M.E. 530.646 Final Project

Radhika Rajaram

Calculations for moving from a point in a plane to another point in another plane, where the plane is defined by the normal:

Suppose the end-effector of our robot is initially in the x-y plane at the origin and we have commands that describe the trajectory of the robot's path in the x-y plane relative to the origin, we wish to generate the same trajectory in some plane and point 3D space.

The trajectory of the end-effector is described here in homogeneous representation.

First we must rotate the trajectory in the x-y plane so that it becomes parallel to the destination plane. Then we must translate trajectory from the origin to the new destination point.

By finding out the cross product between the normals to the two planes we can find the rotation matrix that transforms the trajectory. From the cross product we calculate the twist angle and the twist axes. Then, using Rodrigues's formula, we find the rotation matrix as shown in the function:

```
function R=fRot(v1, v2)
% R*v1=v2
% v1 and v2 should be 3x1

% 1. rotation vector
w=cross(v1,v2);
w=w/norm(w);
w_hat=GetSkew(w);

% 2. rotation angle
cos_theta=v1'*v2/norm(v1)/norm(v2);
theta=acos(cos_theta);

% 3. rotation matrix, using Rodrigues' formula
R=eye(size(v1,1))+w_hat*sin(theta)+w_hat^2*(1-cos(theta));

function x_skew=GetSkew(x)
x_skew=[0 -x(3) x(2);
 x(3) 0 -x(1);
 -x(2) x(1) 0];
end
end
```

NOTE: Vector 1 is the normal to the x-y plane= $[0\ 0\ 1]^T$ (or whichever plane in which we have defined our trajectory- x-z or y-z)

Now we convert this rotation matrix into a 4x4 homogeneous form and multiply this with the points in our trajectory.

```
R=horzcat(R,[0;0;0]);
R=vertcat(R,[0,0,0,1]);
```

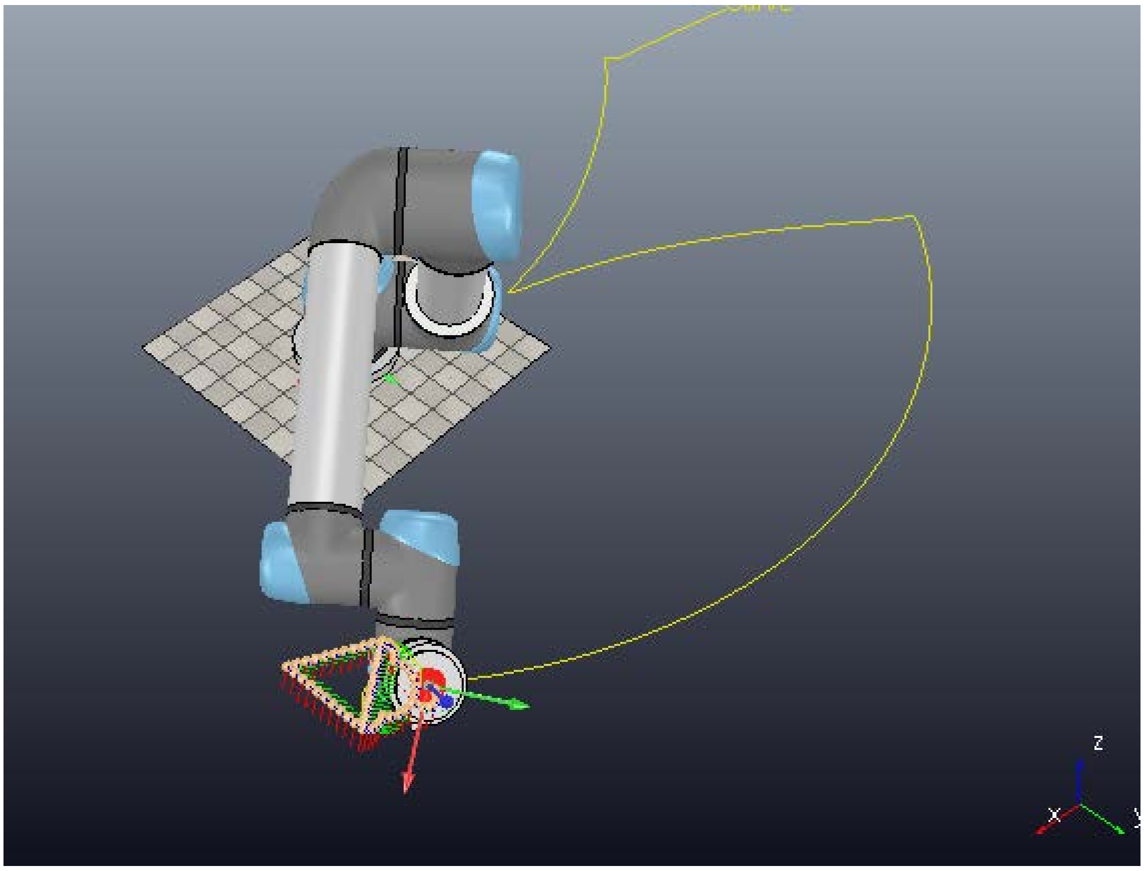
Then multiply with a 4x4 translation matrix (where the first 3x3 elements are the identity rotation matrix and the last column is the point to which we need to move to).

```
t=horzcat(eye(3,3),point);
t=vertcat(trans,[0,0,0,1]);
```

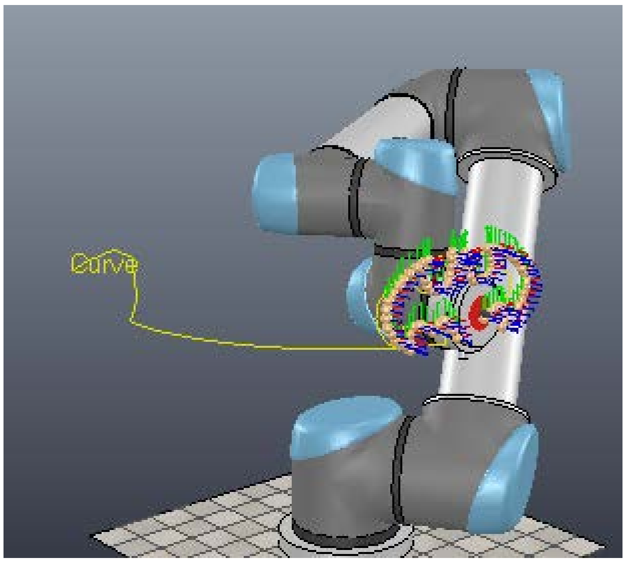
Final transformation matrix: F , Initial representation in the x-y plane: P, Desired point: D

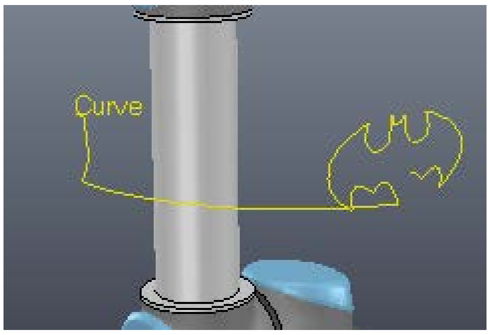
D= F*P
D =t*R*P (rotation followed by translation).

Pictures of the UR5 drawing in VREP:

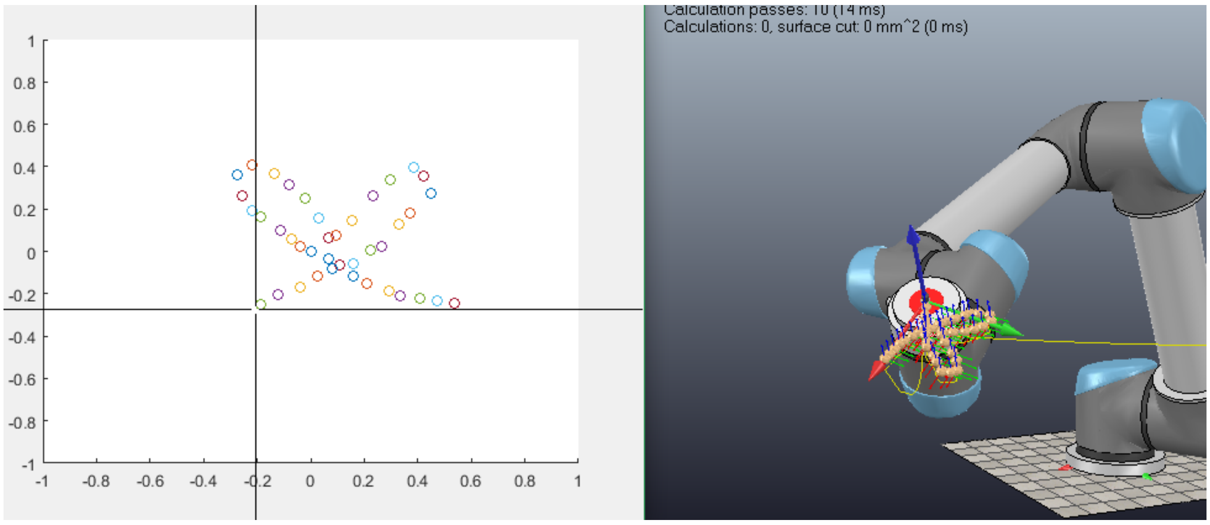


UR5 drawing an 'ice-cream cone with cherry' outline in VREP





UR5 drawing a ‘Batman logo’ outline in VREP



```
Calculation passes: 10 (14 ms)
Calculations: 0, surface cut: 0 mm^2 (0 ms)
```

UR5 Drawing based on user input into a Matlab figure window

Describing lines, circles etc:

For the purpose of generating a linear and circular trajectories, we can use parametric representation of the curve that we wish to draw:

Parametric equation of a line given two points (x_1,y_1) and (x_2,y_2):

$$x = (1 - t)x_1 + tx_2,$$
$$y = (1 - t)y_1 + ty_2$$

At $t=0$, we get the point (x_1,y_1) and at $t=1$, we get the point (x_2,y_2). So if we know two points on the line we wish to draw we can loop through the above equations by increasing the value of t.

Similarly, for a circle, given a centre point (a,b) and radius r, the parametric equation is:

$$x = a + r\cos(t)$$
$$y = b + r\sin(t)$$

By changing the value of t from 0 to 2Π, we can generate a circle.

Other geometries like ellipses, parabolas etc. can be described in a similar manner.

Links to the Youtube videos:

Inverse kinematics with user input

https://youtu.be/6UwqBVUG6N4

Differential Kinematics:

https://youtu.be/QMU7OVCZjvw

Both videos are listed as Private, so they are not searchable.

Thank you!