Para a estratégia de controle anterior seja viável, é necessário se dispor de todo o estado do sistema, ou seja o veto x, dificilmente isto ocorre. Para mensurar o sistema são utilizados sensores e estes possuem ruídos de observação, vamos nos concentrar apenas em uma observação do tipo

$$y = Cx + Du$$

Que para o caso do nosso problema pode ser uma medida de posição ou velocidade das massas. Caso possuíssemos um sensor em x_2, ou seja $y = x_2$, temos

$$y = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + [0]u$$
$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$
$$D = [0]$$

com u o controle em força na massa m_1, isto é $u = f_1$.
O desafio agora é reconstruir o estado completo, a partir das observações dos sensores. Para tanto é proposto uma dinâmica de um observador na forma

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y})$$

Com $\hat{x}$, a estimação do estado x, $(y - \hat{y})$ o erro do estimador para os sensores observados, e L o ganho de correção do estimador, ou simplesmente o ganho do estimador, com o mesmo número de linhas de dim(x) e o mesmo número de colunas de dim(y), para o caso acima com apenas uma observação da posição da massa m_2,

$$L = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix}$$

l_i são as constantes do ganho do estimador a serem determinadas.

Seja o erro do estimador para os sensores observados

$$\tilde{y} \triangleq y - \hat{y}$$

E o erro de estimação dos estados dado por

$$\tilde{x} \triangleq x - \hat{x}$$

Considerando o controle na observação nulo, isto é, a matriz D é nula, o que geralmente é o caso na maioria dos sistemas, temos

$$\hat{y} = C\hat{x}$$

$$\tilde{y} = Cx - C\hat{x}$$
$$\tilde{y} = C(x - \hat{x})$$
$$\tilde{y} = C\tilde{x}$$

Seja a derivada do erro de estimação dos estados

$$\dot{\tilde{x}} = \dot{x} - \dot{\hat{x}}$$

Aplicando a esqução dos estados e do estimador dos estados temos

$$\dot{\tilde{x}} = Ax + Bu - A\hat{x} - Bu - L(y - \hat{y})$$
$$\dot{\tilde{x}} = A(x - \hat{x}) - L(y - \hat{y})$$
$$\dot{\tilde{x}} = A\tilde{x} - L\tilde{y}$$
$$\dot{\tilde{x}} = A\tilde{x} - LC\tilde{x}$$
$$\dot{\tilde{x}} = (A - LC)\tilde{x}$$

Que recai no mesmo problema anteriormente, de encontrar os ganhos do controlador para uma dada dinâmica

De forma análoga temos que

$$det(sI - A + LC) = 0$$

que são os polos da dinâmica do erro do observador. Pode-se então forçar que $det(sI - A + LC)$ coincidam com uma dinâmica desejada a ser projetada, fazendo

$$det(sI - A + LC) = (s - p_1)(s - p_2)(s - p_3) \dots (s - p_i)$$

na equação acima os valores dos polos são conhecidos (projetados), ficando apenas como incógnitas os valores das constantes l_i do ganho do estimador.

Para o problema anterior

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

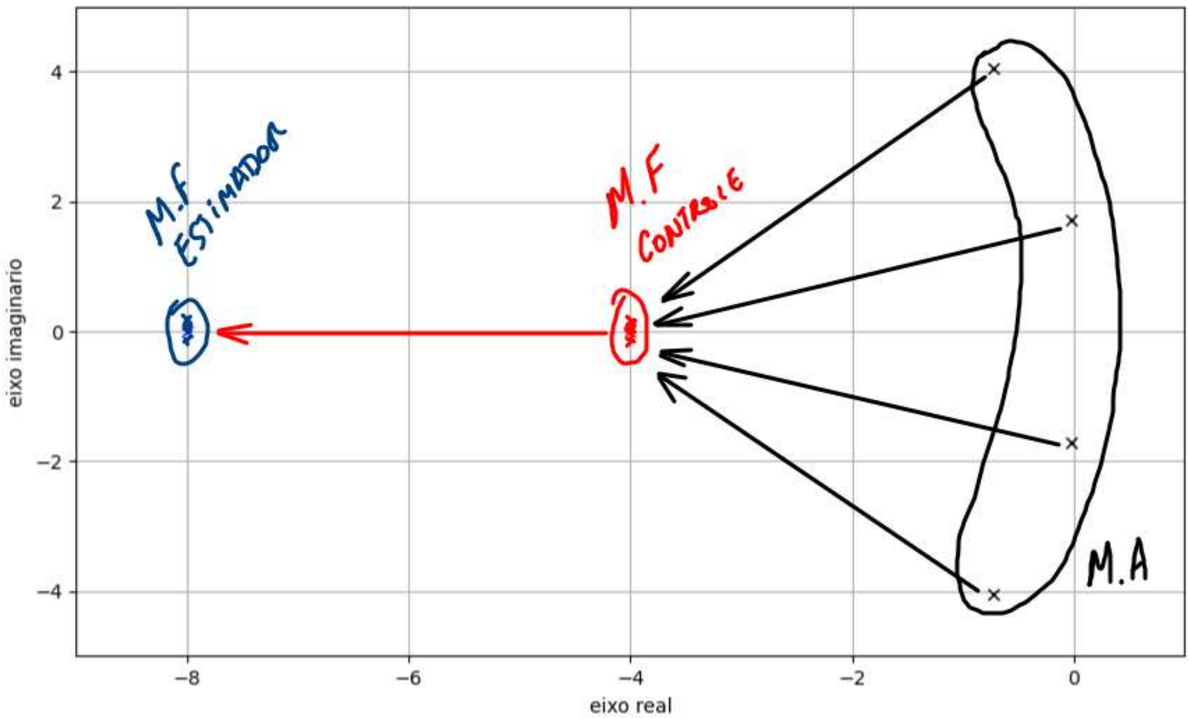
com

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix}$$
$$B = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$
$$D = [0]$$
$$K = \begin{bmatrix} 105.92 & -64.72 & 29 & 17.08 \end{bmatrix}$$
$$N = 51.2$$
$$r = 0.5$$

Com esses ganhos os polos de malha fechada do controlador estão próximos de -4, com pequenos erros devido ao arredondamento dos ganhos k_i do controlador. Que pode ser verificado com

```
import numpy as np
A = np.array([[0,0,1,0],[0,0,0,1],[-10,5,-0.5,0.5],[10,-10,1,-1]])
B = np.array([[0],[0],[0.5],[0]])
K = np.array([[105.92,-64.72,29,17.08]])
print(np.linalg.eigvals(A-B@K))
array([-4.00064257+0.j          , -4.          +0.00064257j,
       -4.          -0.00064257j, -3.99935743+0.j          ])
```

Devemos escolher os polos de malha fechada da dinâmica do erro do estimador, de forma que esta seja bem mais rápida que a dinâmica em malha fechada do sistema, para tanto devemos afastar ainda mais os polos do eixo imaginário em direção ao menos infinito.



Para o caso do exercício, vamos escolher afastar os polos por um fator de 2x, conforme figura acima, ou seja colocar todos em -8, portanto

$$det(sI - A + LC) = (s+8)(s+8)(s+8)(s+8)$$

$$det\left(\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix} + \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}\right) = (s+8)(s+8)(s+8)(s+8)$$

Usando python para facilitar o cálculo de $det(sI - A + LC)$

```
import numpy as np
import sympy as sp
A = np.array([[0,0,1,0],[0,0,0,1],[-10,5,-0.5,0.5],[10,-10,1,-1]])
C = np.array([[0,1,0,0]])
s = sp.symbols('s')
l1,l2,l3,l4 = sp.symbols(['l1','l2','l3','l4'])
L=[[l1],[l2],[l3],[l4]]
print(sp.collect(sp.Matrix(s*np.identity(4)-A+np.dot(L,C)).det(),s))
-5.0*l1 + 5.0*l2 + 10.0*l3 + 10.0*l4 + 1.0*s**4 + s**3*(1.0*l2 + 1.5) + s**2*(1.5*l2 + 1.0*l4 + 20.0) + s*(10.0*l1 + 10.0*l2
+ 1.0*l3 + 0.5*l4 + 5.0) + 50.0
```

Usando python para o cálculo de $(s + 8)(s + 8)(s + 8)(s + 8)$

```
import sympy as sp
s = sp.symbols('s')
p1=-8; p2=-8; p3=-8; p4=-8
print(sp.expand((s-p1)*(s-p2)*(s-p3)*(s-p4)))
s**4 + 32*s**3 + 384*s**2 + 2048*s + 4096
```

igualando as expressões

$$s^4 + (l_2 + 1.5)s^3 + (1.5l_2 + l_4 + 20)s^2 + (10l_1 + 10l_2 + l_3 + 0.5l4 + 5)s - 5l_1 + 5l_2 + 10l_3 + 10l_4 + 50 = s^4 + 32s^3 + 384s^2 + 2048s + 4096$$

Podemos então determinar os coeficientes, igualando as constantes do polinômio e resolvendo o sistema de equações lineares da mesma forma que feito anteriormente

$$L = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix} = \begin{bmatrix} 143.6 \\ 30.5 \\ 142.9 \\ 318.3 \end{bmatrix}$$

Simulando o controle para uma referência em posição da massa 2 igual para 0.5m, com estimador de estados

```
import numpy as np
def RK4(f): return lambda x, u, dt:(lambda dx1:(lambda dx2:(lambda dx3:(lambda

dx4:(dx1+2*dx2+2*dx3+dx4)/6)(dt*f(x+dx3,u)))(dt*f(x+dx2/2,u)))(dt*f(x+dx1/2,u)))(dt*f(x,u))
dx = RK4(lambda x, u: A@x + B@u)
dx_est = RK4(lambda x_est, u: A@x_est + B@u + L@(y-y_est))

m1=2.0; m2=1.0; k1=10.0; k2=10.0; c1=0; c2=1.0;

A = np.array([[0,0,1,0],[0,0,0,1],[-(k1+k2)/m1,k2/m1,-(c1+c2)/m1,c2/m1],[k2/m2,-k2/m2,c2/m2,-c2/m2]])
B = np.array([[0],[0],[1/m1],[0]])
C = np.array([[0,1,0,0]])
D = np.array([[0]])
K = np.array([[105.92,-64.72,29,17.08]])
N = np.array([[51.2]])
L = np.array([[143.6],[30.5],[142.9],[318.3]])

t, tf, dt, u, x, r = 0, 10, .01, np.array([[0]]), np.array([[0.5],[1],[0],[0]]), np.array([[.5]])
X, U, T = x, u, t

x_est = np.array([[0],[0],[0],[0]])
X_est = x_est

for i in range(int((tf-t)/dt)):
    t, x = t + dt, x + dx(x,u,dt)
    y, y_est = C@x, C@x_est
    x_est = x_est + dx_est(x_est,u,dt) # estimação do estado
    u = N@r-K@x_est
    X, U, T = np.append(X,x,axis=1), np.append(U,u,axis=1), np.append(T,t)
    X_est = np.append(X_est,x_est,axis=1)

import matplotlib.pyplot as plt
plt.plot(T,X[1,:],'k',T,X_est[1,:],'b')
plt.xlabel('tempo (s)')
plt.legend(['x2','x2_estimado'])
plt.grid(True)
plt.show()
```