Robotics HW5

Yunfan Gao

October 2020

Problem 1

```
close all; clear all;
ex=[1;0;0];ey=[0;1;0];ez=[0;0;1];zv=[0;0;0];
Box=collisionCylinder(2,0)
square=collisionBox(1,1,0)
square.Pose=trvec2tform([1/sqrt(2) 0 0 ])*[rotz(pi/4) [0;0;0];[0
    ↪ 0 0 1]];
figure(1);show(Box);hold on
view(0,90)
axis(4*[-1 1 -1 1]);axis('square');
N=66; %not to choose the multiple of 4 to avoid naf in dwp
q=(0:2*pi/N:2*pi);
for i=1:numel(q)
    square.Pose=trvec2tform((rotz(q(i))*[0;-2-sqrt(2)/2;0])')*...
    [rotz(pi/4) [0;0;0];[0 0 0 1]];
    [isInt,dist,wp]=checkCollision(square,Box);
    dwp=wp(:,2)-wp(:,1);
    square.Pose=trvec2tform(dwp')*square.Pose;
    pA(:,i)=square.Pose(1:2,4)-[1/sqrt(2);0];
    show(square);
end
plot(pA(1,:),pA(2,:),'kx-','linewidth',3)
pA(:,1:length(pA)-1)
```

Sketch the free space in the robot configuration space with matlab:

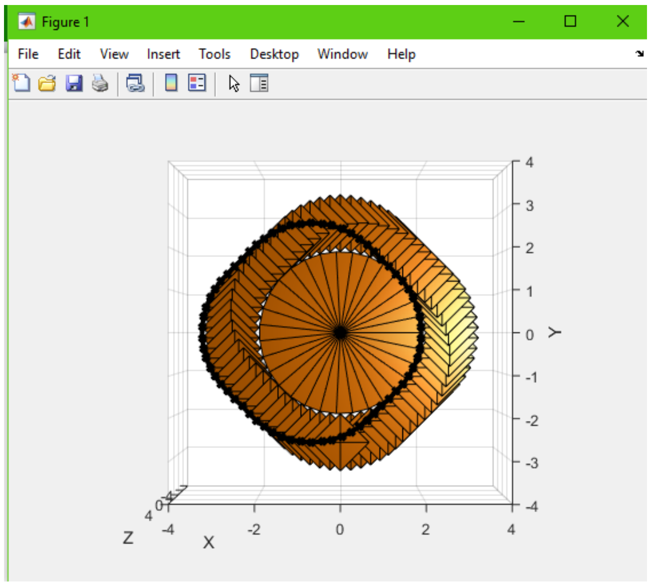
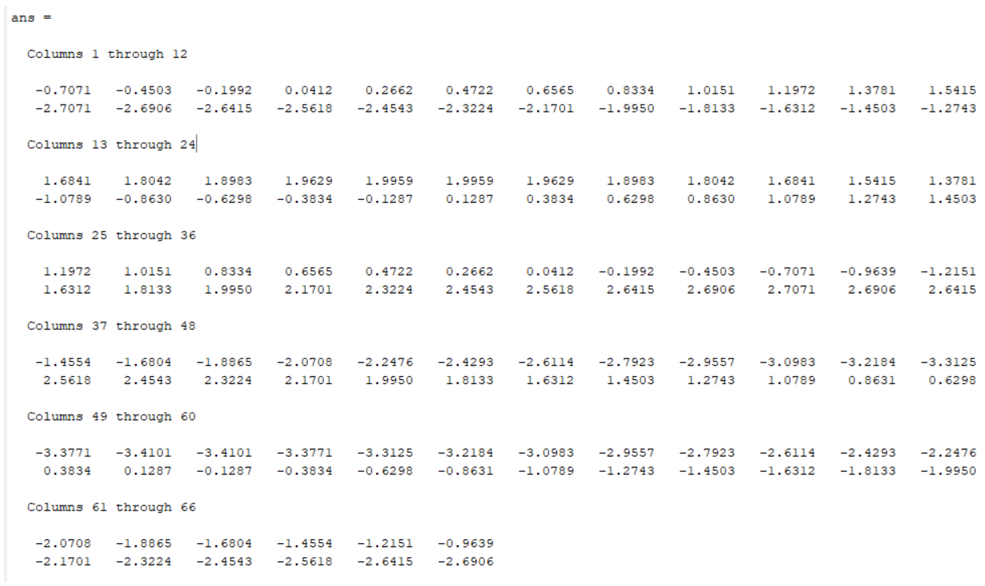


Figure 1: point A free space configuration when N = 66

The numerically generated the precise free space description of point A print when N = 66:



```
ans =

  Columns 1 through 12

   -0.7071   -0.4503   -0.1992    0.0412    0.2662    0.4722    0.6565    0.8334    1.0151    1.1972    1.3781    1.5415
   -2.7071   -2.6906   -2.6415   -2.5618   -2.4543   -2.3224   -2.1701   -1.9950   -1.8133   -1.6312   -1.4503   -1.2743

  Columns 13 through 24

    1.6841    1.8042    1.8983    1.9629    1.9959    1.9959    1.9629    1.8983    1.8042    1.6841    1.5415    1.3781
   -1.0789   -0.8630   -0.6298   -0.3834   -0.1287    0.1287    0.3834    0.6298    0.8630    1.0789    1.2743    1.4503

  Columns 25 through 36

    1.1972    1.0151    0.8334    0.6565    0.4722    0.2662    0.0412   -0.1992   -0.4503   -0.7071   -0.9639   -1.2151
    1.6312    1.8133    1.9950    2.1701    2.3224    2.4543    2.5618    2.6415    2.6906    2.7071    2.6906    2.6415

  Columns 37 through 48

   -1.4554   -1.6804   -1.8865   -2.0708   -2.2476   -2.4293   -2.6114   -2.7923   -2.9557   -3.0983   -3.2184   -3.3125
    2.5618    2.4543    2.3224    2.1701    1.9950    1.8133    1.6312    1.4503    1.2743    1.0789    0.8631    0.6298

  Columns 49 through 60

   -3.3771   -3.4101   -3.4101   -3.3771   -3.3125   -3.2184   -3.0983   -2.9557   -2.7923   -2.6114   -2.4293   -2.2476
    0.3834    0.1287   -0.1287   -0.3834   -0.6298   -0.8631   -1.0789   -1.2743   -1.4503   -1.6312   -1.8133   -1.9950

  Columns 61 through 66

   -2.0708   -1.8865   -1.6804   -1.4554   -1.2151   -0.9639
   -2.1701   -2.3224   -2.4543   -2.5618   -2.6415   -2.6906
```

Figure 2: numerically generated points of A

Problem 2

```
%
close all; clear all;
% 8.2 example
%
% initialization
ex=[1;0;0];ey=[0;1;0];ez=[0;0;1];zv=[0;0;0];

% define robot using PoE convention
l1=1;l2=1;
h1=ez;h2=ez;
p01=0*ex+0*ey;p12=l1*ex;p2T=l2*ex;
type = [0 0];
H=[h1 h2];
P=[p01 p12 p2T];
twolink.H=H;
twolink.P=P;
twolink.joint_type=type; % planar 2R robot
n=2;

robdef=twolink;radius=.05;

% define collision body

[rob,colLink]=collisionBody(robdef,radius);

obs=collisionCylinder(0.3,0);
obs.Pose=trvec2tform([1.2 1.2 0 ])*[rotz(pi/4) [0;0;0];[0 0 0
    ↪ 1]];

% show initial configuration

qf=[pi/2;0];
q0=[0;0];
figure(1);view(0,90);
show(rob,q0,'Collisions','on','Visuals','off');
hold on
show(rob,qf,'Collisions','on','Visuals','off');
show(obs)
view(0,90);
hold off
% check collision for a grid of (q1,q2)

ngrid=40;qlim=3*pi/2;
```

```
q1min=-qlim;q1max=qlim;dq1=(q1max-q1min)/ngrid;
q2min=-qlim;q2max=qlim;dq2=(q2max-q2min)/ngrid;

q1=(q1min:dq1:q1max);
q2=(q2min:dq2:q2max);

nq1=numel(q1);
nq2=numel(q2);

colgrid=zeros(nq1,nq2);
fignum=10;
for i=1:nq1
    for j=1:nq2
        % [isInt,dist,wp]=collisionPlot(rob,robdef,...
        % [q1(i);q2(j)],colLink,obs,fignum);
       [isInt,dist,wp]=collisionCheck(robdef,[q1(i);q2(j)],colLink
           ↪ ,obs);

        view(90,90);
        colgrid(i,j)=max(cell2mat(isInt));
    end
end

waypoint=[-0.4;2.7];
%waypoint=[-.5;2.5];

[ind1,ind2]=find(colgrid>0);
figure(11);
plot(q1(ind1),q2(ind2),'x',[q0(1),qf(1)],[q0(2),qf(2)],...
     'o','linewidth',3);
hold on;
plot([q0(1) waypoint(1) qf(1)],[q0(2) waypoint(2) qf(2)],'r-');
grid;
axis([q1min,q1max,q2min,q2max]);
axis('square');
hold off;

% use final time from S-curve
ldotmax=1;lddotmax=2.5;
[l,ld,ldd,ta,tb,tf]=...
    scurvegen(0,1,0,0,ldotmax,lddotmax,lddotmax,.2,.2,.2,.2,0);
t=(0:.1:tf);

% choosing trajectory generation scheme
trajgenmethod=3;
```

```
if trajgenmethod == 2
    % constant velocity
    lambda=t/tf;
elseif trajgenmethod == 2
    % trapezoidal
    for i=1:numel(t)
        [lambda(i),ld,ldd,ta,tb,tf]=...
            trapgen(0,1,0,0,ldotmax,lddotmax,lddotmax,t(i));
    end
elseif trajgenmethod == 3
    % s-curve
    for i=1:numel(t)
        [lambda(i),ld,ldd,ta,tb,tf]=...
            scurvegen(0,1,0,0,ldotmax,lddotmax,lddotmax
                ↪ ,.2,.2,.2,.2,t(i));
    end
else
    lambda=t; % default
end

q_1=q0*(1-lambda)+waypoint*lambda;
q_2=waypoint*(1-lambda)+qf*lambda;

% plot out the joint angles
figure(3);
plot(t,q_1,t+t(numel(t)),q_2,'linewidth',2)
legend('q_{1_{first segment}}','q_{2_{firstsegment}}',...
    'q_{1_{second segment}}','q_{2_{second segment}}');

% save motion in a moview

figure(4);
for i=1:numel(t);
    show(obs);axis([-1.5,2,-1.5,2,-1.5,1.5]);grid;view(0,90);hold
        ↪ on
    show(rob,q_1(:,i),'Collisions','on');view(0,90);grid;
    drawnow;M(i)=getframe;
    %hold off
end
for i=1:numel(t)
    show(obs);axis([-1.5,2,-1.5,2,-1.5,1.5]);grid;view(0,90);hold
        ↪ on
    show(rob,q_2(:,i),'Collisions','on');view(0,90);grid;
    drawnow;M(numel(t)+i)=getframe;
    %hold off
end
```

```
movie(M)
```

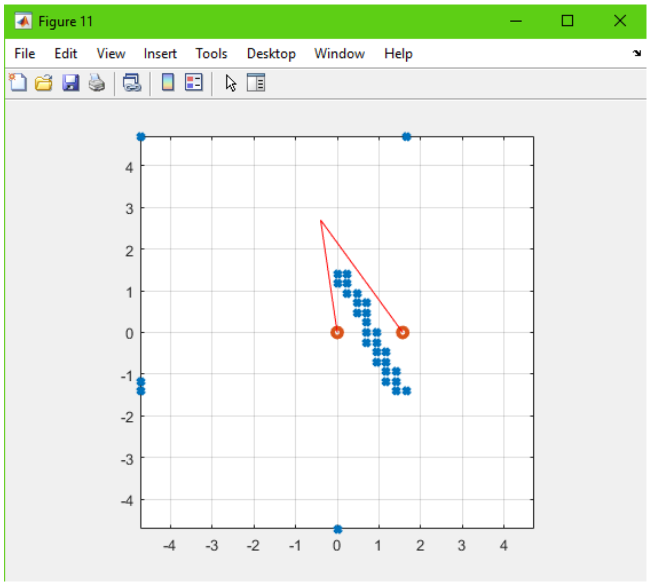


Figure 3: q1 and q2 free space and obstacle dots

b)

By choosing a break point at waypoint=[-0.4; 2.7]; and qf=[pi/2;0]; q0=[0;0]; The robot motion shows as:

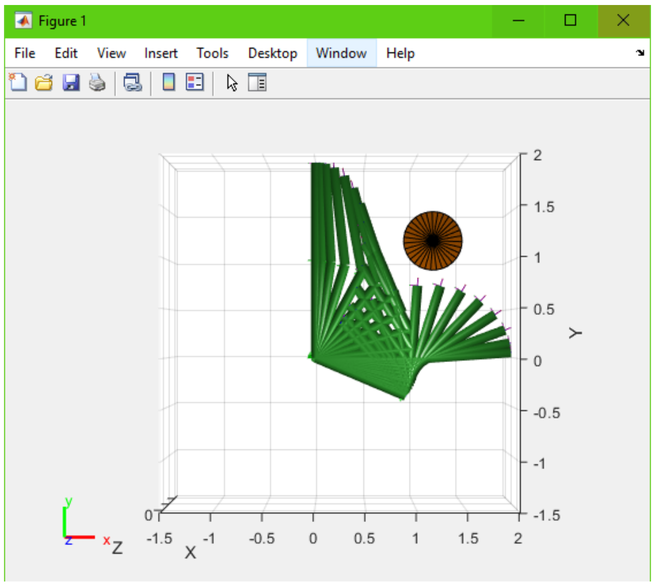


Figure 4: Robot motion in (x,y) space

c)

```
%
% 8.2.c example
%

% try a spherical obstacle to get close a good path

ans=input('Use a surrogate obstacle first (y/n)? ','s');
if ans=='y'
    obs1=collisionSphere(.14); obs1.Pose=obs.Pose;
else
    obs1=obs;
end

% find initial pose
[T0,J0]=fwddiffkinrec(1,eye(4,4),zeros(6,n),q0,robdef);
p0=T0(1:2,4);

% find final pose
[Td,Jd]=fwddiffkinrec(1,eye(4,4),zeros(6,n),qf,robdef);
pd=Td(1:2,4);

% # of iterations
```

```
N=500;

% set up storage space for the path
q=zeros(n,N+1);q(:,1)=q0;
p=zeros(2,N+1);

% update parameters for damped least square
alpha=.2;epsilon=.05;

% parameters for the control barrier function
% lower eta when it gets close
% for radius up to .14, use eta=.2, then decrease eta to .1
% then further lower eta to .05 for the true obstacle
e=.01;eta=0.2;c=100;M=100;

% maximum qdot in each step
qdmx=.1;
qdotmax=qdmx*ones(2,1);
qdotmin=-qdmx*ones(2,1);

% show initial pose
figure(21);show(rob,q0,'Collisions','on','Visuals','off');hold on
    ↪ ;
title('QP based controller');

% path planning loop using the local control barrier function

for i=1:N

    % show progression of the path in the work space
    if mod(i,2)==1;
        show(rob,q(:,i),'Collisions','on','Visuals','off');
    end

    % find current pose
    [Tq,Jq]=fwddiffkinrec(1,eye(4,4),zeros(6,n),q(:,i),robdef);

    % current tip position and Jacobian
    p(:,i)=Tq(1:2,4);J=Jq(4:5,1:2);

    % use the previous path as a starting point to follow the
        ↪ previous path
    % as much as possible
    % if no previous path, then use pd1=pd
    if exist('qprev');
        [Tq1,Jq1]=fwddiffkinrec(1,eye(4,4),zeros(6,n),qprev(:,i),
            ↪ robdef);
        pd1=Tq1(1:2,4);
    else
        pd1=pd;
    end
    deltap=p(:,i)-pd1;
    % velocity constraint (not used for QP controller)
    Aqdot=[eye(n,n);-eye(n,n)];bqdot=[qdotmax;-qdotmin];
    %
    % impose collision constraint
    %
    % check collision to true obstacle
    [isInt,dist,wp]=collisionCheck(robdef,q(:,i),colLink,obs);
    isInt_true(i)=max(cell2mat(isInt));
    d_true(i)=min(cell2mat(dist));
    % check collision to intermediate obstacle
    [isInt,dist,wp]=collisionCheck(robdef,q(:,i),colLink,obs1);
    % stop if even intermediate obstacle doesn't work
    if(max(cell2mat(isInt))>0);break;end
    coldis{i}=dist;
    [d,ind]=min(cell2mat(dist));
    % check the minimum distance d between robot and obstacle
    % if d is within a specified threshold eta, then generate
        ↪ inquality
    % constraint
    if d<eta
        % find the distance vector
        wpvec=wp{ind}(1:2,1)-wp{ind}(1:2,2);
        % inequality constraint is the square of the distance
        hI=d^2;
        % if the collision is with the third link
        % then find the Jacobian to the minimum distance point on
            ↪ the link
        if ind==3;
            % this is the distance from O_2
            d1=norm(wp{ind}(1,1)-l1*cos(q(1,i)));
            % set up Jacobian from qdot to the velocity at the
                ↪ witness pt
            s1=sin(q(1,i));c1=cos(q(1,i));
            s12=sin(q(1,i)+q(2,i));c12=cos(q(1,i)+q(2,i));
            J_d=[-l1*s1-d1*s12 -d1*s12;l1*c1+d1*c12 d1*c12];
            % this is from d h_I/dt = wpvec'*Jd*qdot > sigma(h_I)
            Acol=-wpvec'*J_d;bcol=-sigmafun(hI,eta,c,M,e);
            A=Acol;b=bcol;
            hIdot(:,i)=-Acol';
        elseif ind==2;
            % this is the disance from O_1
            d1=norm(wp{ind}(1,1));
            % set up Jacobian from qdot to the velocity at the
                ↪ witness pt
            s1=sin(q(1,i));c1=cos(q(1,i));
            s12=sin(q(1,i)+q(2,i));c12=cos(q(1,i)+q(2,i));
            J_d=[-d1*s1 0 ; d1*c1 0];
            % this is from d h_I/dt = wpvec'*Jd*qdot > sigma(h_I)
            Acol=-wpvec'*J_d;bcol=-sigmafun(hI,eta,c,M,e);
            A=Acol;b=bcol;
            hIdot(:,i)=-Acol';
        end
    else
        A=Aqdot;b=bqdot;
    end
    % Kinematic controller (different choices)
    dq_dLS=-alpha*J'*inv(epsilon*eye(n,n)+J*J')*deltap;
    dq_inv=-alpha*pinv(J)*deltap;
    dq_grad=-alpha*J'*deltap;
    % QP controller
    opt = optimset('display','off');
    dq_QP=quadprog(J'*J,alpha*J'*deltap,A,b,[],[],[],[],q(:,i),opt
        ↪ );
    % avoid large qdot
    if norm(dq_QP)> norm(qdotmax);
        dq_QP=dq_QP/norm(dq_QP)*norm(qdotmax);
    end
    dq=dq_QP;
    q(:,i+1)=q(:,i)+dq;
end

show(rob,q(:,N+1),'Collisions','on','Visuals','off');
show(obs);
axis([-1,3,-1,3]);
axis('square');
view(0,90);
hold off;

figure(20);plot((0:N),q,'x');legend('q_1','q_2');
figure(30);plot((0:N),p,'x');legend('p_x','p_y');

figure(11);
plot(q1(ind1),q2(ind2),'x',[q0(1),qf(1)],[q0(2),qf(2)],...
     'o','linewidth',3);
hold on;
```

```
plot(q(1,:),q(2,:),'r.','linewidth',2);
hold off

% # of constraint violation point
disp('# of constraint violation')
disp(sum(isInt_true>0))

% check final end point
disp('pd vs. p(:,N)');
disp([pd p(:,N)]);

% ask if want to save for next run (only save if it has converged
    ↪ )
ans=input('want to save current run (y/n)? ','s');
if (ans=='y');
    qprev=q;
    %save qprev1 qprev e eta c M qdmx alpha epsilon
end
```

Use a surrogate obstacle first, there are 9 nine constraints violations:

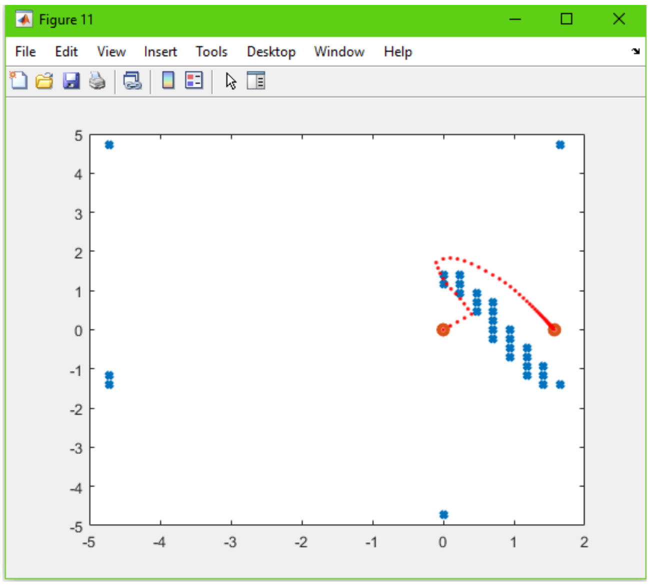


Figure 5: 9 constraints violation

Then we save the result and then save the result and run it again:

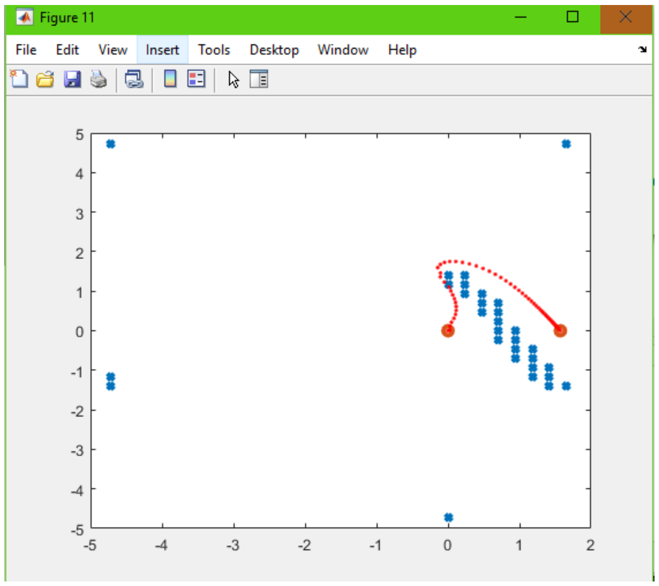


Figure 6: no constraint violation

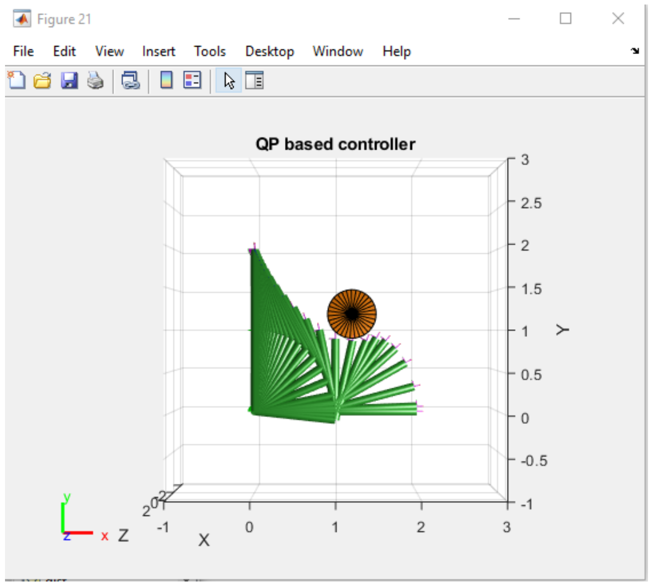


Figure 7: robot motion in x y space

Save the result and run it again, it is getting better:

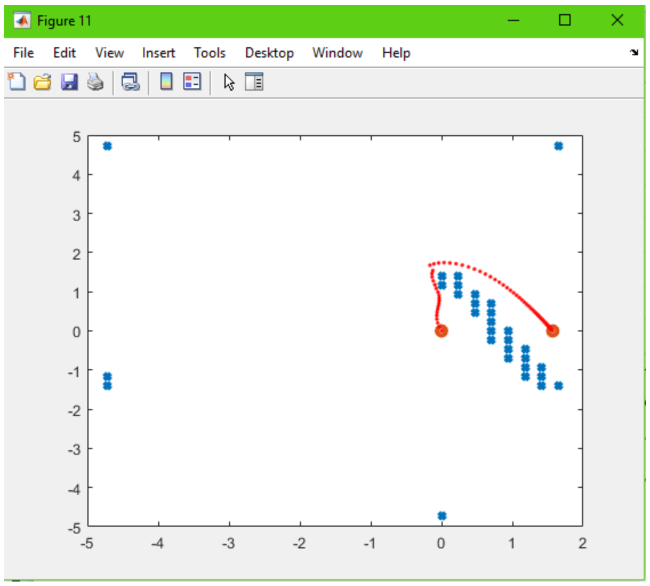


Figure 8: previous result saved and run again

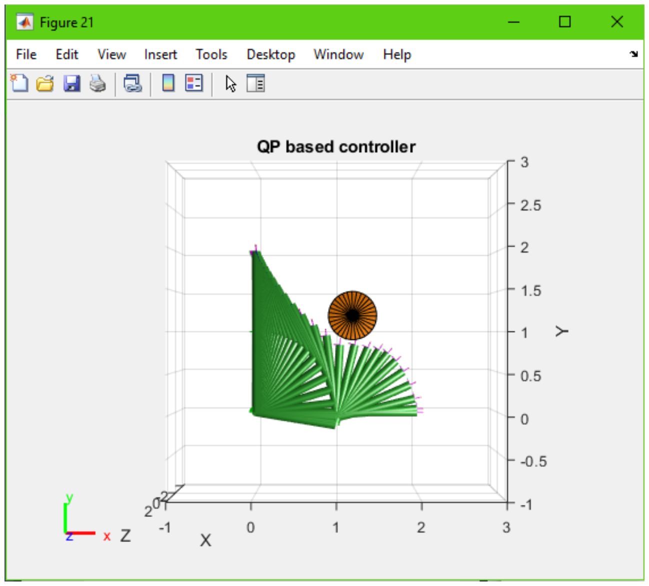


Figure 9: robot motion in x y space

The result is saved, and the program runs again with surrogate obstacle, and then that is also saved. Then the program runs with true obstacle, the result shows:

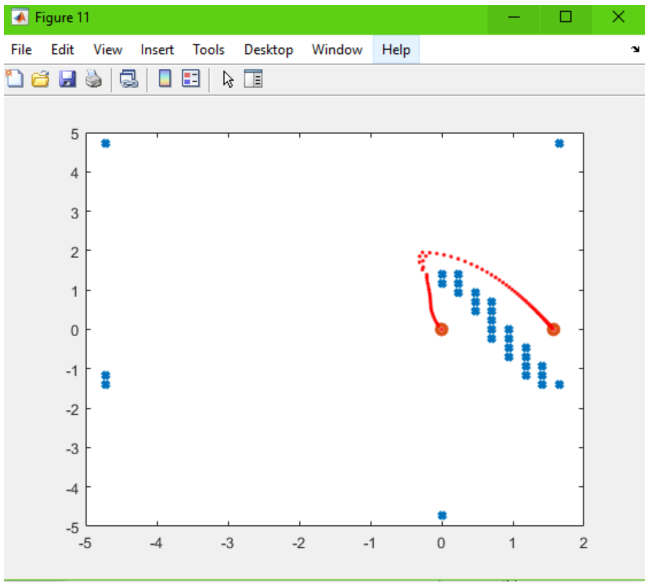


Figure 10: under the true obstacle

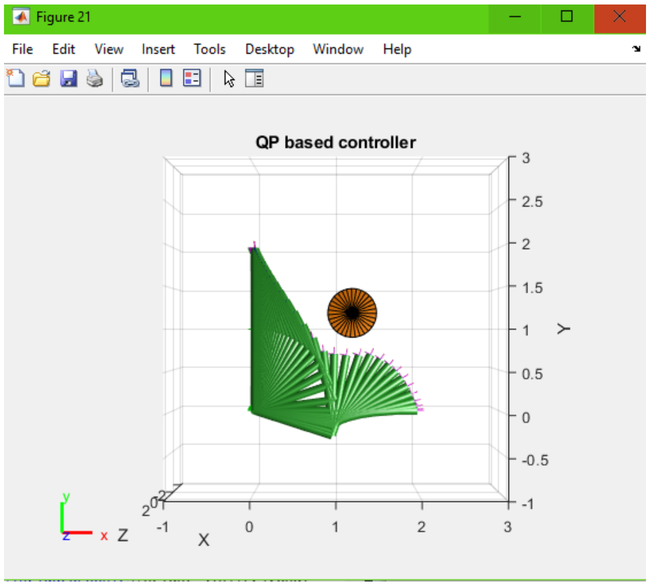


Figure 11: robot motion in x y space

Problem 3

Among the 8 possible solution, looking for the q that is closet to the q0 in terms of the Euclidean norm:

```
Td=[0 0 1 .500;0 1 0 .25;-1 0 0 .35; 0 0 0 1];
abb120def;n=size(abb120.H,2);robdef=abb120;radius=.05;
qd=invelbow(robdef,Td)
[minq,ind]=min(vecnorm(qd-zeros(6,1)))
```

which gives: qd =

$$\begin{matrix} -2.6779 & -2.6779 & 0.4636 & 0.4636 & -2.6779 & -2.6779 & 0.4636 & 0.4636 \\ -1.1896 & -1.4882 & 1.4882 & 1.1896 & -1.1896 & -1.4882 & 1.4882 & 1.1896 \\ 4.6609 & -1.0638 & 4.6609 & -1.0638 & 4.6609 & -1.0638 & 4.6609 & -1.0638 \\ 3.1416 & 3.1416 & 0 & 0 & 0 & 0 & 3.1416 & 3.1416 \\ 1.9005 & 2.1604 & 1.7048 & 1.4450 & -1.9005 & -2.1604 & -1.7048 & -1.4450 \\ 0.4636 & 0.4636 & 0.4636 & 0.4636 & -2.6779 & -2.6779 & -2.6779 & -2.6779 \end{matrix}$$

$\min q = 2.2505$
index = 4
Since q0 = 0

$$q(\lambda) = (1-\lambda)q_0 + \lambda q_f = \lambda \begin{bmatrix} 0.4636 \\ 1.1896 \\ -1.0638 \\ 0 \\ 1.4450 \\ 0.4636 \end{bmatrix}, \text{ where } \lambda \in [0,1]$$

b)

Position: $p(\lambda) = p_0(1-\lambda) + p_f\lambda$
Orientation: $R(\lambda) = R(k, -\theta(\lambda))R_0$
At zero configuration, the exponential product homogeneous transformation matrix $T(0) = \begin{bmatrix} 1 & 0 & 0 & 0.374 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.630 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and $T^{(f_1)} = \begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 0 & 1 & 0 & 0.25 \\ -1 & 0 & 0 & 0.35 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Thus

$$p(\lambda) = p_0(1-\lambda) + p_f\lambda = (1-\lambda)\begin{bmatrix} 0.374 \\ 0 \\ 0.630 \end{bmatrix} + \lambda \begin{bmatrix} 0.5 \\ 0.25 \\ 0.35 \end{bmatrix} = \begin{bmatrix} 0.126 \cdot \lambda + 0.374 \\ 0.25 \cdot \lambda \\ 0.63 - 0.28 \cdot \lambda \end{bmatrix}$$

Since θ_f is defined from $R_0 R_f^T = \mathrm{R}\,(k, \theta_f)$

$$R_0 R_f^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

From R2kth.m, we can find k = [0; -1; 0] and $\theta = \frac{\pi}{2}$
Thus $R(\lambda) = \mathrm{R}\,(k, -\lambda\theta_f) = R(-ey, -\lambda\frac{\pi}{2})$