Robot Control Exercise 4: State Feedback Control

Nicholas Shindler, shindler25@gmail.com

December 3, 2019

1 PID Control Stabilization

1) $[Matlab]$

2) Stabilizing in the initial state of $r = 0$ we determined a PID calibration of $k_p = 9.86;\ k_i = 0.048;\ k_d = 3.87$

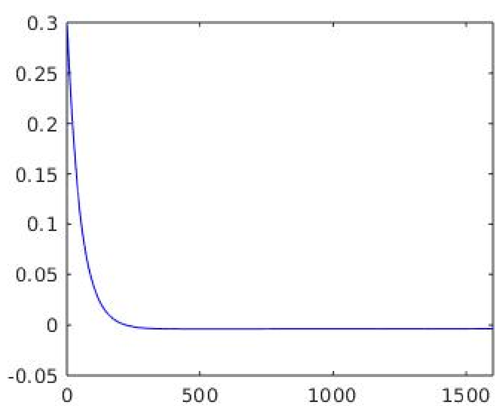

Figure 1: PID calibrated Theta(t) curve

2 State Feedback Stabilization

2.1 Calculate gain matrix

```
syms k1 k2 lambda Theta dTheta;


l = 1; % length of pendulum
g = 9.81; % gravity
c = 1; % control constant
K = [k1 k2]; % gain matrix <-- what we want to know;
x = [Theta;dTheta];
u = -K*x;

% Using the function: x_dot = A*x - B*K*x
%   solve for the vector k by obtaining the eigenvalues.

% ddTheta = (g/l)*sin(Theta) + c*u;
% by small angle theorem sin(Th) --> Th
% ddTheta = (g/l)*Theta + c*u;

% calculate A and B values plugging in the above equation
A = [0,1;(g/l),0];
B = [0;c];

% calculate THETA dot
THETA = A*x + B*u;
% THETA:
% [                 dTheta,                   dTheta]
% [ (981*Theta)/100 - k1*x, (981*Theta*x)/100 - k2]

I = eye(2); % the identity matrix

eigValMat = (A - B*K)-lambda*I; % calculate the matrix to be solved to obtain eigenvalues
% eigValMat:
% [      -lambda,              1]
% [ 981/100 - k1, - k2 - lambda]
det_eigMat = det(eigValMat); % take determinite
% det_eigMat:
% lambda^2 + k2*lambda + k1 - 981/100
%
% given f(lam) = (lam - lam1)(lam - lam2)
% lam1 = -1, lam2 = -2
% equivicate to determinite:
% lambda^2 + 3*lambda + 2 ==> lambda^2 + k2*lambda + k1 - 981/100
% k1 = 2+9.81 = 11.81;
% k2 = 3
```

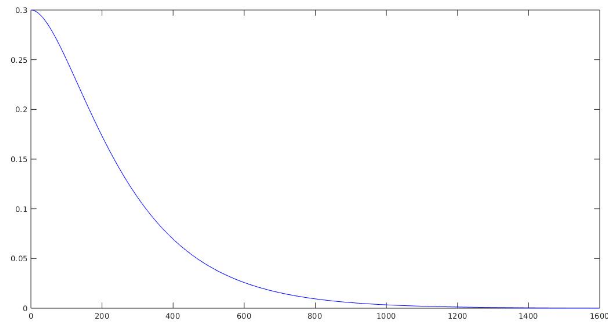

Figure 2: Plotting Theta(t) based on $u = -Kx$

2.2 Segway - linear state space

```
%% Problem 2.2 %%
syms TH dd_X ddTH u;

% constants
g = 9.81;
l = 1;
m = 1;
M = 1;

% small angle approx
sin_TH = TH;
cos_TH = 1;
dTH2 = 0;

ddTH = (g/l)*sin_TH+(1/l)*dd_X*cos_TH;
ddX = (1/(M+m))*(u + m*l*ddTH*cos_TH - m*l*dTH2*sin_TH);
% (9.81*TH)/2 + dd_X/2 + u/2
ddX = ddX*2 - dd_X;
% 9.81 * TH  + u
```

2.3 Segway - controlablity

```
%% Problem 2.3 %%
% r = [B AB ... A^1 B] * [u_n-1;...;u_1;u_0]
% rank(r) = n

syms x dx th dth g_;
A = [0,1,0,0;0,0,9,0;0,0,0,1;0,0,2*g_,0];
```

```
x_ = [x;dx;th;dth];
B = [0;1;0;1];

r = [B A*B A*A*B A*A*A*B];
n = rank(r);
% n === 4
% therefore controlable...
```

2.4 Segway - pole placement

```
%% Problem 2.4 %%
syms k1 k2 k3 k4 lambda;
I = eye(4); % the identity matrix
K = [k1,k2,k3,k4];
u = -K*x_;
fx = A*x_+B*u;
% fx is equal to:
%                                             dx
%         9*th - dth*k4 - dx*k2 - k3*th - k1*x
%                                           dth
% (981*th)/50 - dth*k4 - dx*k2 - k3*th - k1*x

eigValMat = (A - B*K)-lambda*I; % calculate the matrix to be ...
    solved to obtain eigenvalues
% eigValMat is equal to:
%[ -lambda,             1,           0,              0]
%[     -k1, - k2 - lambda,      9 - k3,            -k4]
%[       0,             0,     -lambda,              1]
%[     -k1,           -k2, 981/50 - k3, - k4 - lambda]
det_eigMat = det(eigValMat); % take determinite
% 9*k1 - 2*g_*k1 + 9*k2*lambda - 2*g_*lambda^2 + k1*lambda^2 + ...
    k2*lambda^3 + k3*lambda^2 + k4*lambda^3 + lambda^4 - ...
    2*g_*k2*lambda
%
some = det(lambda*I-A);
%lambda = sqrt(2*g_);
eig = -1;
l1=eig;l2=eig;l3=eig;l4=eig;
f_lam=expand((lambda-l1)*(lambda-l2)*(lambda-l3)*(lambda-l4));
%disp(f_lam);
% lambda^4 - 4*lambda^3 + 6*lambda^2 - 4*lambda + 1
%
% given f(lambda) = lambda^4 - 4*lambda^3 + 6*lambda^2 - ...
    4*lambda + 1
% lambda^4 + (k2+k4)lambda^3 + (k3+k1-2*g)lambda^2 + (9*k2 - ...
    2*g*k2)lambda + (9*k1 - 2*g*k1)
% -4 = k2+k4
% 6 = (k3+k1-2*g)
% -4 = (9*k2 - 2*g*k2)
% 1 = (9*k1 - 2*g*k1)

K_ = [0,1,0,1,-4;1,0,1,0,6+2*g_;0,9-2*g_,0,0,-4;9-2*g_,0,0,0,1];
```

```
k_rd = rref(K_);
%disp(k_rd);
% [ 1, 0, 0, 0,                          -1/(2*g_ - 9)]
% [ 0, 1, 0, 0,                           4/(2*g_ - 9)]
% [ 0, 0, 1, 0, -(- 4*g_^2 + 6*g_ + 53)/(2*g_ - 9)]
% [ 0, 0, 0, 1,           -(8*(g_ - 4))/(2*g_ - 9)]

k1 =   -0.0942;
k2 =    0.3766;
k3 =   25.7142;
k4 =   -4.3766;

% alt:
eig = -1;
l1=eig;l2=eig;l3=eig*2;l4=eig*2;
f_lam=expand((lambda-l1)*(lambda-l2)*(lambda-l3)*(lambda-l4));
%disp(f_lam);
% lambda^4 + 6*lambda^3 + 13*lambda^2 + 12*lambda + 4
l_4 = 1;
l_3 = 6;
l_2 = 13;
l_1 = 12;
l_0 = 4;

K_ = [0,1,0,1,l_3;1,0,1,0,l_2+2*g;0,9-2*g,0,0,l_1;9-2*g,0,0,0,l_0];
k_rd = rref(K_);
%disp(k_rd);
% [ 1, 0, 0, 0,                      -4/(2*g_ - 9)]
% [ 0, 1, 0, 0,                     -12/(2*g_ - 9)]
% [ 0, 0, 1, 0, (4*g_^2 + 8*g_ - 113)/(2*g_ - 9)]
% [ 0, 0, 0, 1,        (6*(2*g_ - 7))/(2*g_ - 9)]

k1 =   -0.3766;
k2 =   -1.1299;
k3 =   32.9966;
k4 =    7.1299;

% alt:
eig = -1;
l1=eig;l2=eig*2;l3=eig*3;l4=eig*4;
f_lam=expand((lambda-l1)*(lambda-l2)*(lambda-l3)*(lambda-l4));
%disp(f_lam);
% lambda^4 + 10*lambda^3 + 35*lambda^2 + 50*lambda + 24
l_4 = 1;
l_3 = 10;
l_2 = 35;
l_1 = 50;
l_0 = 24;

K_ = [0,1,0,1,l_3;1,0,1,0,l_2+2*g;0,9-2*g,0,0,l_1;9-2*g,0,0,0,l_0];
k_rd = rref(K_);
%disp(k_rd);
% [ 1, 0, 0, 0,                      -24/(2*g_ - 9)]
% [ 0, 1, 0, 0,                      -50/(2*g_ - 9)]
% [ 0, 0, 1, 0, (4*g_^2 + 52*g_ - 291)/(2*g_ - 9)]
% [ 0, 0, 0, 1,          (20*(g_ - 2))/(2*g_ - 9)]
k1 =   -2.2599;
```

```
k2 =   −4.7081;
k3 =   56.8799;
k4 =   14.7081;
```

This was tested using 3 sets of pole placement values to get a good result. The initial placement has not been included in the plot as it did not converge. the red line represents the 2^{nd} trial ($\lambda = [1, 1, 2, 2]$) and the green represents the 3^{rd} ($\lambda = [1, 2, 3, 4]$).

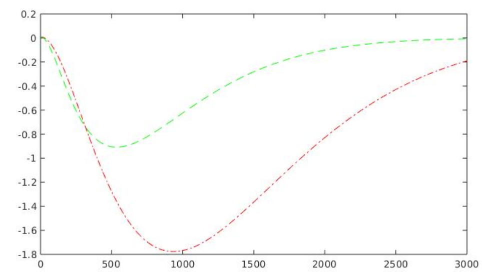

Figure 3: Plotting x(t) based on $u = -Kx$

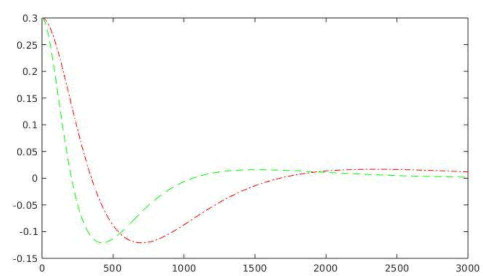

Figure 4: Plotting Theta(t) based on $u = -Kx$

2.5 Segway - pole placement

Using the best K from the previous section we determined that x_0 could be scaled by 3.33 and be able to converge.

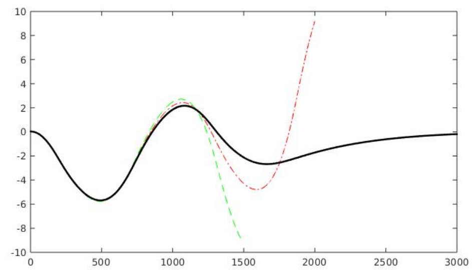

Figure 5: x(t) based on scale values 3.35(green), 3.34(red), 3.33(black)

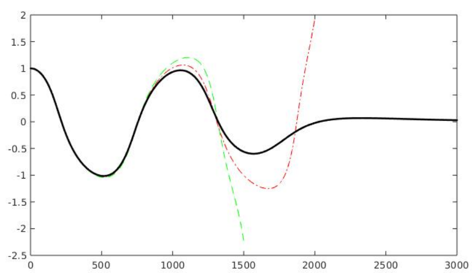

Figure 6: Theta(t) based on scale values 3.35(green), 3.34(red), 3.33(black)

3 Observer Design

3.1 Observably

```
% Observability
%% 3.1
% the system is observable if the rank of the observability ...
    matrix is such that
syms g_;
A = [0,1,0,0;0,0,9,0;0,0,0,1;0,0,2*g_,0];
C = [1 0 0 0];
O = [C;C*A;C*(A*A);C*(A*A*A)];
% O =
%    [ 1, 0, 0, 0]
%    [ 0, 1, 0, 0]
%    [ 0, 0, 9, 0]
%    [ 0, 0, 0, 9]
% rank(O) = 4
```

```
% n = 4 == rank(O) = 4
% Observable? yes.
```

3.2 Observer Gain Matrix

```
%% 3.2
syms x dx th dth g_ x_hat L u;
A = [0,1,0,0;0,0,9,0;0,0,0,1;0,0,2*g_,0];
A_ = [0,1,0,0;0,0,9,0;0,0,0,1;0,0,2*9.81,0];
B = [0;1;0;1];
C = [1 0 0 0];
x_ = [x;dx;th;dth];
y = C*x_;


Dx_hat = A*x_hat + B*u + L*(y-C*x_hat);

syms k1 k2 k3 k4 lambda l1 l2 l3 l4 e_;
I = eye(4); % the identity matrix
K = [k1,k2,k3,k4];
u = -K*x_;
Dx = A*x_+B*u;

e = (x_-x_hat);
L = [l1;l2;l3;l4];
De = (A-L*C)*e_;
%(A-L*C) =
% [ -l1, 1,     0, 0]
% [ -l2, 0,     9, 0]
% [ -l3, 0,     0, 1]
% [ -l4, 0, 2*g_, 0]

p = det((A-L*C)-lambda*I);
% ACTUAL: p = 9*l4 - 2*g_*l2 + 9*l3*lambda - 2*g_*lambda^2 + ...
    l1*lambda^3 + l2*lambda^2 + lambda^4 - 2*g_*l1*lambda
% DESIRED: (lambda + 1)^4 = lambda^4 + 4*lambda^3 + 6*lambda^2 + ...
    4*lambda + 1
L_ = [1,0,0,0,4;0,-2*g_,0,0,6;-2*g_,0,9,0,4;0,-2*g_,0,9,1];
l_rd = rref(L_);
% [ 1, 0, 0, 0,              4]
% [ 0, 1, 0, 0,          -3/g_]
% [ 0, 0, 1, 0, (8*g_)/9 + 4/9]
% [ 0, 0, 0, 1,           -5/9]
l1 = 4;
l2 = -3/9.81;
l3 = (8*9.81)+4/9;
l4 = -5/9;
```

3.3 Scaling

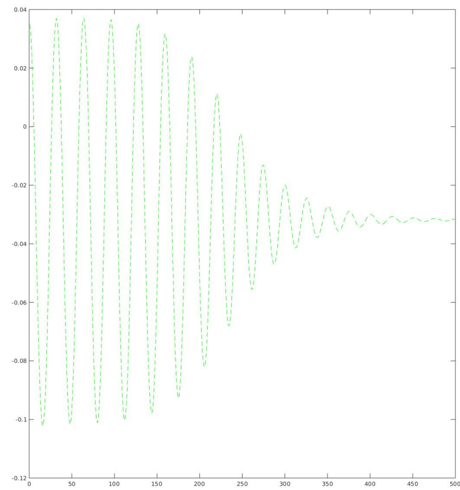

Figure 7: x(t) based on scale value 3.6

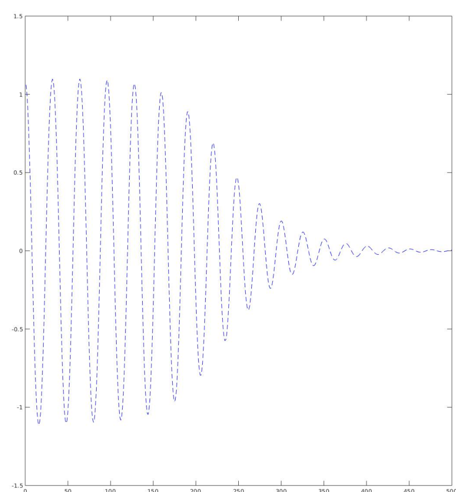

Figure 8: Theta(t) based on scale value 3.6

Through testing we determined that the maximal scaling factor was about 3.6.

3.4 Initial Scaling

Setting only $\hat{x}_0$ to be scaled resulted in a much lower potential scaling factor of only about 2.6.

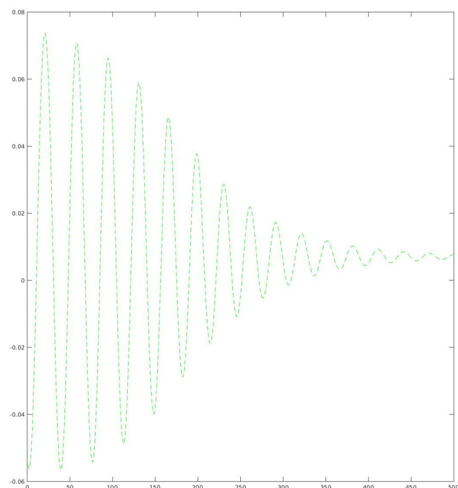

Figure 9: x(t) based on scale value 2.6 of only $\hat{x}_0$

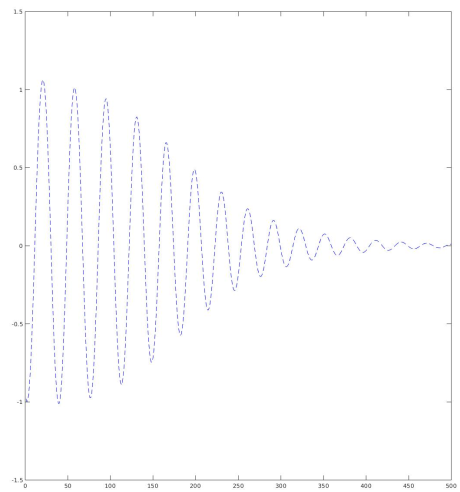

Figure 10: Theta(t) based on scale value 2.6 of only $\hat{x}_0$

3.5 Tilt measurement

By changing the C matrix I needed to change the P matrix to a value that would converge (-36 -36 1 1).

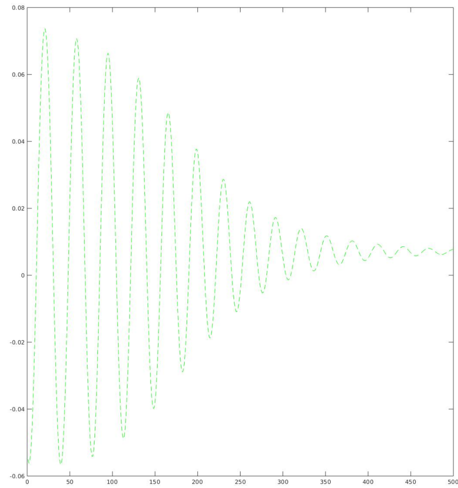

Figure 11: x(t) with gyro as part of C matric

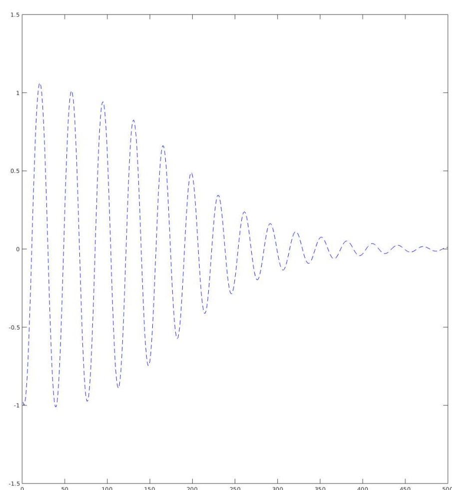

Figure 12: Theta(t) with gyro as part of C matric

I was able to get a higher scaling factor for a global initial scale in this situation.

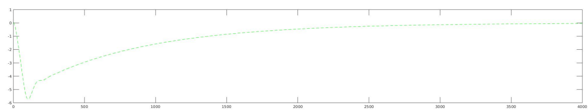

Figure 13: x(t) with gyro as part of C matric scaled by 4.33

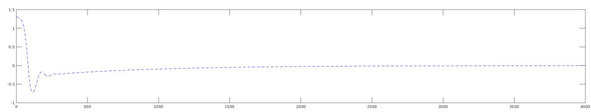

Figure 14: Theta(t) with gyro as part of C matric scaled by 4.33

Setting only $\hat{x}_0$ to be scaled resulted in a much lower potential scaling factor of only about 2.6.

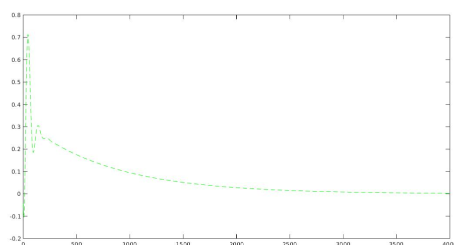

Figure 15: x(t) with gyro as part of C matric scaled by 1.25 of only for $\hat{x}_0$

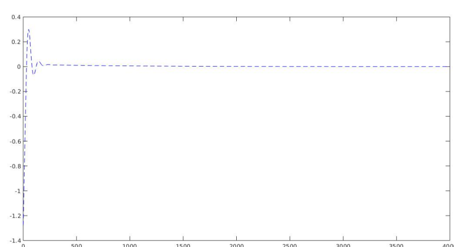

Figure 16: Theta(t) with gyro as part of C matric scaled by 1.25 of only for $\hat{x}_0$

Setting only $\hat{x}_0$ to be scaled resulted in a much much lower potential scaling factor of only about 1.25.

3.6 Reference Gain

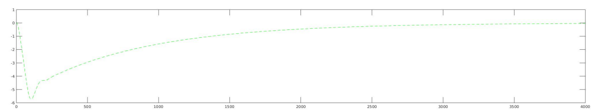

Figure 17: x(t) with gain matrix – r(t) = 1

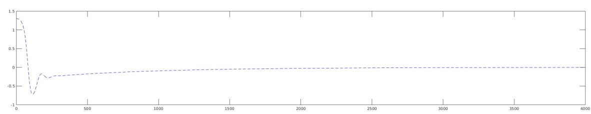

Figure 18: Theta(t) with gain matrix – r(t) = sin(t)

4 Code

`inv_pend_test.m` - inverse pendulum test with u calculated with pid or -Kx

`segway_test.m` - segway with control calculated u, with tests for max scaled initial state

`segway_gains.m` - segway with gain matrix added

`segway_giro_test.m` - segway with giro controller added in

`segway_obs_test.m` - test function for segway using observer calculations

`segway_test_initial_state.m` - segway test script for initial scaling using the control function

`observer.m` - calculations for observably

`lin_space_rep_seg.m` - calculations for segway control

`lin_space_test.m` - test for inverse pendulum control `lin_space_rep.m` - calculations for inverse pendulum control

`PIDController` - pid class for inverse pendulum