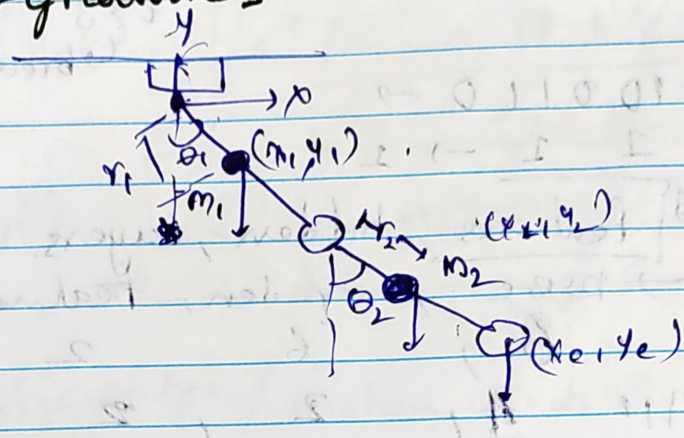
(a) Dynamics



$x_1 = r_1 \cos\theta_1$ — (1)
$y_1 = r_1 \sin\theta_1$ — (2)
} Position of CM (center of Mass) link1

$x_2 = L_1 \cos\theta_1 + r_2 \cos(\theta_1+\theta_2)$ — (3)
$y_2 = L_1 \sin\theta_1 + r_2 \sin(\theta_1+\theta_2)$ — (4)
} CM - link2

$\dot{x}_1 = -r_1 \sin\theta_1 \cdot \dot{\theta}_1$
$\dot{y}_1 = (+r_1 \cos\theta_1)\dot{\theta}_1$
} velocity of CM link1

velocity of CM - link2

$\dot{x}_1 = -L_1 \sin\theta_1 \dot{\theta}_1 - r_2 \sin(\theta_1+\theta_2)\dot{\theta}_1 - r_2 \sin(\theta_1+\theta_2)\dot{\theta}_2$

$\dot{y}_1 = L_1 \cos\theta_1 + r_2 \cos(\theta_1+\theta_2)\dot{\theta}_1 + r_2 \cos(\theta_1+\theta_2)\dot{\theta}_2$

Total kinetic energy

$K.E = \cancel{\tfrac{1}{2} m_1 (\dot{x}_1^2 + \dot{x}_2^2)}$

$= \tfrac{1}{2}(m_1 \dot{x}_1^2 + m_2 \dot{x}_2^2) + \tfrac{1}{2} I_1 (\dot{\theta}_1^2) + I_2 \dot{\theta}_2^2$

Total potential energy

$P.E = m_1 g y_1 + m_2 g y_2$

$\therefore$ the lagrangian is given by

$$L = K - V$$

$$L = \left[\frac{1}{2}\left(m_1 \dot{x}_1^2 + m_2 \dot{x}_2^2 + m_1 \dot{y}_1^2 + m_2 \dot{y}_2^2\right) + \frac{1}{2} I_1 \dot{\theta}_1^2 + \frac{1}{2} I_2 \dot{\theta}_2^2\right]$$

$$- \left(m_1 g y_1 + m_2 g y_2\right)$$

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q}$$

$\rightarrow$ solving this with mathematica.

$$\tau = M(\theta)\ddot{\theta} + C(\theta,\dot{\theta}) + G(\theta)$$

$$\tau_1 = \ddot{\theta}_2 (l_1 m_2 r_2 \cos\theta_2 + m_2 r_2^2) + \ddot{\theta}_1 (2 l_1 m_2 r_2 \cos\theta_2 + m_2 r_2^2 + m_1 r_1^2 + l_1^2 m_2 + I_1)$$

$$+ \; -2 l_1 m_2 r_2 \sin\theta_2 \dot{\theta}_1 \dot{\theta}_2 - l_1 m_2 r_2 \sin\theta_2 \dot{\theta}_2^2 + g l_1 m_2 \cos\theta_1 + g m_1 r_1 \cos\theta_1 + g m_2 r_2 \cos(\theta_1+\theta_2).$$

$$\tau_2 = \ddot{\theta}_2 m_2 r_2^2 + \ddot{\theta}_1 (l_1 m_2 r_2 \cos\theta_2 + m_2 r_2^2 + I_2) + \dot{\theta}_1^2 (l_1 m_2 r_2 \sin\theta_2) + g m_2 r_2 \cos(\theta_1+\theta_2)$$

$$\begin{pmatrix}\tau_1 \\ \tau_2\end{pmatrix} = \overbrace{\begin{bmatrix} I_1 + l_1^2 m_2 + m_1 r_1^2 + m_2 r_2^2 + 2 l_1 m_2 r_2 \cos\theta_2 & m_2 r_2^2 + l_1 m_2 r_2 \cos\theta_2 \\ m_2 r_2^2 + l_1 m_2 r_2 \cos\theta_2 & I_2 + m_2 r_2^2 \end{bmatrix}}^{M} \begin{bmatrix}\ddot{\theta}_1 \\ \ddot{\theta}_2\end{bmatrix}$$

$$+ \overbrace{\begin{bmatrix} -2 l_1 m_2 r_2 \sin(\theta_2)\dot{\theta}_2 & -l_1 m_2 r_2 \sin\theta_2 \dot{\theta}_2 \\ l_1 m_2 r_2 \sin\theta_2 \dot{\theta}_1 & 0 \end{bmatrix}}^{C} * \begin{bmatrix}\dot{\theta}_1 \\ \dot{\theta}_2\end{bmatrix}$$

$$+ \overbrace{\begin{bmatrix} g m_2 l_1 \cos\theta_1 + g m_1 r_1 \cos\theta_1 + g m_2 r_2 \cos(\theta_1+\theta_2) \\ g m_2 r_2 \cos(\theta_1+\theta_2) \end{bmatrix}}^{G}$$

$$\therefore \tau = M(\theta)\ddot{\theta} + C(\theta,\dot{\theta}) + g(\theta).$$ $\boxed{\text{solution obtained in MATHEMATICA}}$

(b) (coded in python, attached the ipython notebook).

lets parameterize the equations for the joint angles as the 5th order polynomial

$T = [0, 2]$

$$\theta_1(t) = a_1 t^5 + b_1 t^4 + c_1 t^3 + d_1 t^2 + e_1 t + f_1$$
$$\theta_2(t) = a_2 t^5 + b_2 t^4 + c_2 t^3 + d_2 t^2 + e_2 t + f_2$$

using the boundary conditions

$\theta_1(0) = -\pi/4 \quad \theta_2(0) = 0$

$\theta_1(2) = \pi/4 \quad \theta_2(2) = \pi/2$

$f_1 = -\pi/4$

$a_1 32 + b_1 16 + 8c_1 + d_1 4 + e_1 2 + f_1 = \pi/4$

$f_2 = 0$

$a_2 32 + b_2 16 + c_2 8 + d_2 4 + e_2 2 + f_2 = \pi/2$

we can also derive boundary conditions for $\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2$

$$\Rightarrow \dot{\theta}_i = 5a_i t^4 + 4b_i t^3 + 3c_i t^2 + 2d_i t + e_i$$

$$\Rightarrow \ddot{\theta}_i = 2d_i + 6c_i t + 12b_i t^2 + 20a_i t^3$$

The functions in the ipython notebook work as follow

→ trajectory boundary conditions are stored in variables thetai_bvp as dictionary containing positions (pos), velocities (vel), acceleration (acc) at time(t) as a list, same for vel, acc.

{pos: [[times], [pos values]]}

→ trajGen-function computes the coefficients $a_i, b_i, c_i, d_i, e_i, f_i$ → (passed to trajectory)

→ trajectory function works as follows
trajectory (time, coeffecient list)
for theta1, theta2

Ⓒ Designing the controller.

The error dynamics is given by

$$\theta_e(t) = \theta_d(t) - \theta(t)$$

$$\Rightarrow \theta(t) = \theta_d - \theta_e$$

$\theta = \theta_d - \theta_e$

$\dot{\theta} = -\dot{\theta}_e$

$\ddot{\theta} = -\ddot{\theta}_e$

I implemented the Feed forward plus the Feedback control in section 11.4.1.3

using the error dynamics as

$\theta_e = \theta_d - \theta \Rightarrow \ddot{\theta} = \ddot{\theta}_d - \ddot{\theta}_e$

we obtain τ as

$$\tau = \tilde{M}(\theta)\left(\ddot{\theta}_d + k_p\theta_e + k_i\int \theta_e(t)\,dt + k_p\dot{\theta}_e\right) + h(\theta,\dot{\theta})$$

$h(\theta,\dot{\theta}) = c(\theta,\dot{\theta}) + g(\theta)$

① Once we get τ_1, τ_2:

② To get the actual angle we can solve the differential equations to obtain θ_1, θ_2,

$u = \dot{\theta}_1(t)$, $v = \dot{\theta}_2(t)$

$k_1\dot{u} + k_2\dot{v} - k_3uv - k_4v^2 + k_{51} - \tau_1 = 0$ — ①

$k_5 \cdot \dot{u} + k_6\dot{v} + k_8 - k_7u^2 - \tau_2 = 0$ — ②

solving ①, ② for $\dot{u}, \dot{v}$ we get

$$\dot{u} = \frac{\left(k_{s1} \cdot k_6 + k_2 \cdot k_8 + k_6 \theta_1 + k_3 k_6 u \cdot v - k_2 \theta_2 - k_2 k_7 u^2 + k_4 k_6 v^2\right)}{\left(k_2 k_5 - k_1 k_6\right)}$$

$$\dot{v} = \frac{\left(-k_5 \cdot k_{s1} + k_1 k_8 + k_5 \theta_1 - k_1 \theta_2 - k_1 k_7 u^2 + k_3 u \cdot v + k_4 k_5 \cdot v^2\right)}{-k_2 k_5 + k_1 k_6}$$

③ Integrate $\dot{u}, \dot{v}$ to obtain $\dot{\theta}_1, \dot{\theta}_2$ & integrate $\dot{\theta}_1$ & $\dot{\theta}_2$ to obtain θ_1, θ_2

Thus we have the sense & feed back loop ready for PID the algorith is implemented in python in the accompanying ipython notebook.

```
In[ ]:= x1 = r1 * Cos[θ1[t]];
        y1 = r1 * Sin[θ1[t]];
        x2 = L1 * Cos[θ1[t]] + r2 * Cos[θ1[t] + θ2[t]];
        y2 = L1 * Sin[θ1[t]] + r2 * Sin[θ1[t] + θ2[t]] ;

In[ ]:= x1dot = D[x1, t];

In[ ]:= y1dot = D[y1, t];
        x2dot = D[x2, t];
        y2dot = D[y2, t];

In[ ]:= P = m1 * g * y1 + m2 * g * y2;

In[ ]:= K = 0.5 * (m1 * (x1dot ^ 2 + y1dot ^ 2) + m2 * (x2dot ^ 2 + y2dot ^ 2)) +
           0.5 * ((I1 * (D[θ1[t], {t, 1}]) ^ 2) + (I2 * (D[θ2[t], {t, 1}]) ^ 2));

In[ ]:= L = K - P;

In[ ]:= term1 = D[Grad[L, {D[θ1[t], t], D[θ2[t], t]} ], t];(*d/dt(dL/dqdot)*)

In[ ]:= term2 = Grad[L, {θ1[t], θ2[t]}]; (*dL/dq*)

In[ ]:= Tau = term1 - term2;

In[ ]:= tau1 = TrigReduce[Tau⟦1⟧]

        g L1 m2 Cos[θ1[t]] + g m1 r1 Cos[θ1[t]] + g m2 r2 Cos[θ1[t] + θ2[t]] - 2.` L1 m2 r2 Sin[θ2[t]] θ1′[t] θ2′[t] -
          1.` L1 m2 r2 Sin[θ2[t]] θ2′[t]^2 + 1.` I1 θ1″[t] + 1.` L1^2 m2 θ1″[t] + 1.` m1 r1^2 θ1″[t] +
          1.` m2 r2^2 θ1″[t] + 2.` L1 m2 r2 Cos[θ2[t]] θ1″[t] + 1.` m2 r2^2 θ2″[t] + M =
         np.array ([[m1 * r1 ** 2 + m2 * r2 ** 2 + m2 * L1 ** 2 + 2 * m2 * L1 * r2 * np.cos (l_theta2 _[i]), m2 * L1 *
              r2 * np.cos (l_theta2 _[i])], [m2 * r2 ** 2 + m2 * L1 * r2 * np.cos (l_theta2 _[i]), m2 * r2 ** 2]])
        C = np.array ([[- m2 * L1 * r2 * l_theta2 _ddot[i] * np.sin (l_theta2 _[i]),
            - m2 * L1 * r2 * l_theta1 _ddot[i] * np.sin (l_theta2 _[i]) -
             m2 * L1 * r2 * l_theta2 _ddot[i] * np.sin (l_theta2 _[i])], [
            m2 * L1 * r2 * l_theta1 _ddot[i] * np.sin (l_theta1 _[i]), 0]])
        G = np.array ([(m1 * L1 + m2 * L1) * g * np.cos (l_theta1 _[i]) + m2 * r2 * g * np.cos
              (l_theta2 _[i] + l_theta1 _[i]), m2 * r2 * g * np.cos (l_theta2 _[i] + l_theta1 _[i])])
        1.`
         L1
         m2
         r2
         Cos[
          θ2[
           t]] θ2″[t]

In[ ]:= tau2 = TrigReduce[Tau⟦2⟧]

Out[ ]= g m2 r2 Cos[θ1[t] + θ2[t]] + 1. L1 m2 r2 Sin[θ2[t]] θ1′[t]^2 +
         1. m2 r2^2 θ1″[t] + 1. L1 m2 r2 Cos[θ2[t]] θ1″[t] + 1. I2 θ2″[t] + 1. m2 r2^2 θ2″[t]
```

In[ ]:= poly = a*t^5+ b*t^4+c*t^3+d*t^2+e*t+f

Out[ ]= $f + e\,t + d\,t^2 + c\,t^3 + b\,t^4 + a\,t^5$

In[ ]:= polyVel = D[poly, t]

Out[ ]= $e + 2\,d\,t + 3\,c\,t^2 + 4\,b\,t^3 + 5\,a\,t^4$

In[ ]:= polyAcc = D[poly, {t, 2}]

Out[ ]= $2\,d + 6\,c\,t + 12\,b\,t^2 + 20\,a\,t^3$

In[ ]:= (*Solve[{Diff1==0,Diff2==0},{p,q}]*)

In[ ]:= tau2

Out[ ]= $g\,m2\,r2\,\text{Cos}[\theta1[t]+\theta2[t]] + 1.\,L1\,m2\,r2\,\text{Sin}[\theta2[t]]\,\theta1'[t]^2 + 1.\,m2\,r2^2\,\theta1''[t] + 1.\,L1\,m2\,r2\,\text{Cos}[\theta2[t]]\,\theta1''[t] + 1.\,I2\,\theta2''[t] + 1.\,m2\,r2^2\,\theta2''[t]$

In[ ]:= M = MatrixForm[
{{I1+m1*r1^2+m2*r2^2+m2*L1^2+2 m2*L1*r2*Cos[θ2], m2*r2^2+m2*L1*r2*Cos[θ2]},
{m2*r2^2+m2*L1*r2*Cos[θ2], m2*r2^2+I2}}]

Out[ ]//MatrixForm=

$$\begin{pmatrix} I1 + L1^2\,m2 + m1\,r1^2 + m2\,r2^2 + 2\,L1\,m2\,r2\,\text{Cos}[\theta2] & m2\,r2^2 + L1\,m2\,r2\,\text{Cos}[\theta2] \\ m2\,r2^2 + L1\,m2\,r2\,\text{Cos}[\theta2] & I2 + m2\,r2^2 \end{pmatrix}$$

In[ ]:= C1 =
MatrixForm[{{-m2*2*L1*r2*D[θ2[t], {t, 1}]*Sin[θ2], -m2*L1*r2*D[θ2[t], {t, 1}]*Sin[θ2]},
{m2*L1*r2*D[θ1[t], {t, 1}]*Sin[θ2], 0}}]

Out[ ]//MatrixForm=

$$\begin{pmatrix} -2\,L1\,m2\,r2\,\text{Sin}[\theta2]\,\theta2'[t] & -L1\,m2\,r2\,\text{Sin}[\theta2]\,\theta2'[t] \\ L1\,m2\,r2\,\text{Sin}[\theta2]\,\theta1'[t] & 0 \end{pmatrix}$$

In[ ]:= G = Simplify[
MatrixForm[{{g *L1 *m2 *Cos[θ1[t]]+g *m1 *r1* Cos[θ1[t]]+g *m2 *r2 *Cos[θ1[t]+θ2[t]]},
{g *m2* r2* Cos[θ1[t]+θ2[t]]}}]]

Out[ ]//MatrixForm=

$$\begin{pmatrix} g\,((L1\,m2 + m1\,r1)\,\text{Cos}[\theta1[t]] + m2\,r2\,\text{Cos}[\theta1[t]+\theta2[t]]) \\ g\,m2\,r2\,\text{Cos}[\theta1[t]+\theta2[t]] \end{pmatrix}$$

In[ ]:= MInv = Inverse[{{m11, m12}, {m21, m22}}]

Out[ ]= $\left\{\left\{\frac{m22}{-m12\,m21+m11\,m22}, -\frac{m12}{-m12\,m21+m11\,m22}\right\}, \left\{-\frac{m21}{-m12\,m21+m11\,m22}, \frac{m11}{-m12\,m21+m11\,m22}\right\}\right\}$

In[ ]:= m11 = M[[1, 1, 1]]

Out[ ]= $I1 + L1^2\,m2 + m1\,r1^2 + m2\,r2^2 + 2\,L1\,m2\,r2\,\text{Cos}[\theta2]$

In[ ]:= m12 = M[[1, 1, 2]]

Out[ ]= $m2\,r2^2 + L1\,m2\,r2\,\text{Cos}[\theta2]$

```
In[ ]:= m21 = M[[1, 2, 1]]
Out[ ]= m2 r2^2 + L1 m2 r2 Cos[θ2]

In[ ]:= m22 = M[[1, 2, 2]]
Out[ ]= I2 + m2 r2^2

(*Dot[MInv ,{{tau11},{tau21}}], u=theta1dot ,v=theta2dot *)
```

Obtaining Differential Equations

```
In[ ]:= Solve[{k1 * D[u[t], t] + k2 * D[v[t], t] - k3 * u * v - k4 * v ^ 2 + k51 - th1 == 0 ,
          k5 * D[u[t], t] + k6 * D[v[t], t] - k7 * u ^ 2 + k8 - th2 == 0 }, {D[u[t], t], D[v[t], t]}]
Out[ ]= {{Derivative [1][u][t] →
          -(((-k51) * k6 + k2 * k8 + k6 * th1 - k2 * th2 - k2 * k7 * u ^ 2 + k3 * k6 * u * v + k4 * k6 * v ^ 2) /
            (k2 * k5 - k1 * k6)),
         Derivative [1][v][t] → -(((-k5) * k51 + k1 * k8 + k5 * th1 - k1 * th2 -
              k1 * k7 * u ^ 2 + k3 * k5 * u * v + k4 * k5 * v ^ 2) / ((-k2) * k5 + k1 * k6))}}

In[ ]:= Collect[tau1 , {D[θ1[t], {t, 2}], D[θ2[t], t], D[θ1[t], t], D[θ2[t], t]}]
Out[ ]= g L1 m2 Cos[θ1[t]] + g m1 r1 Cos[θ1[t]] + g m2 r2 Cos[θ1[t] + θ2[t]] -
         2. L1 m2 r2 Sin[θ2[t]] θ1'[t] θ2'[t] - 1. L1 m2 r2 Sin[θ2[t]] θ2'[t]^2 +
         (1. I1 + 1. L1^2 m2 + 1. m1 r1^2 + 1. m2 r2^2 + 2. L1 m2 r2 Cos[θ2[t]]) θ1''[t] +
         1. m2 r2^2 θ2''[t] + 1. L1 m2 r2 Cos[θ2[t]] θ2''[t]

In[ ]:= Collect[tau2 , {D[θ1[t], {t, 2}], D[θ2[t], t], D[θ1[t], t], D[θ2[t], t]}]
Out[ ]= g m2 r2 Cos[θ1[t] + θ2[t]] + 1. L1 m2 r2 Sin[θ2[t]] θ1'[t]^2 +
         (1. m2 r2^2 + 1. L1 m2 r2 Cos[θ2[t]]) θ1''[t] + 1. I2 θ2''[t] + 1. m2 r2^2 θ2''[t]

In[ ]:= Solve[{k1 * D[u[t], t] + k2 * D[v[t], t] - k3 * u * v - k4 * v ^ 2 - th1 == 0 ,
           k5 * D[u[t], t] + k6 * D[v[t], t] - k7 * u * v - th2 == 0 }, {u, v}];
```

In [1]:
```
from sympy.solvers import solve
from sympy import symbols
from scipy.integrate import solve_ivp
import numpy as np
import scipy.integrate as it
import pdb
```

In [2]:
```
a1,b1,c1,d1,e1,f1 = symbols('a1 b1 c1 d1 e1 f1')
a2,b2,c2,d2,e2,f2= symbols('a2 b2 c2 d2 e2 f2')
t = symbols('t')
```

In [3]:
```
eq_pos_t1 = a1*t**5 + b1*t**4 + c1*t**3 + d1*t**2 + e1*t + f1
eq_vel_t1 = e1 + 2*d1*t + 3*c1*t**2 + 4*b1*t**3 + 5*a1*t**4
eq_acc_t1 = 2*d1 + 6*c1*t + 12*b1*t**2 + 20*a1*t**3

eq_pos_t2 = a2*t**5 + b2*t**4 + c2*t**3 + d2*t**2 + e2*t + f2
eq_vel_t2 = e2 + 2*d2*t + 3*c2*t**2 + 4*b2*t**3 + 5*a2*t**4
eq_acc_t2 = 2*d2 + 6*c2*t + 12*b2*t**2 + 20*a2*t**3
```

In [4]:
```
theta1_bvp={'pos':[[0,2],[-np.pi/4., np.pi/4.]], 'vel':[[0,2],[0,0]], 'acc':[[
#create the boundary value problem for theta1
```

In [5]:
```
theta2_bvp={'pos':[[0,2],[0., np.pi/2.]], 'vel':[[0,2],[0,0]], 'acc':[[0,2],[0
#create the boundary value problem for theta2
```

In [6]:
```
#trajectory generator function returns the coefficients of the polynomials
def trajGen(theta1_bvp, theta2_bvp):
    expr_list_t1 =  []
    for i in range(len(theta1_bvp['pos'][0])):
        expr_list_t1.append(eq_pos_t1.subs(t,theta1_bvp['pos'][0][i])-theta1_b
    for i in range(len(theta1_bvp['vel'][0])):
        expr_list_t1.append(eq_vel_t1.subs(t,theta1_bvp['vel'][0][i])-theta1_b
    for i in range(len(theta1_bvp['vel'][0])):
        expr_list_t1.append(eq_acc_t1.subs(t,theta1_bvp['acc'][0][i])-theta1_b
    expr_list_t2 =  []
    for i in range(len(theta2_bvp['pos'][0])):
        expr_list_t2.append(eq_pos_t2.subs(t,theta2_bvp['pos'][0][i])-theta2_b
    for i in range(len(theta2_bvp['vel'][0])):
        expr_list_t2.append(eq_vel_t2.subs(t,theta2_bvp['vel'][0][i])-theta2_b
    for i in range(len(theta2_bvp['vel'][0])):
        expr_list_t2.append(eq_acc_t2.subs(t,theta2_bvp['acc'][0][i])-theta2_b
    return solve(expr_list_t1), solve(expr_list_t2)
```

In [7]:
```
#get the coeffecients
theta1co, theta2co = trajGen(theta1_bvp, theta2_bvp)
# theta1co
# print(theta2co)
```

In [8]:

```
#takes the time and the coefficients gives the value of the q,qdot,qddot q=the
def trajectory(time1, theta1co, theta2co):
    theta1co[t]=time1
    theta2co[t]=time1
    q = [eq_pos_t1.subs(theta1co),eq_pos_t2.subs(theta2co)]
    qdot = [eq_vel_t1.subs(theta1co),eq_vel_t2.subs(theta2co)]
    qddot = [eq_acc_t1.subs(theta1co),eq_acc_t2.subs(theta2co)]
    return q, qdot, qddot
```

In [9]:

```
q, qdot, qddot = trajectory(0, theta1co, theta2co) #test trajectory genrator
```

In [10]:

```
#I've rewritten the Differential equations as:
# ddtheta1_d/dt^2
# eq3 = k1*u(t).diff(t) + k2*v(t).diff(t) - k3*u(t)*v(t) - k4*v(t)**2 - tau1
# ddtheta2_d/dt^2
# eq4 = k5*u(t).diff(t) + k6*v(t).diff(t) - k7*u(t)*v(t) - tau2
```

In [11]:

```
def ddTh_dT(t, Y, th1, th2, th1dot, th2dot, Y3, Y4):
    # pass the current values as the initial values of theta, compute torque 
    m2 = 2.
    m1 = 3.
    I1 = 2.
    I2 = 1.
    L1 = 1.
    g = 9.8
    r1 = 0.5
    r2 = 0.5

    u = th1dot  # u = dtheta1_dt
    v = th2dot # v = dtheta2_dt
    th1 = th1  # theta1
    th2 = th2  # theta2

    tau1 = Y3  # Torque1
    tau2 = Y4  # Torque2

    k1 = I1 + m1 * r1 ** 2 + m2 * r2 ** 2 + m2 * L1 ** 2 + 2 * m2 * L1 * r2 *
    k2 = m2 * r2 ** 2 + m2 * L1 * r2 * np.cos(th2)
    k3 = 2. * L1 * m2 * r2 * np.sin(th2)
    k4 = 1. * L1 * m2 * r2 * np.sin(th2)
    k51 = g * ((m1 * r1 + m2 * L1) * np.cos(th1) + m2 * r2 * g * np.cos(th1 +

    k5 = m2 * r2 ** 2 + m2 * L1 * r2 * np.cos(th2)
    k6 = m2 * r2 ** 2 + I2
    k7 = 1. * L1 * m2 * r2 * np.sin(th2)
    k8 = m2 * r2 * g * np.cos(th1 + th2)

    du_dt = -(((-k51) * k6 + k2 * k8 + k6 * tau1 - k2 * tau2 - k2 * k7 * u **
                            k2 * k5 - k1 * k6))
    dv_dt = -(((-k5) * k51 + k1 * k8 + k5 * tau1 - k1 * tau2 - k1 * k7 * u **
                            (-k2) * k5 + k1 * k6))

#     pdb.set_trace()


    return [Y[0], Y[1], du_dt, dv_dt]
```

Pseudocode for PID control

```
time = 0                          // dt = servo cycle time
eint = 0                          // error integral
qprev = senseAngle                // initial joint angle q
loop
  [qd,qdotd] = trajectory(time) // from trajectory generator

  q = senseAngle                  // sense actual joint angle
  qdot = (q - qprev)/dt           // simple velocity calculation
  qprev = q

  e = qd - q
  edot = qdotd - qdot
  eint = eint + e*dt

  tau = Kp*e + Kd*edot + Ki*eint
  commandTorque(tau)

  time = time + dt
end loop
```

computing torques as $\tau =$

$$\tau = M(\theta)\ddot{\theta} + c(\theta,\dot{\theta}) + g(\theta) \qquad \text{or} \qquad M(\theta)\ddot{\theta} + h(\theta,\dot{\theta}),$$

$$\theta_e = \theta_d - \theta, \dot{\theta}_e = -\dot{\theta}, \text{ and } \ddot{\theta}_e = -\ddot{\theta}.$$

$$\ddot{\theta} = \ddot{\theta}_d + K_d\dot{\theta}_e + K_p\theta_e + K_i\int \theta_e(\mathrm{t})d\mathrm{t}.$$

$$\boxed{\tau = \tilde{M}(\theta)\left(\ddot{\theta}_d + K_p\theta_e + K_i\int \theta_e(\mathrm{t})d\mathrm{t} + K_d\dot{\theta}_e\right) + \tilde{h}(\theta,\dot{\theta}).}$$

In [12]:

```
import matplotlib.pyplot as plt
%matplotlib inline
def pidLoop(t, q, qdot, qddot):

    theta1 = [float(q[0,0])]
    theta2 = [float(q[0,1])]
    theta1dot = [float(qdot[0,0])]
    theta2dot = [float(qdot[0,1])]
    theta1ddot = [float(qddot[0,0])]
    theta2ddot = [float(qddot[0,1])]
```

```
    #list for storing errors in theta2
    theta1_e = [1];
    theta1_edot = [0];
    theta1_eddot = [0];

    theta2_e = [1];
    theta2_edot = [0];
    theta2_eddot = [0];

    m2 = 2.
    m1 = 3.
    I1 = 2.
    I2 = 1.
    L1 = 1.
    g = 9.8
    r1 = 0.5
    r2 = 0.5

    Kp1 = 5
    Kd1 = 4
    Ki1 = 3

    Kp2 = 5
    Kd2 = 3
    Ki2 = 2

    dt = 1. / len(t)
    tau1_l = []
    tau2_l = []

    for i in range(len(t)):
#         print(t[i])
        theta1d = q[i,0]
        theta1dotd = qdot[i,0]
        theta1ddotd = qddot[i,0]

        theta2d = q[i,1]
        theta2dotd = qdot[i,1]
        theta2ddotd = qddot[i,1]

        # calucluate error;

        theta1_e.append(theta1d - theta1[-1])
        theta1_edot.append(theta1dotd - theta1dot[-1])
        theta1_eddot.append(theta1ddotd - theta1ddot[-1])

        theta2_e.append(theta2d - theta2[-1])
        theta2_edot.append(theta2dotd - theta2dot[-1])
        theta2_eddot.append( theta2ddotd - theta2ddot[-1])

        u1 = Kp1 * theta1_e[-1] + Kd1 * theta1_edot[-1] + Ki1 * it.simpson(the


        u2 = Kp2 * theta2_e[-1] + Kd2 * theta2_edot[-1] + Ki2 * it.simpson(the

        inp_pid = np.array([[u1], [u2]])

        td = np.array([[theta1dot[-1]], [theta2dot[-1]]])

        tdd = np.array([[theta1ddot[-1]], [theta2ddot[-1]]])
```

```
        M = np.array([[I1 + m1 * r1 ** 2 + m2 * r2 ** 2 + m2 * L1 ** 2 + 2 * m
                       m2 * r2 ** 2 + m2 * L1 * r2 * np.cos(theta2[-1])],
                      [m2 * r2 ** 2 + m2 * L1 * r2 * np.cos(theta2[-1]), m2 * 

        C = np.array([[-m2 * L1 * r2 * theta2ddot[-1] * np.sin(theta2[-1]),
                       -m2 * L1 * r2 * theta1ddot[-1] * np.sin(theta2[-1]) - m
                           theta2[-1])], [m2 * L1 * r2 *theta1ddot[-1] * np.si

        G = np.array(
            [[g * ((m1 * r1 + m2 * L1) * np.cos(theta1[-1]) + m2 * r2 * g * np
             [ m2 * r2 * g * np.cos(theta2[-1] + theta1[-1])]])


        result1 = M@(tdd+ inp_pid) + C@td + G
        tau1 = result1[0,0]*0.1
        tau2 = result1[1,0]*0.2
        theta_dots = solve_ivp(ddTh_dT,
                               [t[i], t[i] + dt],
                               [theta1[-1], theta2[-1], theta1dot[-1], theta2d
                               args=(theta1[-1], theta2[-1], theta1dot[-1], th
                               )
#         pdb.set_trace()

        theta1.append(it.simpson(theta_dots.y[2,:]))
        theta2.append(it.simpson(theta_dots.y[3,:]))
        theta1dot.append(theta_dots.y[2,-1]);
        theta2dot.append(theta_dots.y[3, -1]);
        theta1ddot.append(theta1dot[-2] - theta1dot[-1])
        theta2ddot.append(theta2dot[-2] - theta2dot[-1])
        tau1_l.append(tau1)
        tau2_l.append(tau2)

#     plt.plot(q[:,0], "-r", label="reference")
#     plt.plot(theta1, "-b", label="actual")
#     plt.legend(loc="upper left")
#     plt.xlabel("time steps")
#     plt.ylabel("theta1 value rads")
#     plt.show()

#     plt.plot(q[:,1], "-r", label="reference")
#     plt.plot(theta2, "-b", label="actual")
#     plt.legend(loc="upper left")
#     plt.xlabel("time steps")
#     plt.ylabel("theta2 value rads")
#     plt.show()


#     plt.plot(tau1_l, "-r", label="tau1")
#     plt.plot(tau2_l, "-b", label="tau2")
#     plt.legend(loc="upper left")
#     plt.xlabel("time steps")
#     plt.ylabel("tau value")
#     plt.show()
    return tau1_l, tau2_l, theta1, theta2
```

In [13]:

```
# Step 1 get desired q,qdot,qddot
# Using the boundary value problem
```

```
# theta1_bvp={'pos':[[0,2],[-pi/4., pi/4.]], 'vel':[[0,2],[0,0]], 'acc':[[0,2]
# theta2_bvp={'pos':[[0,2],[0., pi/2.]], 'vel':[[0,2],[0,0]], 'acc':[[0,2],[0,
# Get coefficients of polynomials using
# theta1co, theta2co = trajGen(theta1_bvp, theta2_bvp)

q = [] #{theta1,theta2}
qdot = [] #{theta1dot,theta2dot}
qddot = [] #{theta1ddot,theta2ddot}

timesteps = 50 #50 timesteps between 0-2
for i in np.linspace(0,2,timesteps):
    q1, qdot1, qddot1 = trajectory(i, theta1co, theta2co)
    q.append(q1)
    qdot.append(qdot1)
    qddot.append(qddot1)



q=np.array(q)
qdot=np.array(qdot)
qddot=np.array(qddot)

tau1_, tau2_, l_theta1_, l_theta2_ = pidLoop(np.linspace(0,2,timesteps), q, qd
```

In [14]:

```
def end_effector(theta1, theta2):
    theta1 = np.array(theta1)
    theta2 = np.array(theta2)
    L1 = 1.
    L2 = 1.
    x1 = L1*np.cos(theta1)
    y1 = L1*np.sin(theta1)
    x2 = x1 + L2*np.cos(theta1+theta2)
    y2 = y1 + L2*np.sin(theta1+theta2)
    return (x1,y1,x2,y2)
```

In [15]:

```
x1,y1,x2,y2 = end_effector(l_theta1_, l_theta2_)
```

In [16]:

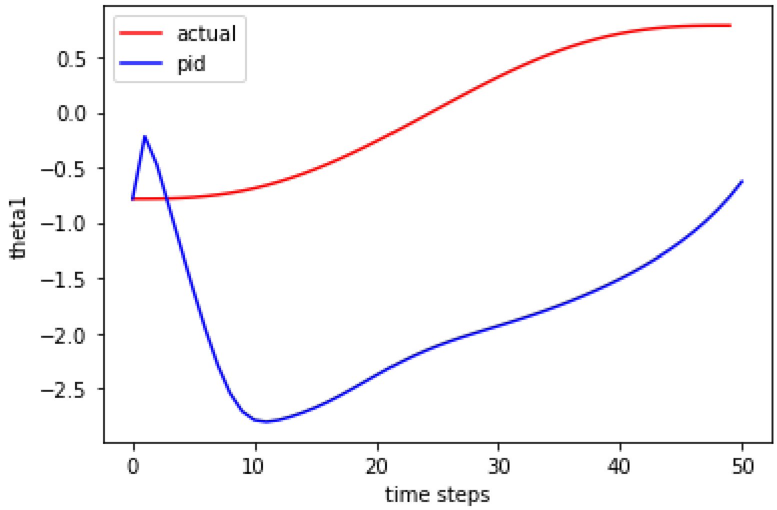
```
# q[:,0]
```

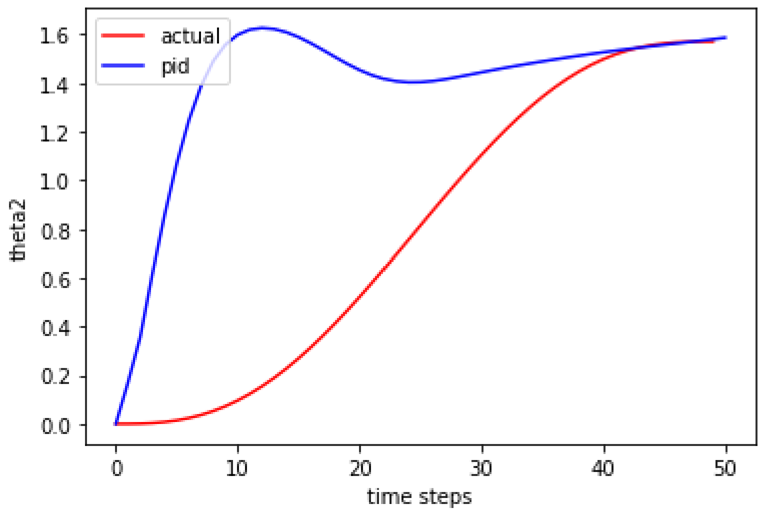
In [17]:

```
# the reference joint angles, actual joint angles,
plt.plot(q[:,0], "-r", label="actual")
plt.plot(l_theta1_, "-b", label="pid")
# plt.legend('theta1 actual, theta1 error', ncol=2, loc='upper left');
plt.legend(loc="upper left")
plt.xlabel("time steps")
plt.ylabel("theta1")
# plt.ylim(-100, 100)
plt.show()

plt.plot(q[:,1], "-r", label="actual")
plt.plot(l_theta2_, "-b", label="pid")
# plt.legend('theta1 actual, theta1 error', ncol=2, loc='upper left');
plt.legend(loc="upper left")
plt.xlabel("time steps")
```

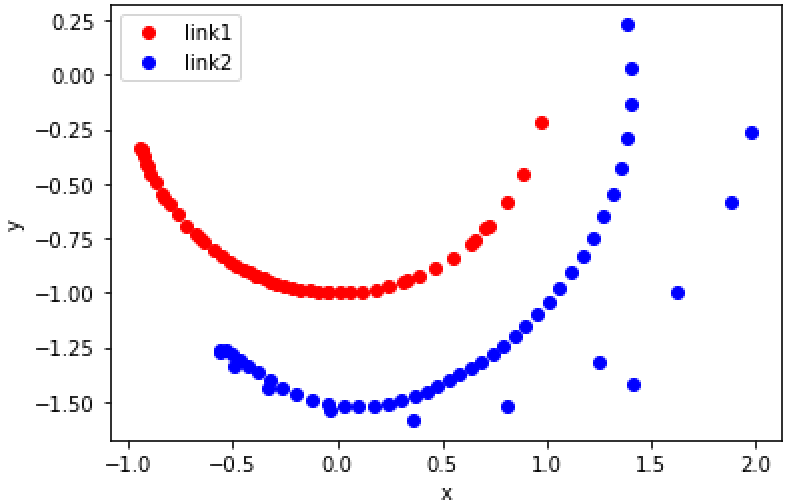
```
plt.ylabel("theta2")
# plt.ylim(-100, 100)
plt.show()
```





In [18]:

```
# end-effector motion in task space
plt.plot(x1, y1, "or", label="link1")
plt.plot(x2, y2, "ob", label="link2")
# plt.plot(x2,y2, "ob", label="link2")
# plt.legend('theta1 actual, theta1 error', ncol=2, loc='upper left');
plt.legend(loc="upper left")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



In [19]:
```
x1[-1]
```

Out[19]: 0.8085727911162902

In [20]:
```
# torques
plt.plot(tau1_, "-r", label="tau1")
plt.plot(tau2_, "-b", label="tau2")
plt.legend(loc="upper left")
plt.xlabel("time steps")
plt.show()
```