 rows for each joint (for our purposes, 4 - one for each joint, and a fourth for the endpoint).

To create the Jacobian, we determine first if the joint is prismatic or revolute. If a joint is prismatic, then the linear velocity is $z_{i-1}x(o_n - o_{i-1})$ and its angular velocity is z_{i-1}. If the joint is revolute, then the linear velocity is z_{i-1} and the angular velocity is 0. In all of these equations, *n* is the number of frames (4 for our purposes) and *i* is the considered frame. o_n is the origin of the given joint within the frame of the base coordinate system.

From this, we build the Jacobian for our arm to be:

$$J = \begin{bmatrix} z_0 x(o_4 - o_0) & z_1 x(o_4 - o_1) & z_2 & z_3(o_4 - O_3) \\ z_0 & z_1 & 0 & z_3 \end{bmatrix}$$

Figure 5: Jacobian Equation for the Denso HSR Arm

Once we have the Jacobian, we can find the velocities of ξ, as below:

$$\xi = \begin{bmatrix} v_{\text{linear}} \\ v_{\text{angular}} \end{bmatrix}$$

$$\xi = J\dot{q}$$

Figure 6: Getting velocities with the Jacobian

Results

VI-1

In this section, we make use of the robotics toolbox to create the robotic arm and create the following animation:

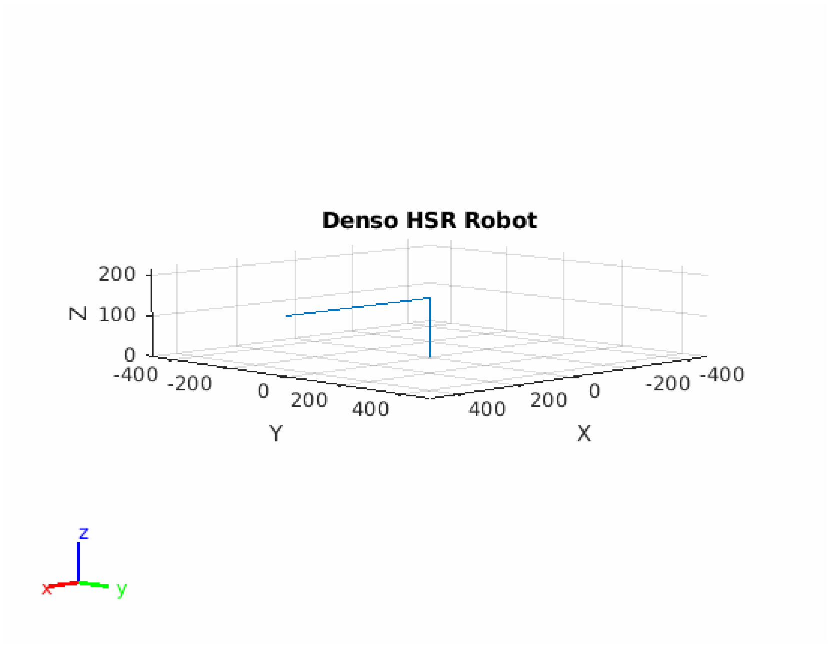


Figure 7: Denso HSR Robot Simulation

Note that we modified the above defined DH table to create extra fixed joints to maintain the "L" arm shape of the robot arm.

```
phi = 0;
theta = 0;
psi = 0;
d1 = 0;

dh_table = [    % a      alpha      d       theta
                 0       0          146     phi     ;
                 205     0          0       0       ;
                 275     0          0       theta   ;
                 0       0         -41     0       ;
                 0       pi         d1      0       ;
                 0       0          0       psi     ;
           ];
% Build our robot arm
robot = robotics.RigidBodyTree;

body1 = robotics.RigidBody("body1");
link1 = robotics.RigidBody("link1");
body2 = robotics.RigidBody("body2");
body3 = robotics.RigidBody("body3");
body4 = robotics.RigidBody("body4");
```

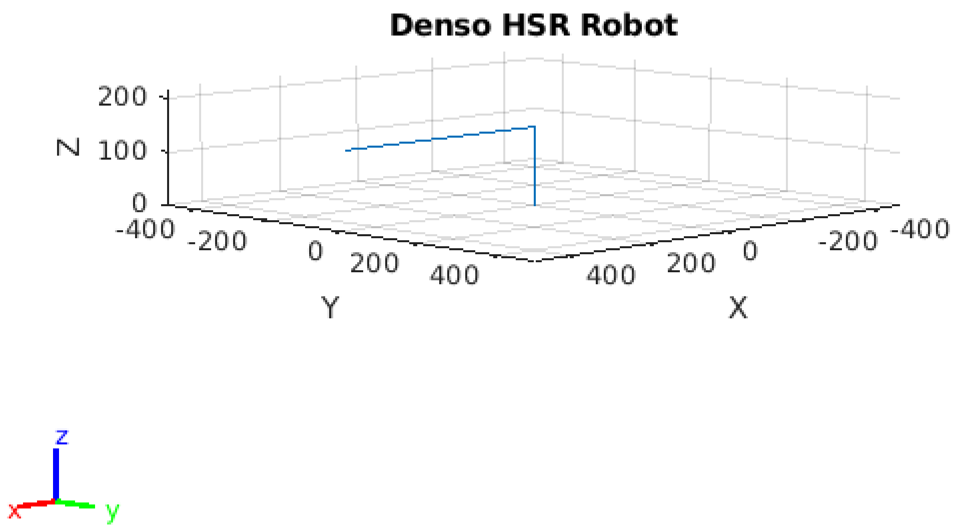
```
joint1 = robotics.Joint("joint1", "revolute");
fixed1 = robotics.Joint("fixed1", "fixed");
joint2 = robotics.Joint("joint2", "revolute");
joint3 = robotics.Joint("joint3", "prismatic");
joint4 = robotics.Joint("joint4", "revolute");


setFixedTransform(joint1, dh_table(1, :), 'dh');
setFixedTransform(fixed1, dh_table(2, :), 'dh');
setFixedTransform(joint2, dh_table(3, :), 'dh');
setFixedTransform(joint3, dh_table(5, :), 'dh');
setFixedTransform(joint4, dh_table(6, :), 'dh');

body1.Joint = joint1;
link1.Joint = fixed1;
body2.Joint = joint2;
body3.Joint = joint3;
body4.Joint = joint4;

addBody(robot, body1, "base");
addBody(robot, link1, "body1");
addBody(robot, body2, "link1");
addBody(robot, body3, "body2");
addBody(robot, body4, "body3");

showdetails(robot)
```

```
--------------------
Robot: (5 bodies)

 Idx     Body Name     Joint Name     Joint Type     Parent Name(Idx)   Children Name(s)
 ---     ---------     ----------     ----------     ----------------   ----------------
   1         body1         joint1       revolute              base(0)   link1(2)
   2         link1         fixed1          fixed             body1(1)   body2(3)
   3         body2         joint2       revolute             link1(2)   body3(4)
   4         body3         joint3      prismatic             body2(3)   body4(5)
   5         body4         joint4       revolute             body3(4)
--------------------
```

```
x = show(robot, homeConfiguration(robot))
```

```
x =
  Axes (Primary) with properties:

             XLim: [-528 528]
             YLim: [-528 528]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
```

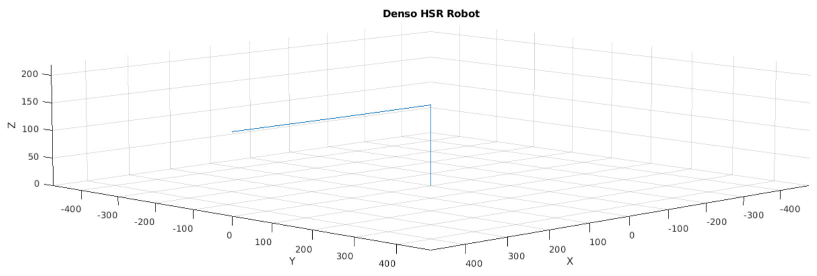
```
title("Denso HSR Robot")
axis([-480 480 -480 480 0 220]);
```

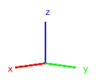


```
range = 0:0.01:1.0;
```

Here we see our robot in its default home configuration, with all joints set to 0. The image is generated by the robotics toolbox. From here, we can generate the animation by modifying the configuration by adjusting our joints to the predefined function for each and the given timestep. Please note that this often opens in an additional Figure window, and may not show the animation here.

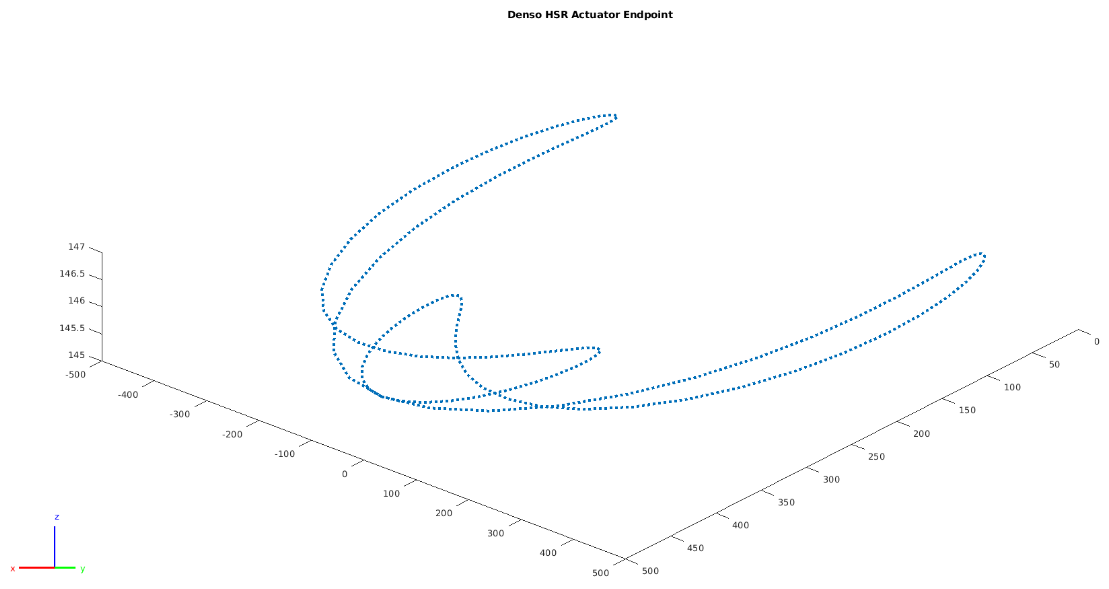
```
phis = [];
thetas = [];
Xs = [];
Ys = [];
Zs = [];

filename='denso-hsr-5-1.gif';
first = true;
jacobians = {};
configurations = {};
configuration = homeConfiguration(robot);

% Now animate
for t = range
    phi = -pi/4 * sin(2*pi*t);
    theta = pi/2 * sin(4*pi*t);
    phis(end+1) = phi;
    thetas(end+1) = theta;

    configuration(1).JointPosition = phi;
    configuration(2).JointPosition = theta;

    configurations{end+1} = configuration;

    x = show(robot, configuration);

    transform = getTransform(robot, configuration, "body3");
    actuator = transform * [0; 0; 0; 1;];
    Xs(end+1) = actuator(1);
    Ys(end+1) = actuator(2);
    Zs(end+1) = actuator(3);

    jacobians{end+1} = geometricJacobian(robot, configuration, 'body3');

    axis([-480 480 -480 480 0 220]);
    title("Denso HSR Robot");

    pause(5/length(range))

end
```





Here, we plot the actuator endpoint position by tracking the X, Y, and Z locations of the endpoint at each stage of the above simulation.

```
plot3(Xs, Ys, Zs, "LineStyle", ':', "Linewidth", 3)
title("Denso HSR Actuator Endpoint")
view([130.82 70.31])
```



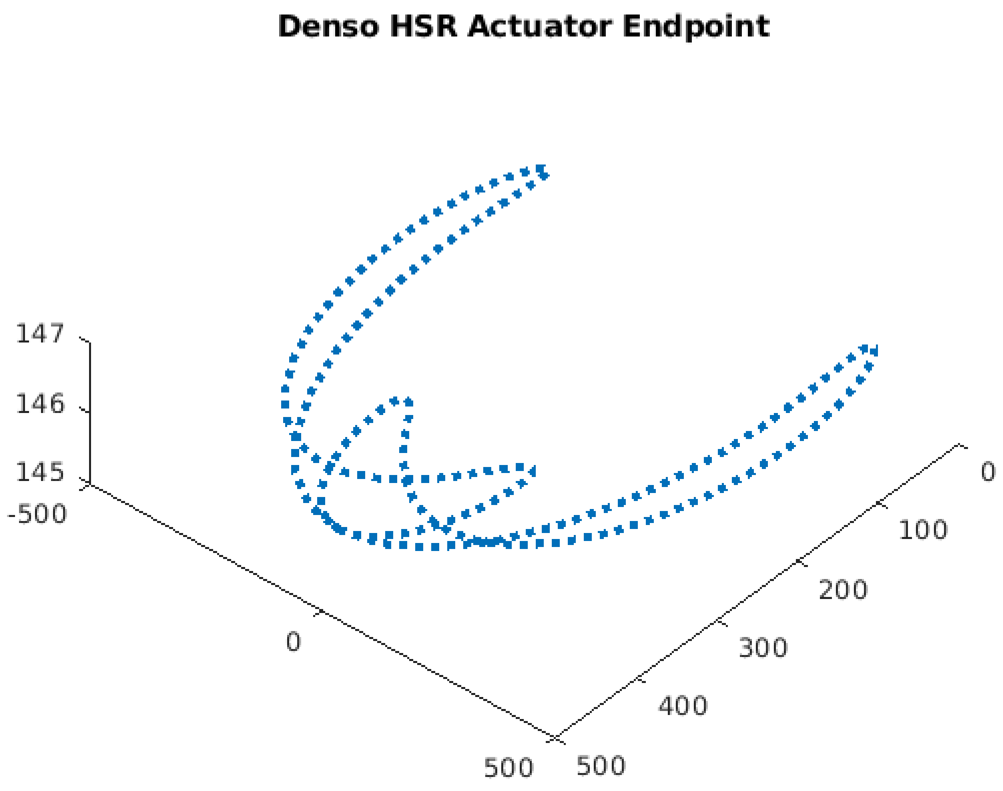


Figure 8: Endpoint Pathing through the simulation

VI-2

During the simulation, we recorded a set of phi and theta values for each timestep. Here, we are using the *trapveltraj* function (short for Trapezoidal Velocity Trajectory) to generate the estimated values for the positions and velocities for each joint.

```
[q,qd,qdd,tSamples,pp] = trapveltraj(cat(1, phis, thetas), 501);
subplot(2,1,1)
plot(tSamples, q)
title("Joint Positions")
xlabel('t')
ylabel('Positions')
legend('Phi','Theta')
subplot(2,1,2)
plot(tSamples, qd);
title("Joint Velocities")
xlabel('t')
ylabel('Velocities')
legend('Phi','Theta')
drawnow
close all
```

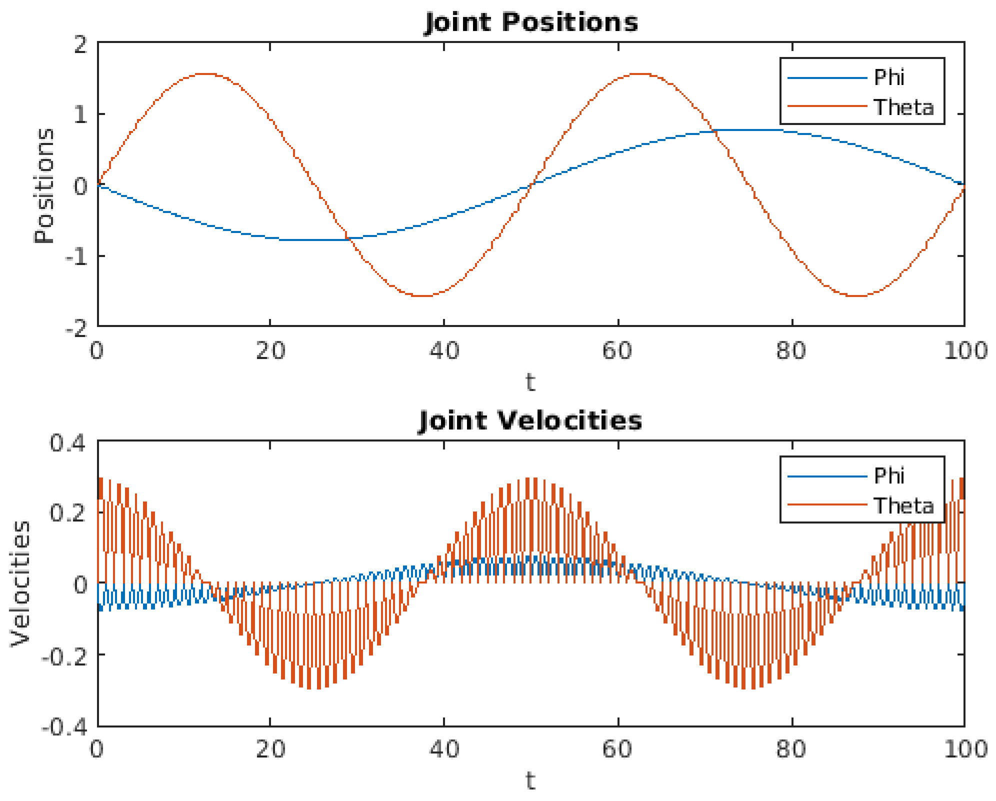


Figure 9: Positions and velocities for ϕ and θ, or joint 1 and 2, respectively

Now we calculate our Jacobian. We define a step (which you can configure) as the time step that our robot moved in the previous simulation. There are 100 possible time steps to choose from. Here we generate the Jacobian and offer the toolbox provided *geometricJacobian*'s calculated output to compare.

For simplicity, we redefined the arm without fixed joints to fix errors and possible confusions in the calculations. While this would not draw correctly in the simulation, it provides the same actuator endpoint values and is functionally equivalent of the above arm.

```
step = 15;

% Set our variables to the equivalent setup for the above robot.
phi = phis(15);
theta = thetas(15);
psi = 0;
d1 = 0;

% Redefine our robot
dh_table = [   % a      alpha     d       theta
                 205    0       146     phi    ;
                 275    pi       -41       theta  ;
                 0      0       0       d1;
                 0      0       0       psi;
           ];

robot = robotics.RigidBodyTree;

shoulder = robotics.RigidBody("shoulder");
elbow = robotics.RigidBody("elbow");
wrist = robotics.RigidBody("wrist");
actuator = robotics.RigidBody("actuator");

joint1 = robotics.Joint("joint1", "revolute");
joint2 = robotics.Joint("joint2", "revolute");
joint3 = robotics.Joint("joint3", "prismatic");
joint4 = robotics.Joint("joitn4", "revolute");

setFixedTransform(joint1, dh_table(1, :), 'dh');
setFixedTransform(joint2, dh_table(2, :), 'dh');
setFixedTransform(joint3, dh_table(3, :), 'dh');
setFixedTransform(joint4, dh_table(4, :), 'dh');

shoulder.Joint = joint1;
elbow.Joint = joint2;
wrist.Joint = joint3;
actuator.Joint = joint4;

addBody(robot, shoulder, "base");
addBody(robot, elbow, "shoulder");
addBody(robot, wrist, "elbow");
addBody(robot, actuator, "wrist");

% Set up our configuraton for the given robot's position
configuration = homeConfiguration(robot);
configuration(1).JointPosition = phi;
conifguration(2).JointPosition = theta;
configuration(3).JointPosition = d1;
configuration(4).JointPosition = psi;
```

```
% Generate the transform for each coordinate transform back to the base
% frame
H_0_1 = getTransform(robot, configuration, "shoulder")
```

```
H_0_1 = 4×4
    0.8224    0.5689         0  168.5943
   -0.5689    0.8224         0 -116.6231
         0         0    1.0000  146.0000
         0         0         0    1.0000
```

```
H_0_2 = getTransform(robot, configuration, "elbow")
```

```
H_0_2 = 4×4
    0.8224   -0.5689   -0.0000  394.7574
   -0.5689   -0.8224   -0.0000 -273.0688
         0    0.0000   -1.0000  105.0000
         0         0         0    1.0000
```

```
H_0_3 = getTransform(robot, configuration, "wrist")
```

```
H_0_3 = 4×4
    0.8224   -0.5689   -0.0000  394.7574
   -0.5689   -0.8224   -0.0000 -273.0688
         0    0.0000   -1.0000  105.0000
         0         0         0    1.0000
```

```
H_0_4 = getTransform(robot, configuration, "actuator")
```

```
H_0_4 = 4×4
    0.8224   -0.5689   -0.0000  394.7574
   -0.5689   -0.8224   -0.0000 -273.0688
         0    0.0000   -1.0000  105.0000
         0         0         0    1.0000
```

```
% We define the origin of each coordinate frame with the robot's current
% configuration, relative to the base frame's coordinate system
origin = [0; 0; 0; 1;];
o0 = origin;
o1 = H_0_1 * origin;
o2 = H_0_2 * origin;
o3 = H_0_3 * origin;
o4 = H_0_4 * origin;

% We are dropping the "1" from the homogenous points here
o0(end) = [];
o1(end) = [];
o2(end) = [];
o3(end) = [];
o4(end) = [];

% Define the z axis orientation for each coordinate axis relative to the base frame
z0 = [0; 0; 1;];
z1 = [0; 0; 1;];
z2 = [0; 0; -1;];
z3 = [0; 0; -1;];
z4 = [0; 0; -1;];
```

```
% Calculate the cross product calculations for each revolute joint
c1 = cross(z0, o4-o0);
c2 = cross(z1, o4-o1);
c3 = cross(z3, o4-o3);

disp("My calculated Jacobian:")
```

```
My calculated Jacobian:
```

```
jacobian = [
    c1(1)     c2(1)     z2(1)    c3(1);
    c1(2)     c2(2)     z2(2)    c3(2);
    c1(3)     c2(3)     z2(3)    c3(3);
    z0(1)     z1(1)     0        z4(1);
    z0(2)     z1(2)     0        z4(2);
    z0(3)     z1(3)     0        z4(3);
]
```

```
jacobian = 6×4
  273.0688  156.4457         0         0
  394.7574  226.1631         0         0
         0         0   -1.0000         0
         0         0         0         0
         0         0         0         0
    1.0000    1.0000         0   -1.0000
```

```
disp("Toolbox calculated Jacobian:")
```

```
Toolbox calculated Jacobian:
```

```
geometricJacobian(robot, configuration, "actuator")
```

```
ans = 6×4
         0    0.0000         0   -0.0000
         0    0.0000         0   -0.0000
    1.0000    1.0000         0   -1.0000
  273.0688  156.4457   -0.0000         0
  394.7574  226.1631   -0.0000         0
         0   -0.0000   -1.0000         0
```

Success! We immediately note the "flip" of the Jacobian amongst the first and second set of three rows. The standard that we followed for the Jacobian is reversed from the toolbox, apparently. For our Jacobian, the linear velocity Jacobian is the top three rows while the angular velocity Jacobian is the bottom three rows. For the toolbox, this is the opposite and worth noting when doing calculations.

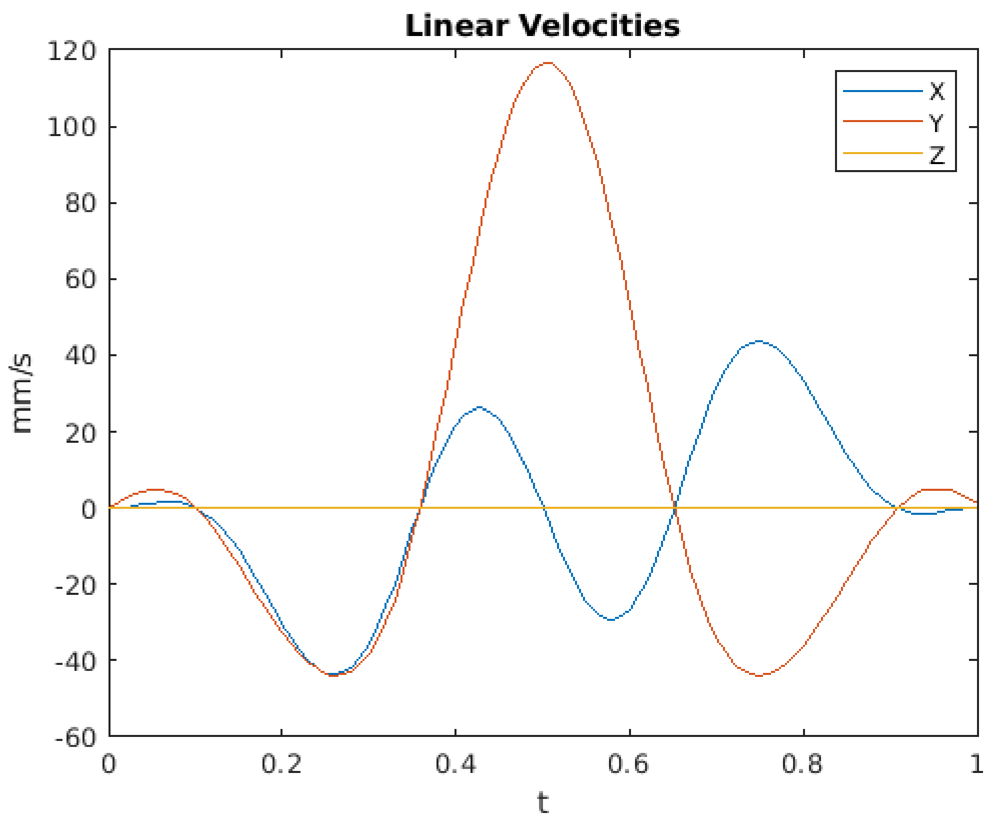
For the next step, we will now take a look at each timestep, and use the Jacobian methodolgy established above ot calculate the linear and angular velocity of the end effector at each stage of the simulation. We'll be using the *trapveltraj* function again, but using only one more sample than the current time range for the simulation. This allows us to map the Jacobian to a given timestep within the returned q_d result. Had we provided the equivalent size of the time range, the *trapveltraj* function would have returned all 0's for q_d.

We will create the sample size for *trapveltraj* and generate our q_d. Note that we expand q_d to include rows of 0's to represent the lack of change for our prismatic joint, d_1, and our actuator rotation, ψ. From here we can use our Jacobian to calculate ξ.

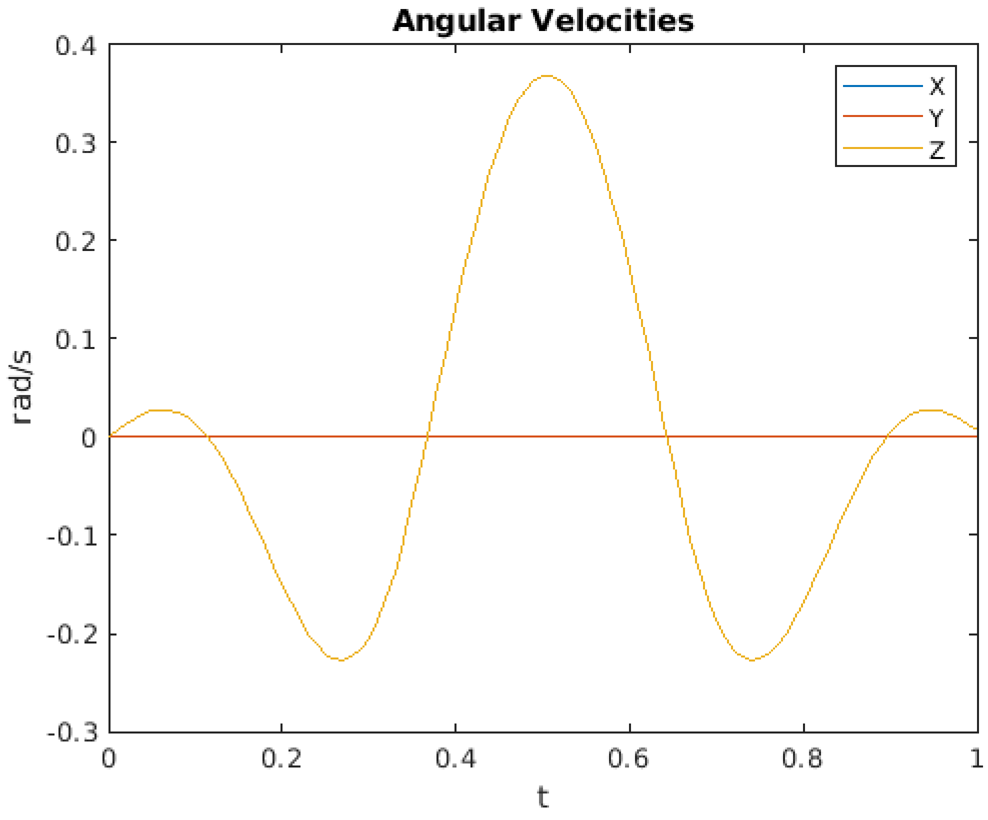
```
sample_size = length(phis) + 1;
[q,qd,qdd,tSamples,pp] = trapveltraj(cat(1, phis, thetas), sample_size);

% We expand the qd to include the two joints we didn't move - d1 and psi
expanded_qd = cat(1, qd, zeros(1, sample_size));
expanded_qd = cat(1, expanded_qd, zeros(1, sample_size));


% xi = zeros(6,sample_size-1);
xi = zeros(6, length(phis));
for index = 1:length(phis)
    configuration = homeConfiguration(robot);
    configuration(1).JointPosition = phis(index);
    conifguration(2).JointPosition = thetas(index);
    configuration(3).JointPosition = 0;
    configuration(4).JointPosition = 0;

    origin = [0; 0; 0; 1;];

    % Generate the transform for each coordinate transform back to the base
    % frame
    H_0_1 = getTransform(robot, configuration, "shoulder");
    H_0_2 = getTransform(robot, configuration, "elbow");
    H_0_3 = getTransform(robot, configuration, "wrist");
    H_0_4 = getTransform(robot, configuration, "actuator");

    % Get the origin coordinates for each frame relative to the base
    % frame's coordinate system
    o0 = origin;
    o1 = H_0_1 * origin;
    o2 = H_0_2 * origin;
    o3 = H_0_3 * origin;
    o4 = H_0_4 * origin;

    % We are dropping the "1" from the homogenous points here
    o0(end) = [];
    o1(end) = [];
    o2(end) = [];
    o3(end) = [];
    o4(end) = [];


    z0 = [0; 0; 1;];
    z1 = [0; 0; 1;];
    z2 = [0; 0; -1;];
    z3 = [0; 0; -1;];

    % Calculate the cross product calculations for each revolute joint
    c1 = cross(z0, o4-o0);
    c2 = cross(z1, o4-o1);
    c3 = cross(z3, o4-o3);

    jacobian = [
        c1(1)     c2(1)     z2(1)    c3(1);
        c1(2)     c2(2)     z2(2)    c3(2);
        c1(3)     c2(3)     z2(3)    c3(3);
        z0(1)     z1(1)     0        z3(1);
        z0(2)     z1(2)     0        z3(2);
        z0(3)     z1(3)     0        z3(3);
    ];

    xi(:,index) = jacobian * expanded_qd(:,index);
end

% Define the linear velocity as the first three rows, and angular velocity
% as the next 6 rows.
linear_velocity = xi(1:3,:);
angular_velocity = xi(4:6,:);
```

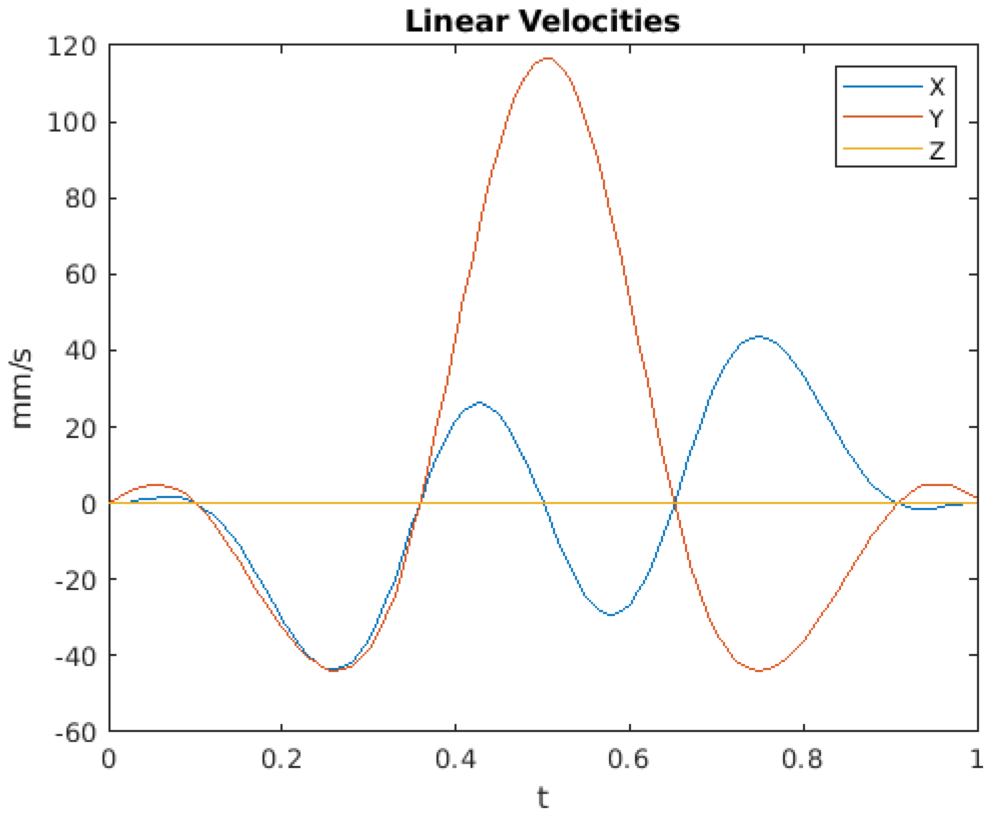
And finally we plot our results:

```
plot(range, linear_velocity(1, :))
hold on;
plot(range, linear_velocity(2, :))
plot(range, linear_velocity(3, :))
title("Linear Velocities")
xlabel('t')
ylabel('mm/s')
legend('X','Y', 'Z')
hold off;
```



```
plot(range, angular_velocity(1, :))
hold on;
plot(range, angular_velocity(2, :))
plot(range, angular_velocity(3, :))
title("Angular Velocities")
xlabel('t')
ylabel('rad/s')
legend('X','Y', 'Z')
hold off;
```


Angular Velocities
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
rad/s
0
0.2
0.4
0.6
0.8
1
t
X
Y
Z


Linear Velocities
120
100
80
60
40
20
0
-20
-40
-60
mm/s
0
0.2
0.4
0.6
0.8
1
t
X
Y
Z

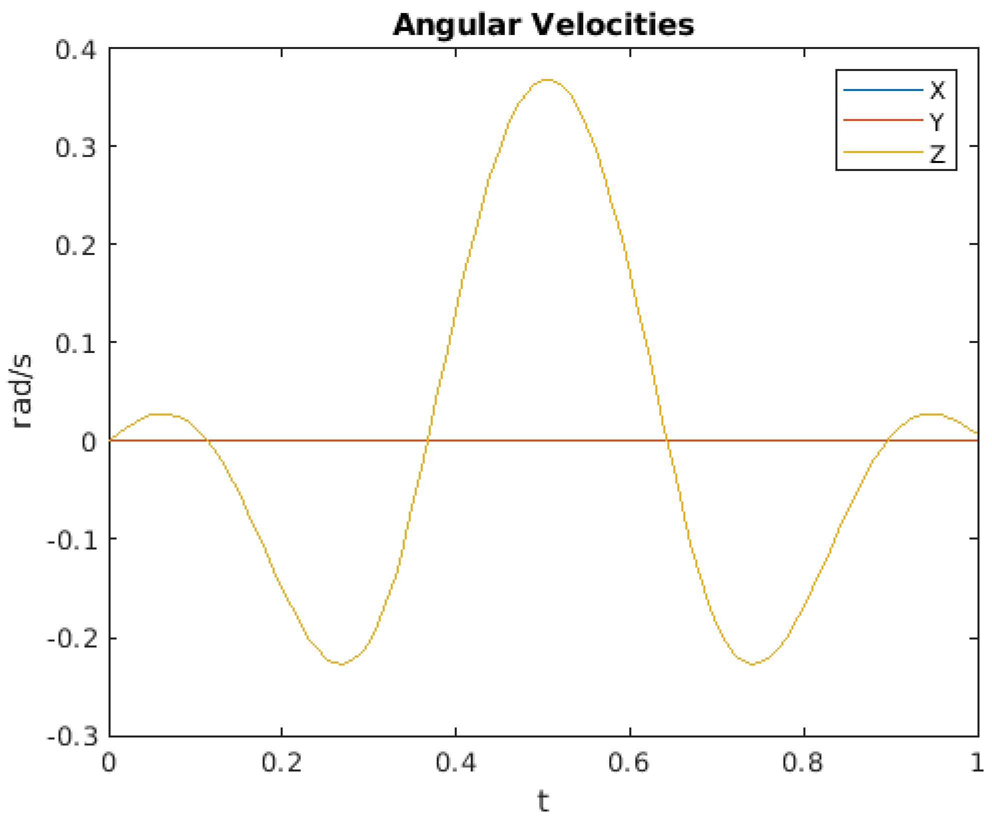


Figure 10: The linear and angular velocity of the tooling in dimensions X, Y, and Z.

Discussion

This week's assignment's purpose was to familiarize ourselves with the Mathworks' MATLAB Robotics Toolbox and to demonstate our lessons on the Jacobian and velocity kinematics of robotics arm.

To demonstrate the understanding of the robotics toolbox, we looked at a familiar robotics arm that we have worked with in past weeks, creating a simulation of the arm using the new tools. The tools demonstrated that we can generate a DH table for a given robot arm and then easily simulate it in the toolbox.

Once we had simulated the robot arm, we used values observed during the simulation to look at the velocity of each joint.

Finally, we computed the Jacobian for our arm manually. We then calculated and plotted the velocity - linear and angular - for the Denso's actuator tip.