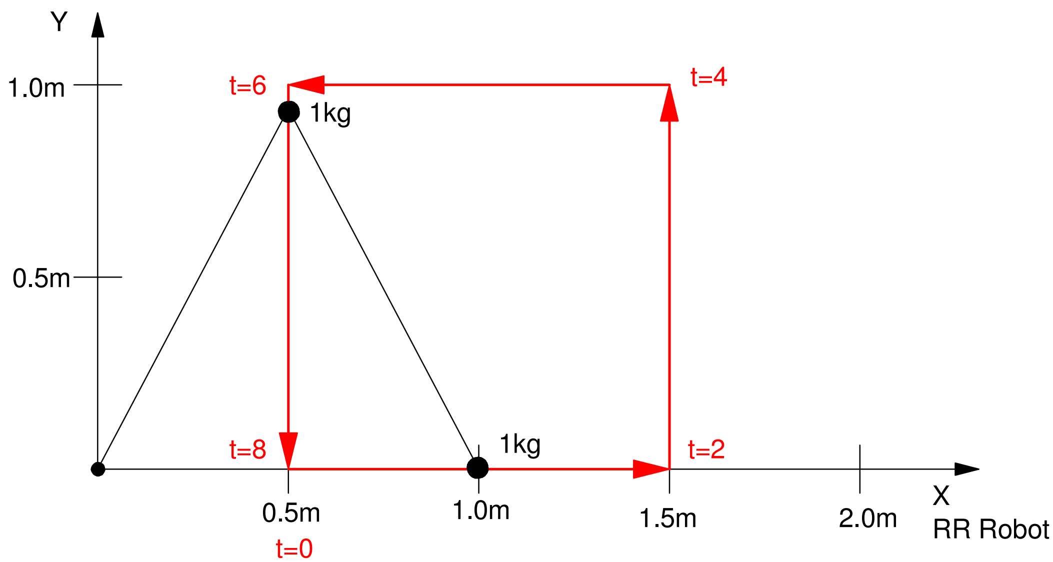
Control of a RR Robot

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



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.

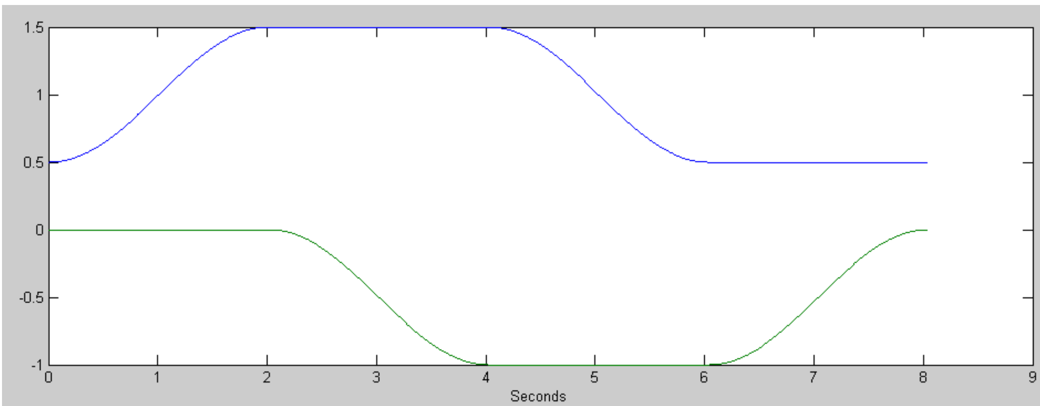
First, let's use the previous path-planning routines for the RRR robot to define the desired

- Tip positions, and
- Joint angles

First, define the tip positions. Using the MoveTo() routine from before, this can be done as follows:

```
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];
```



Desired Tip Position to Trace Out a Square

Next, convert these to joint angles. To do this, write a routine to compute the joint angle given the tip position:

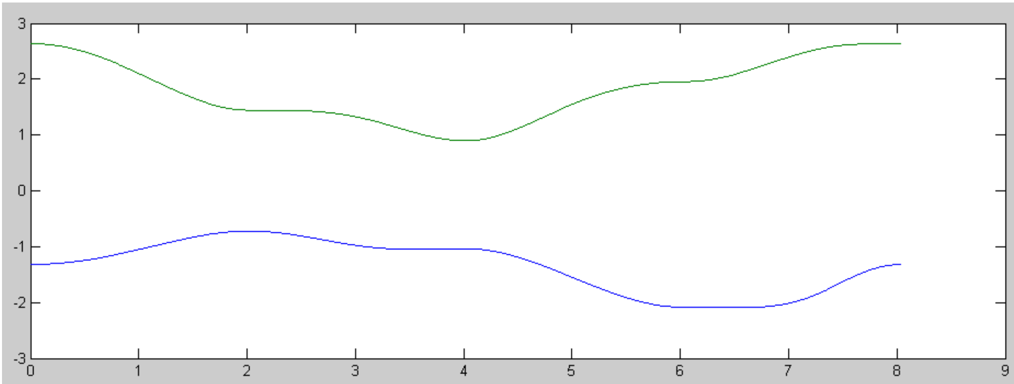
```
function [Q] = InverseRR(TIP)
   x = TIP(1);
   y = TIP(2);

   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
```



Desired Joint Angles vs Time for tracing out a square

Once you know where the joint angles are supposed to be, you can start defininig the feedback control law.

PD Control

If you have decoupled systems with inertia, J, and no friciton, 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

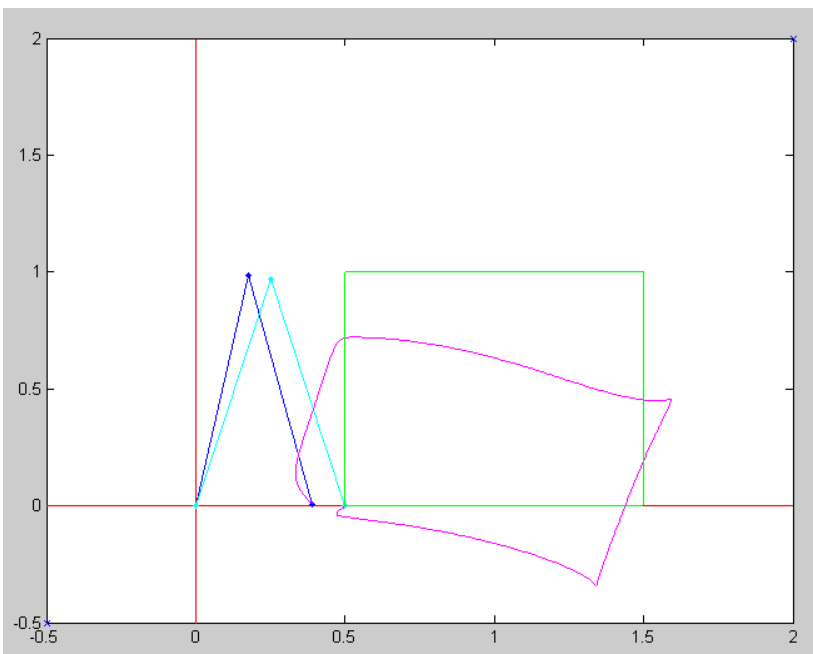
Applying this feedback control law

```
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;

  % rest of code ...
```



Tracking of the RR robot for a PD controller

One of the reasons the robot is not tracking the desired angle well is gravity is pulling down.

PD Control with Gravity Compensation (FeedForward Control)

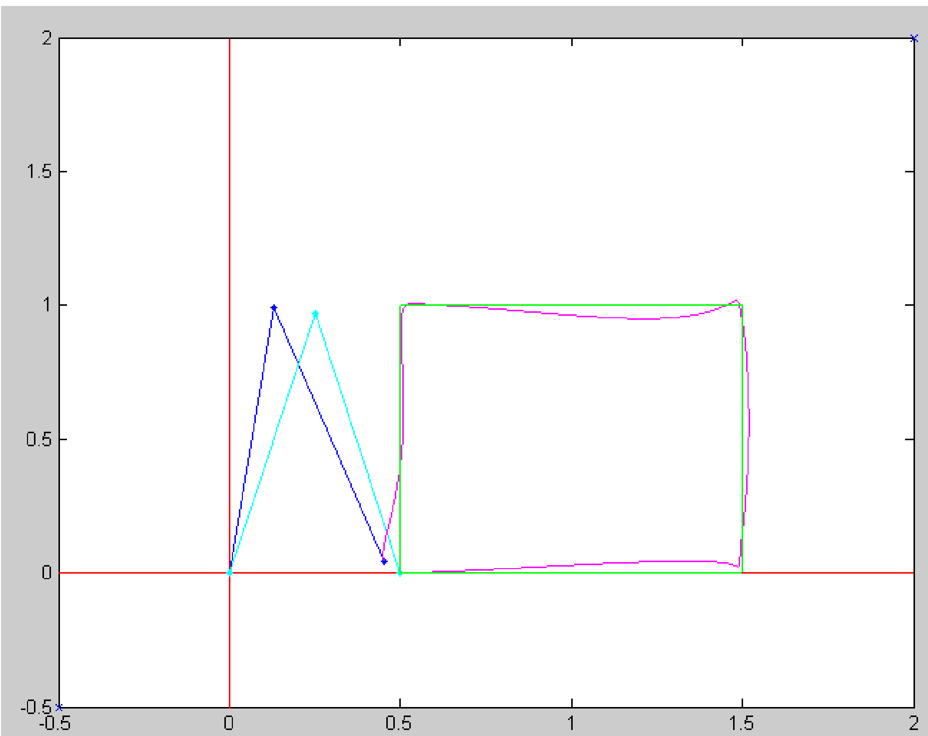
If you solve the previous dynamics for torque, you get:

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

To compensate for gravity, add a term

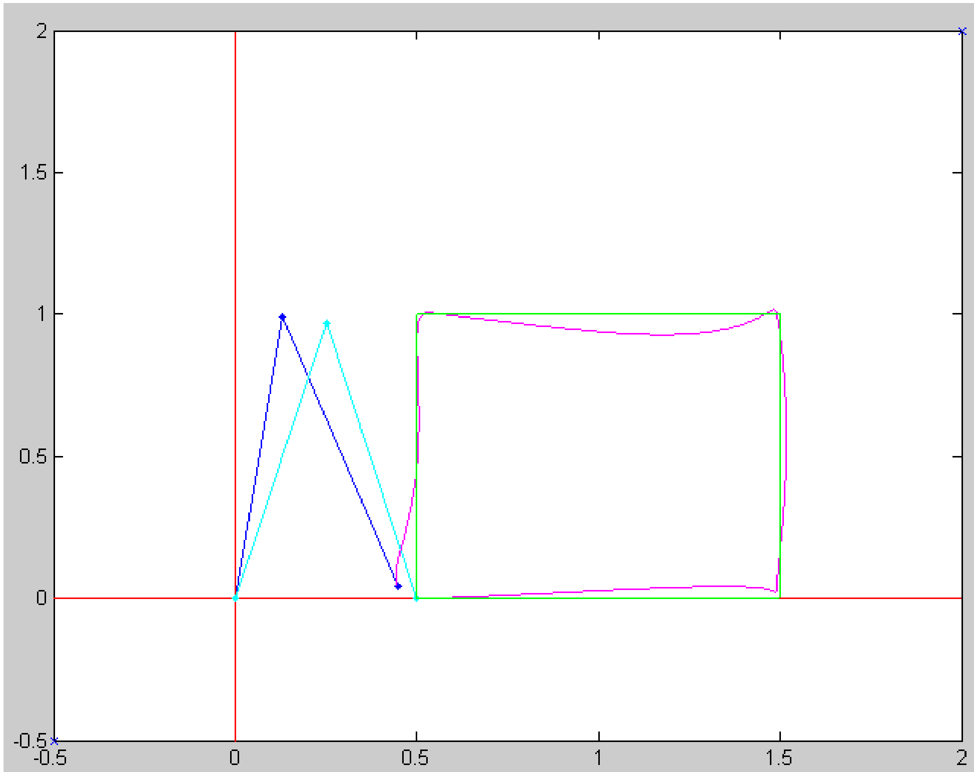
$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = T_{PD} - g\begin{bmatrix} 3c_1 + c_{12} \\ c_{12} \end{bmatrix}$$

Note that you can do this offline: once you compute the desired tip positions and angles, you can compute the torque due to gravity. This speeds up the compuations while running.



PD Control with Gravity and Coriolis Force Compensation (Feedforward Control)

Similarly, if you add in the coriolis forces as well you get slightly better tracking



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 funciton 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 deisred angles, and
- Bias the torque by D times this derivative

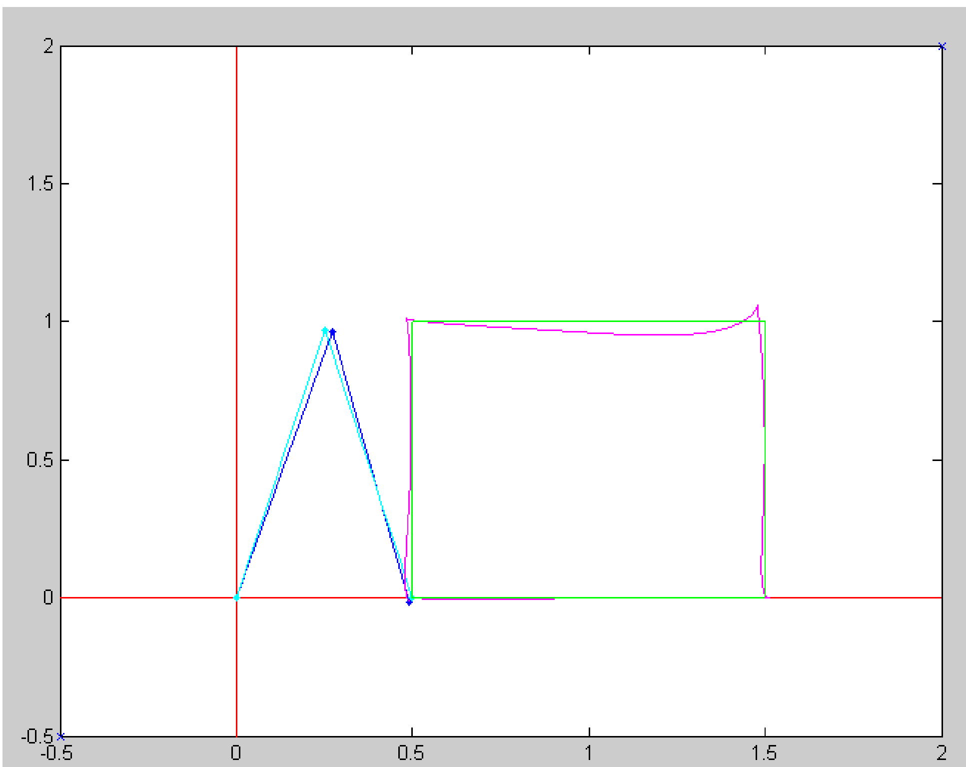
In Matlab:

```
% 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);
```



Accelearation Feedfoward Control:

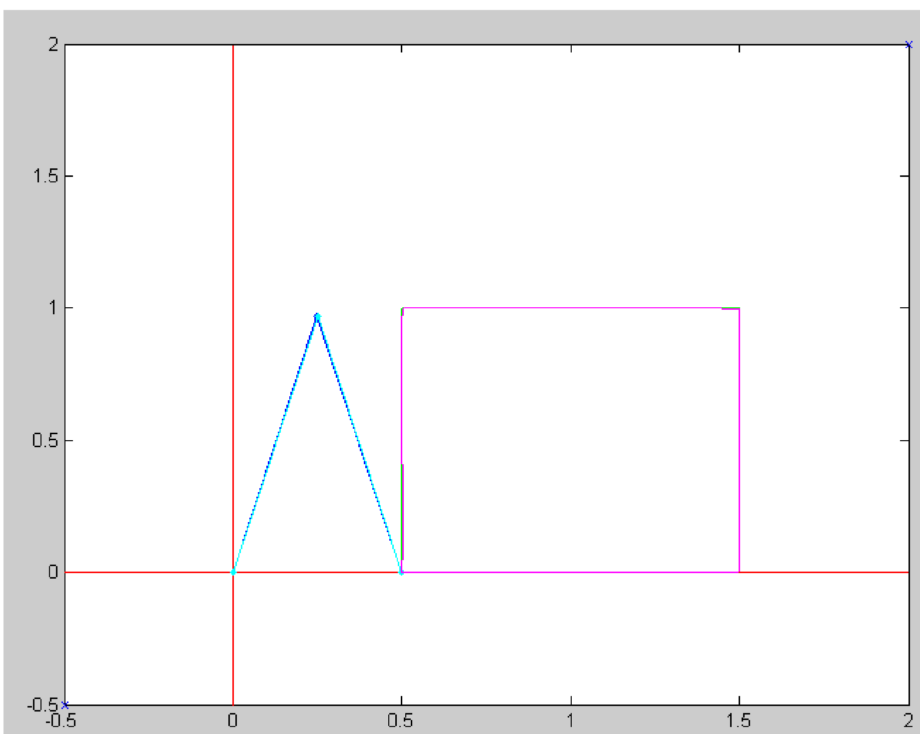
Finally, if you also bias the torque by the acceleration term:

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

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

```
% RR_Control.txt
% Position control of a RR robot
% similar to a RRR robot with Q1 = 0, meaning Y=0
%
% Define a square to trace
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');
   T = 2;
   t = [0.01:0.01:T];
   a = (1 - cos(t*pi/T))/2;
   T0 = P1*ones(1,50);
   T1 = P1*(1-a) + P2*a;
   T2 = P2*(1-a) + P3*a;
   T3 = P3*(1-a) + P4*a;
   T4 = P4*(1-a) + P5*a;
   T5 = P5*ones(1,50);
   TIP = [T0,T1,T2,T3,T4,T5];

disp('Calculating joint angles');
% Determie the joint angles every 10ms
    Qr = [];
    for i=1:length(TIP)
        q = InverseRR(TIP(:,i));
        Qr = [Qr, q];
        end
    c1 = cos(Qr(1,:));
    s1 = sin(Qr(1,:));
    c2 = cos(Qr(2,:));
    s2 = sin(Qr(2,:));
    c12 = cos(Qr(1,:) + Qr(2,:));
    s12 = sin(Qr(1,:) + Qr(2,:));

disp('Calulating gravity matrix');
% gravity
    g = 9.8;
    G = -g*[2*c1 + c12 ; c12 ];

disp('Calulating gravity torques');
% Velocity
    dQr1 = derivative(Qr(1,:));
    dQr2 = derivative(Qr(2,:));
    dQr = [dQr1 ; dQr2];

disp('Calulating coriolis torques');
% Coriolis Forces
    C = [ 2*s2.*dQr1.*dQr2 + s2.*dQr2.*dQr2 ; -s2.*dQr1.*dQr1 ];

disp('Calulating inertia torques');
% Acceleration
    ddQr1 = Derivative(dQr(1,:));
    ddQr2 = Derivative(dQr(2,:));
    ddQr = [ddQr1 ; ddQr2];

   dQ = [0; 0];
    T = [0; 0];
```

```
    t = 0;
    dt = 0.01;

% -------------------- Main Loop ----------------

X = [];
Xr = [];
T12 = [];

disp('Tracing out a Square');
    for i=1:length(Qr)

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

% plus derivative
% (already in the T1 and T2 equations )

% plus coriolis
        T = T - C(:,i)*0;

% plus acceleration
        c2 = cos(Q(2));
        T = T + [3+2*c2, 1+c2 ; 1+c2, 1]*ddQr(:,i)*0;

% Inegrate
        ddQ = RRDynamics(Q, dQ, T);
        dQ = dQ + ddQ * dt;
        Q = Q + dQ*dt;
        t = t + dt;

        RR(Q, Qr(:,i), TIP);

        X = [X , [cos(Q(1))+cos(Q(1)+Q(2)) ; sin(Q(1))+sin(Q(1)+Q(2))]];

        T12 = [T12, T];

        pause(0.01);
    end

pause(5);


 t = [1:length(TIP)] * 0.01;
 clf
 subplot(211)
 plot(t,X,t,TIP);
 xlabel('Time (seconds)');
 ylabel('Tip (meters)');
 subplot(212)
 plot(t,T12);
 xlabel('Time (seconds)');
 ylabel('Torque (Nm)');
```