Control of a 2-Link Arm

Lecture #11

ECE 761: Robotics

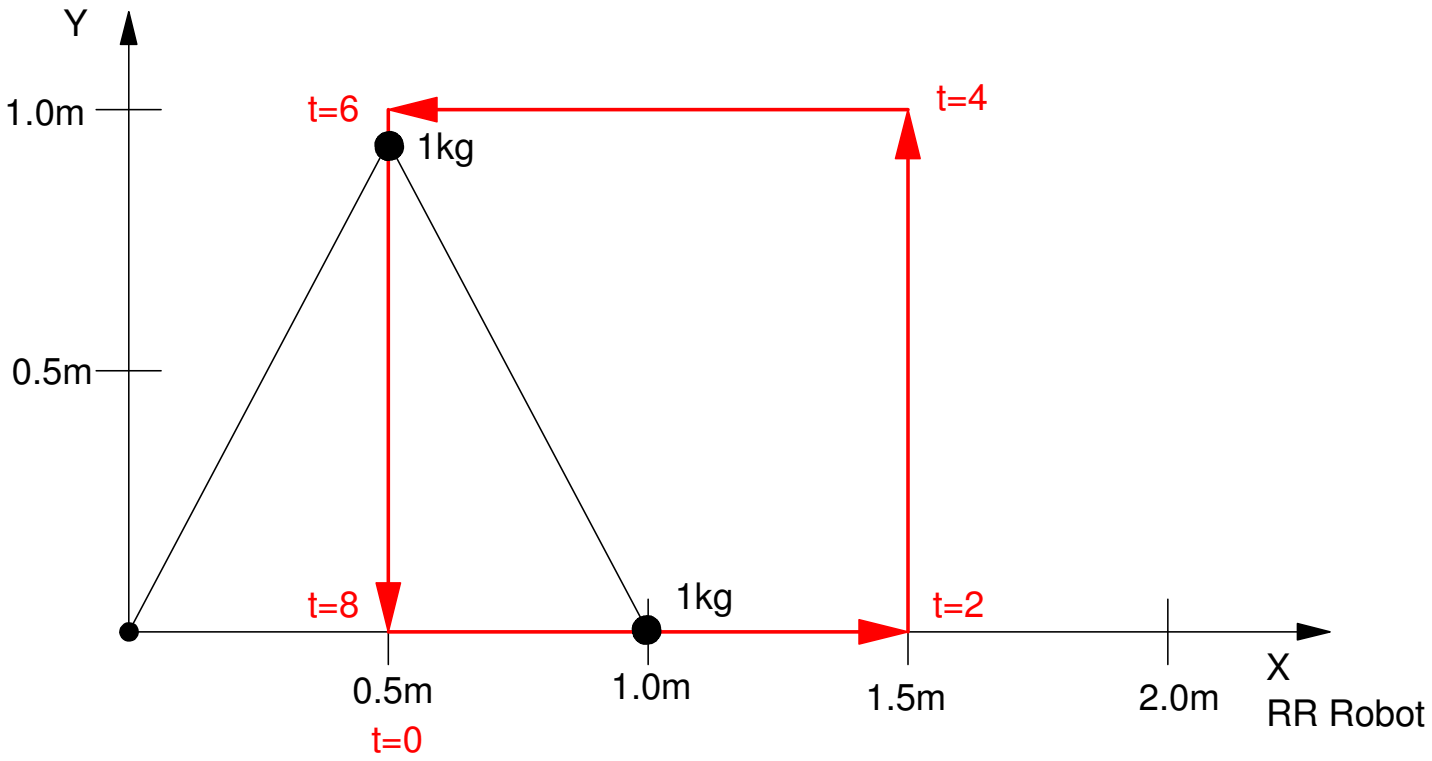
Class taught at North Dakota State University
Department of Electrical and Computer Engineering

Please visit www.BisonAcademy.com for corresponding lecture notes,
homework sets, and solutions.

Control of a RR Robot

Problem:

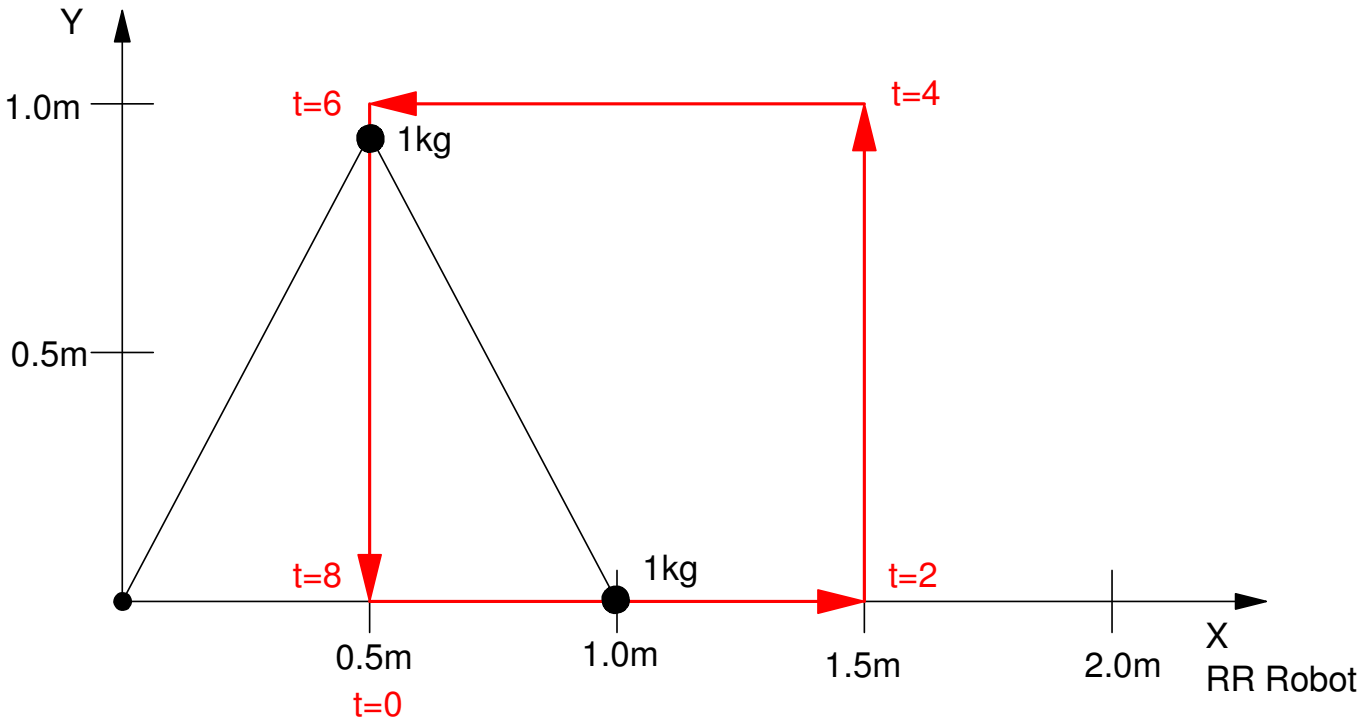
- Controlling the tip-position of a 2-link robotic arm.
- Assume it is to trace out a square in 8 seconds:



2-Link Dynamics

From before, the dynamics of the robotic arm are:

$$\begin{bmatrix} (3+2c_2) & (1+c_2) \\ (1+c_2) & 1 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} + \begin{bmatrix} 2s_2\dot{\theta}_1\dot{\theta}_2 + s_2\dot{\theta}_2^2 \\ -s_2\dot{\theta}_1^2 \end{bmatrix} - g \begin{bmatrix} 3c_1 + c_{12} \\ c_{12} \end{bmatrix}$$



To control the angle of each motor, you need to

- Define the desired angle at any given time (the set-point), and
- Determine the torque required to drive the motor to that angle.

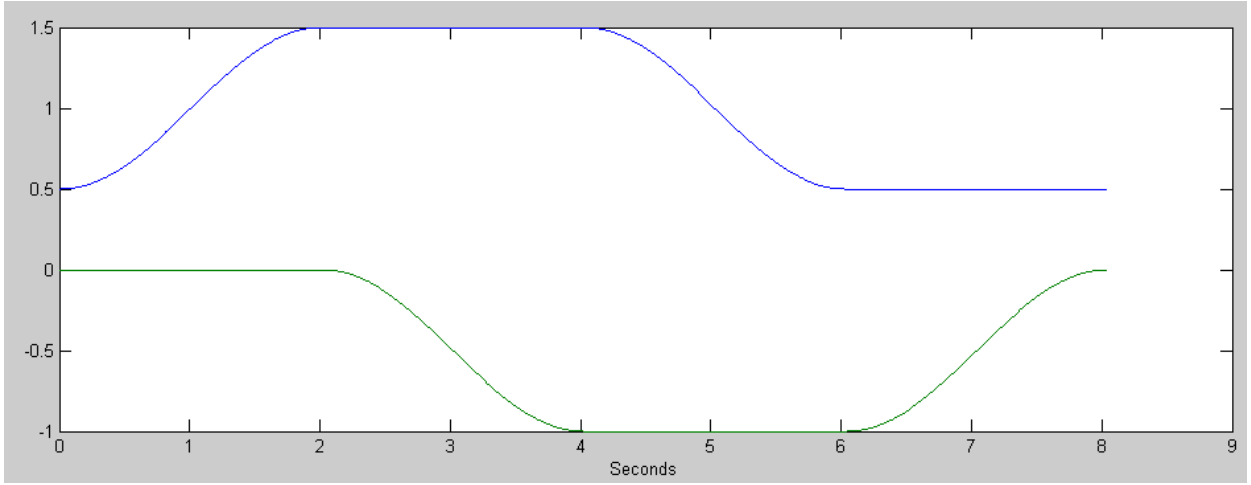
First, let's use the previous path-planning routines for the RRR robot to define the desired

- Tip positions, and
- Joint angles

Path Planning:

First, define the tip positions

```
disp('Defining Path to Follow');
P1 = [0.5, 0]';
P2 = [1.5, 0]';
P3 = [1.5, 1]';
P4 = [0.5, 1]';
P5 = P1;

disp('Calculating tip positions');
% Determine the tip positions
every 10ms
[A,T1] = MoveTo(P1,P2,2);
[A,T2] = MoveTo(P2,P3,2);
[A,T3] = MoveTo(P3,P4,2);
[A,T4] = MoveTo(P4,P5,2);
TIP = [T1,T2,T3,T4];
```



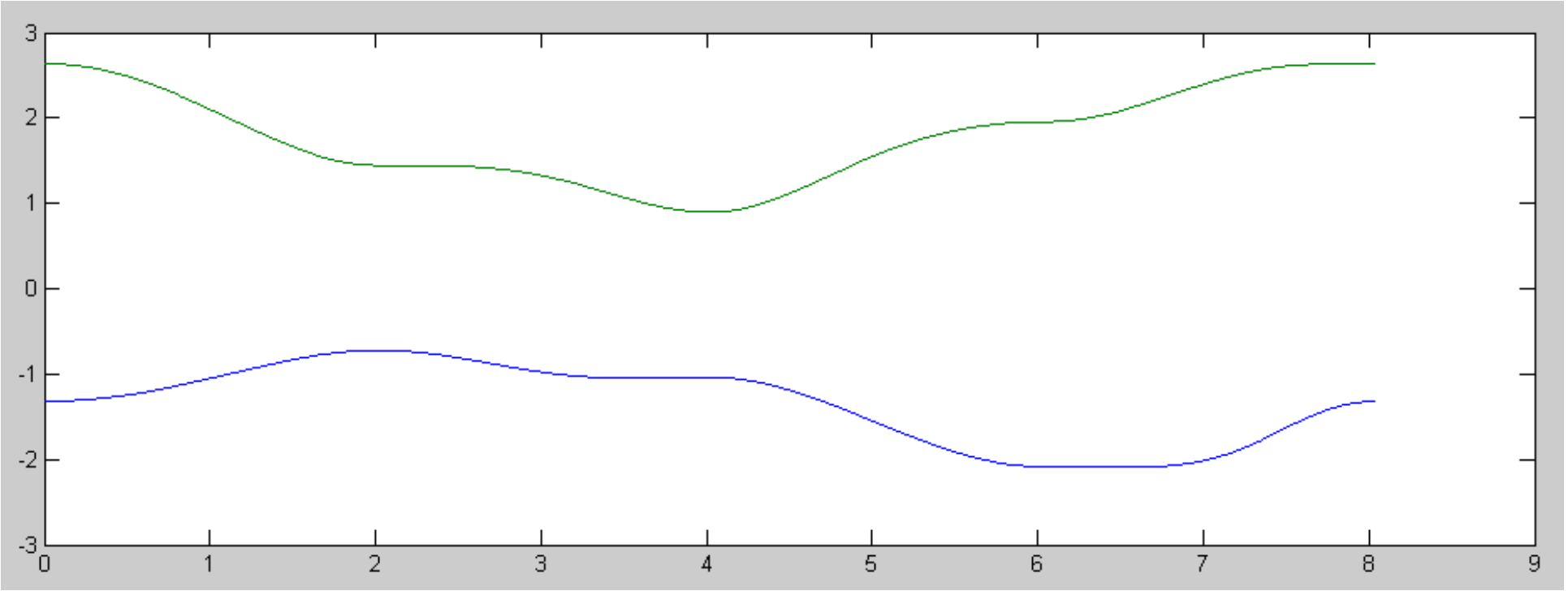
Next, convert these to joint angles.

```
function [Q] = InverseRR(TIP)
   x = TIP(1);
   y = TIP(2);

   r = sqrt(x^2 + y^2);
   Qa = atan2(y, x);
   Qb = acos(r/2);
   Q1 = Qa + Qb;
   Q2 = -2*Qb;

   Q = [Q1; Q2];
   end
```

With this, convert tip positions to joint angles

```
disp('Calculating joint angles');
% Determie the joint angles every 10ms
Qr = [];
for i=1:length(TIP)
    q = InverseRR(TIP(:,i));
    Qr = [Qr, q];
end
```

Program: Desired Joint Angles vs. Time



Desired Joint Angles vs Time for tracing out a square

PD Control

If you have decoupled systems with inertia, J, and no friction, the dynamics are

$$T = Js^2\theta$$

If you apply a proportional-derivative feedback control law

$$T = P(\theta_r - \theta) - Ds\theta$$

then the dynamics become

$$P\theta_r = Js^2\theta + Ds\theta + P\theta$$

or

$$\theta = \left(\frac{P}{Js^2+Ds+P}\right)\theta_r$$

D and P are chosen to place the poles of the closed-loop system.

Assume J = 5 (worst case for mass 1). To place the closed-loop poles at

$$s = -4 \pm j4$$

you get

$$Js^2 + Ds + P = 5(s^2 + 8s + 32)$$

D = 40

P = 160

Assume J = 1 (worse case for mass 2)

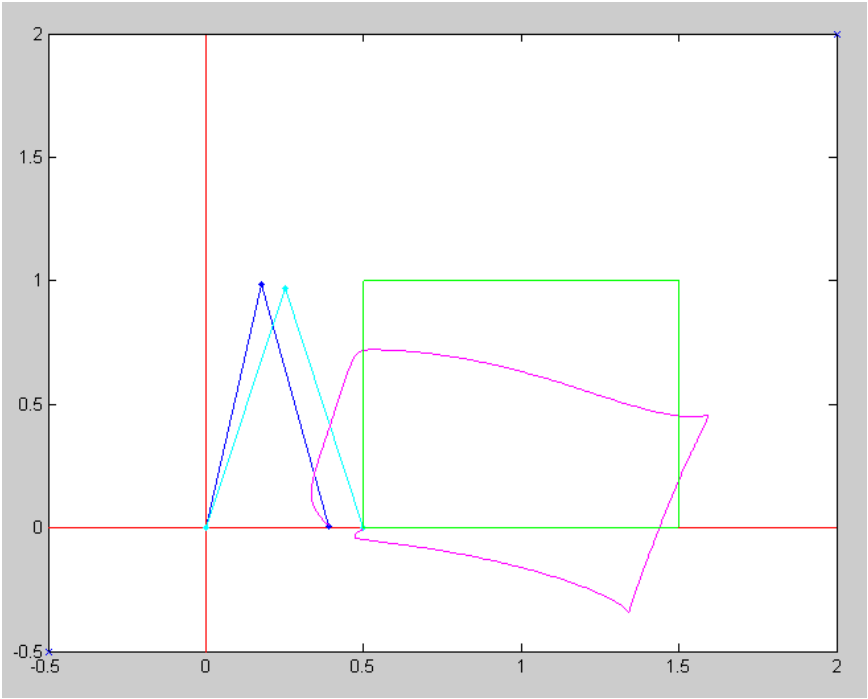
$$Js^2 + Ds + P = 1(s^2 + 2s + 2)$$

D = 2

P = 2

Applying this feedback control law

```
for i=1:length(Qr)

    T1 = 160*(Qr(1,i) - Q(1))  + 40*(0 - dQ(1));
    T2 = 32*(Qr(2,i) - Q(2))  + 8*(0 - dQ(2));
    T = [T1; T2];

    ddQ = TwoLinkDynamics(Q, dQ, T);
    dQ = dQ + ddQ * dt;
    Q = Q + dQ*dt;
    t = t + dt;
```



PD Control with Gravity Compensation

From before

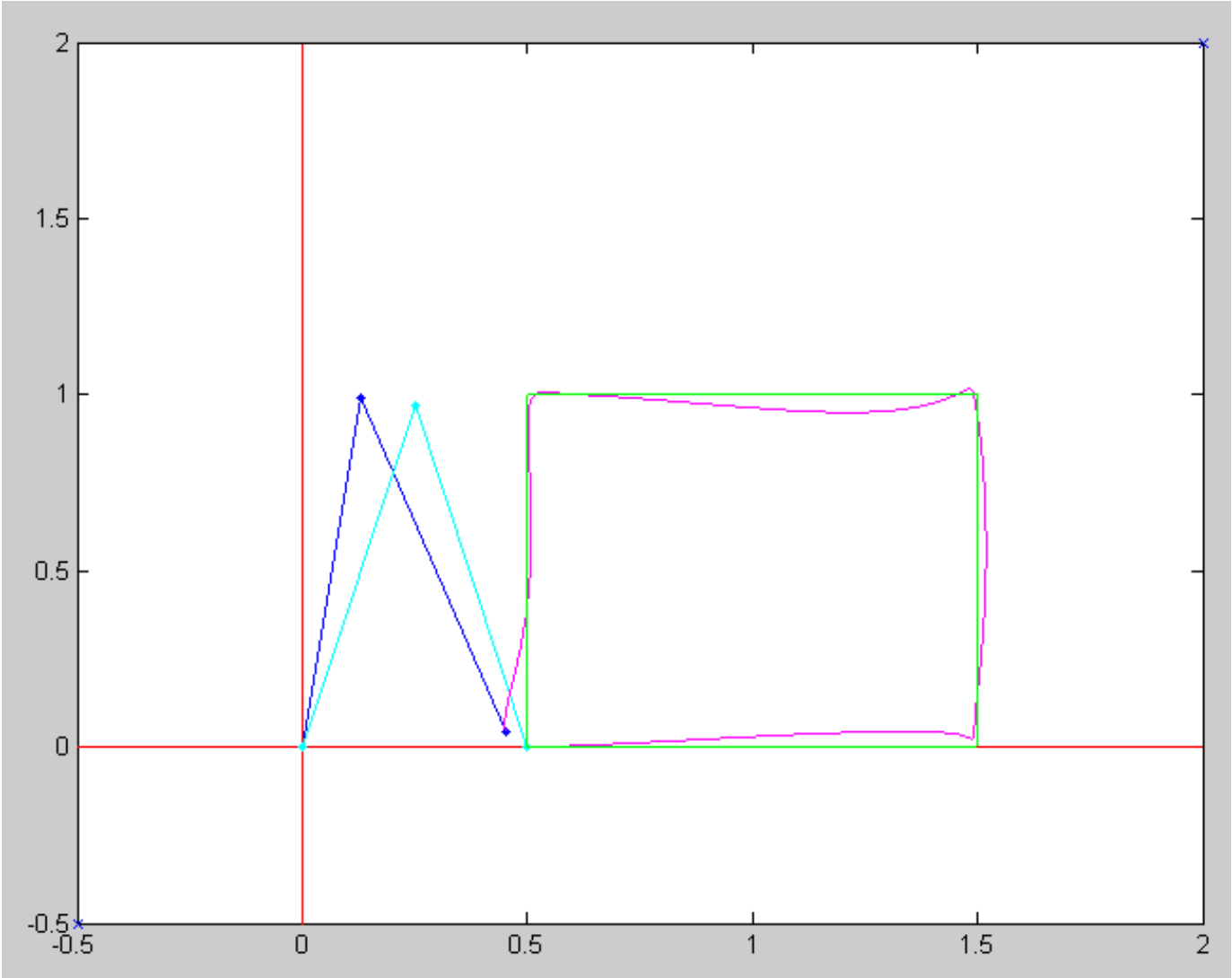
$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} (4+2c_2) & (1+c_2) \\ (1+c_2) & 1 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} - \begin{bmatrix} 2s_2\dot{\theta}_1\dot{\theta}_2 + s_2\dot{\theta}_2^2 \\ -s_2\dot{\theta}_1^2 \end{bmatrix} + g \begin{bmatrix} 3c_1 + c_{12} \\ c_{12} \end{bmatrix}$$

To compensate for gravity, add a term

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = T_{PD} - g \begin{bmatrix} 3c_1 + c_{12} \\ c_{12} \end{bmatrix}$$

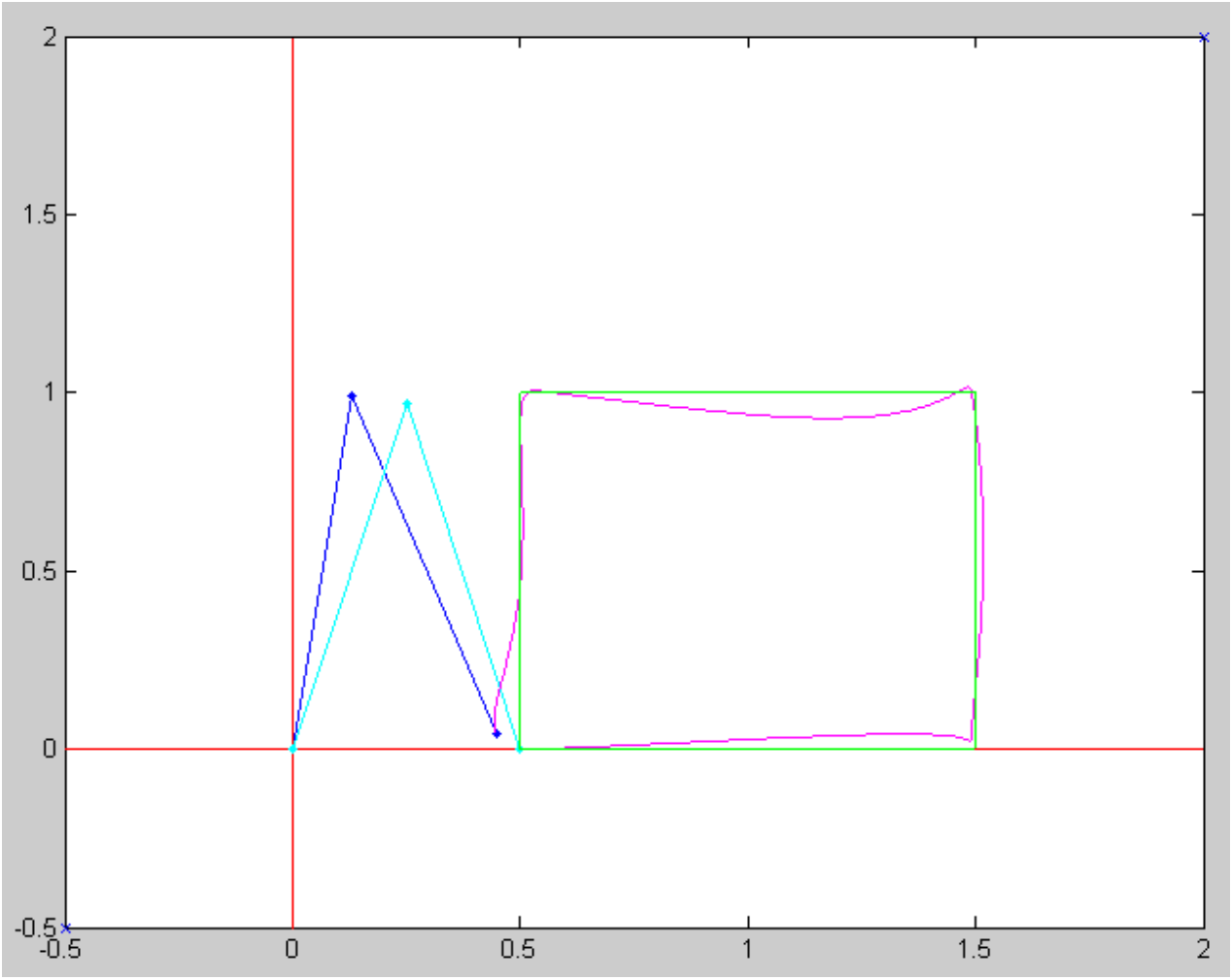
Note

- You know the joint angles vs. time (path planning)
- You can pre-calculate the gravity term.



PD Control with Gravity Feedforward Term

PD Control with Gravity and Coriolis Force



Velocity Feedfoward Control:

Once you cancel the gravity and coriolis terms, the dynamics become

$$\theta = \left(\frac{P}{Js^2+Ds+P}\right)\theta_r$$

Ideally, the transfer function should be 1 (meaning the angle exactly matches the desired angle). If you add a derivative term

$$T = T_{PD} - T_g + Ds\theta_r$$

you get

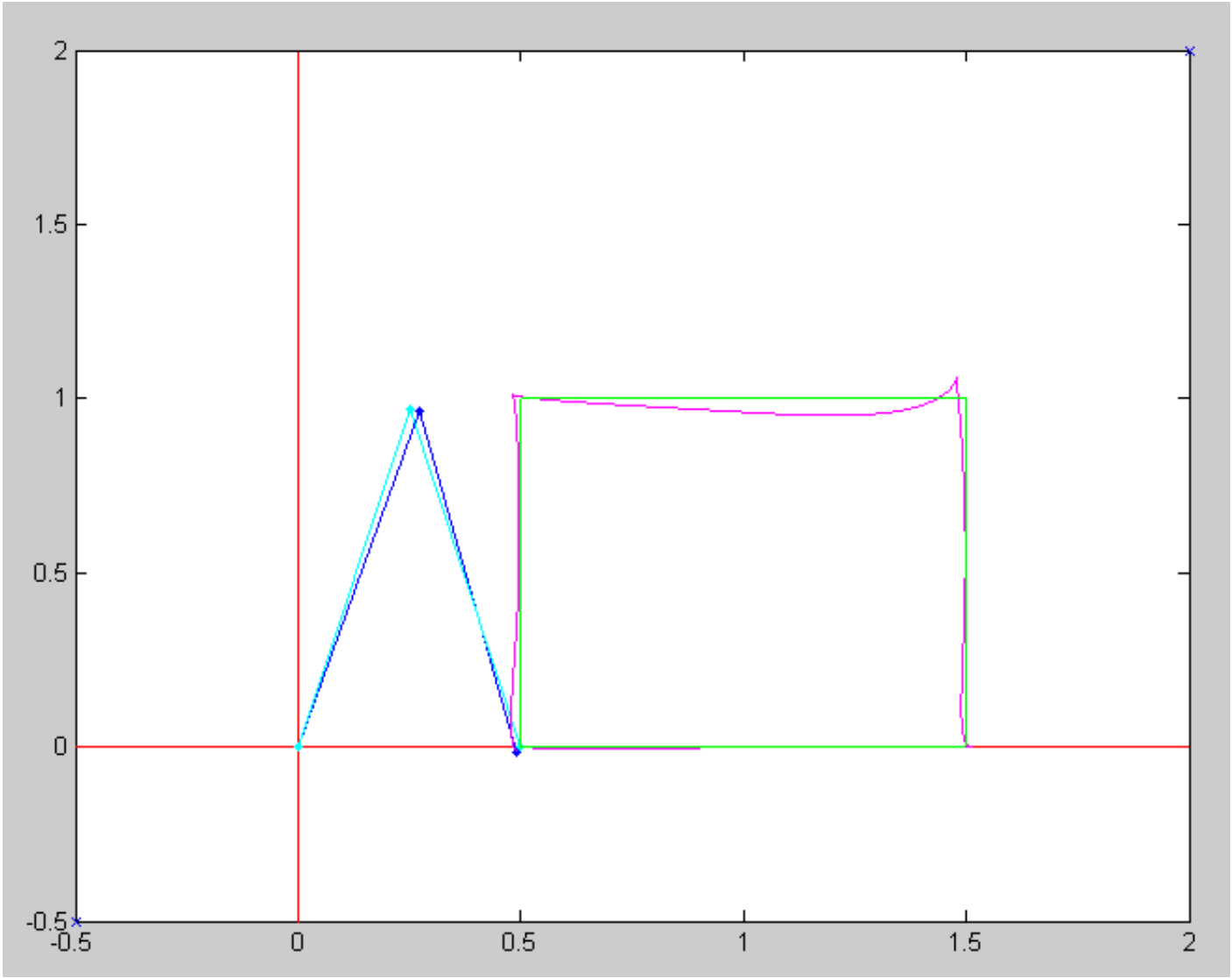
$$\theta = \left(\frac{Ds+P}{Js^2+Ds+P}\right)\theta_r$$

which is closed to one (meaning better tracking). To do this, you need to

- Take the derivative of the desired angles, and
- Bias the torque by D times this derivative

In Matlab:

```
% Velocity - right after computing the desired angles
dQr1 = Derivative(Qr(1,:));
dQr2 = Derivative(Qr(2,:));
dQr = [dQr1 ; dQr2];


for i=1:length(Qr)

    T1 = 160*(Qr(1,i) - Q(1))  + 40*(dQr(1,i) - dQ(1));
    T2 = 32*(Qr(2,i) - Q(2))  + 8*(dQr(2,i) - dQ(2));
    T = [T1; T2];

     % plus gravity
    T = T - G(:,i);
    % plus coriolis
    T = T - C(:,i);
```



PD Control + Gravity + Coriolis + Velocity

Acceleration Feedfoward Control:

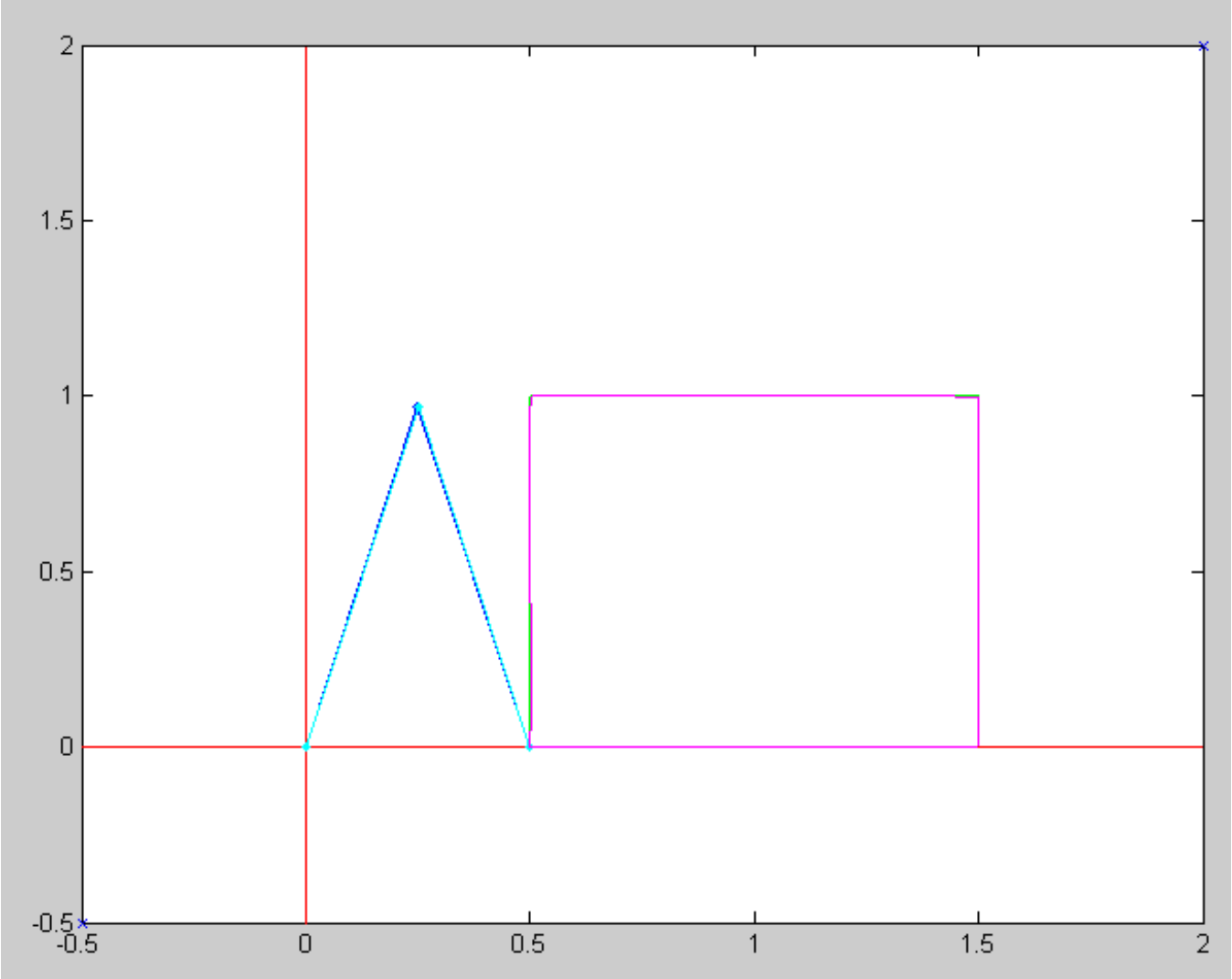
Finally, if you also bias the torque by the acceleration term:

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} (3+2c_2) & (1+c_2) \\ (1+c_2) & 1 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

you get a transfer function of

$$\theta = \left(\frac{Js^2+Ds+P}{Js^2+Ds+P} \right) \theta_r$$

```
% plus gravity
T = T - G(:,i);
% plus derivative
T = T + diag([40, 8]) *dQr(:,i);
% plus coriolis
T = T - C(:,i);
% plus acceleration
c2 = cos(Q(2));
T = T + [3+2*c2, 1+c2 ; 1+c2, 1]*ddQr(:,i);
```



Actual & Dsired Tip Position for PD, Gravity, Coriolis, Derivative, and Inertia Compensation