EEE4119 PROJECT REPORT

May 14, 2022

UCT Department of Electrical Engineering

AUTHOR:

Dylan White *WHTDYL001*

1 Introduction

This report outlines the process of designing a control system for warehouse organisational robot. This robot consists of a two link arm mounted to a sliding base as displayed below.

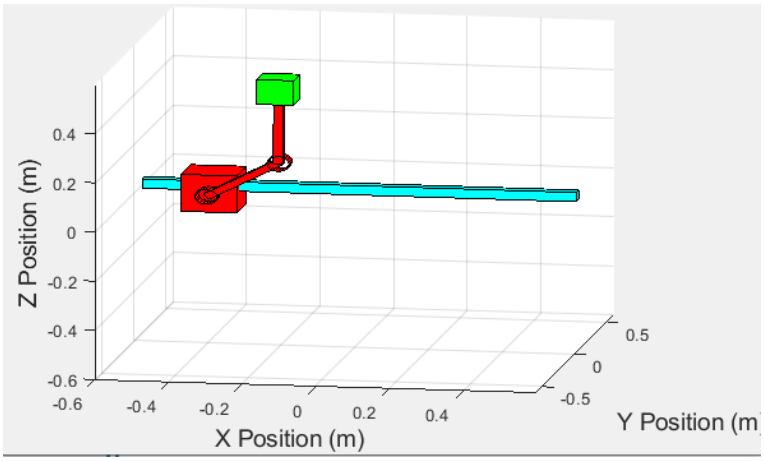

Figure 1. Simulated Warehouse Package Manipulator

The goal for this system is to be able to give two references to the control system. The first being the location of the box to be fetched, the second being the location to where the box should be taken. This project does not involve the design of the box detector system, this is aleady completed.

As shown in the image above, the system is constrained to a single plane and thus the references will consist only of an x and a z position. An additional requirement is that all manipulated packages need to remain upright for the duration of their manipulation. Because of this confinement to a single plane, this system has three degrees of freedom can can be totally described by:

- the position of the slider: x
- the angle of the first link: θ_1
- the angle between the first and the second link: θ_2

This report outlines the process of modelling the robot as well as the process of designing a suitable control system achieve the described system behaviour

2 Theory

2.1 Legrangian Mechanics

In order to design a control system, the dynamics of the robot need to be well described and modelled. The dynamics of the system were identified using legrangian mechanics, this takes an energy based approach to describing mechanics and uses generalised coordinates. This is ideal for our 3 degree of freedom system where we would much rather work with our 3 generalised coordinates (x, θ_1, θ_2)

If our generalised coordinates are described by a vector q. The Lagrangian equation can be expressed as [1]

$$L(q, \dot{q}) = T(q, \dot{q}) - V(q, \dot{q}) \tag{1}$$

where T is the sum of all the kinetic energy in the system, and V is the sum of all the potential energy in the system.This equation is very powerful when combined with the Lagrange dynamic equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{2}$$

and enables us to calculate the equation of motion for each generalised coordinate $\ddot{q}_i$

2.2 The Manipulator Equation

The equations of motion calculated through 2 can quickly become unwieldy, thus a more compact way of storing the dynamics of a constrained system is introduced in the "Manipulator Equation" [1]:

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) = \tau \tag{3}$$

In this equation, τ would be a vector of input force (or torques) of the same order as number of generalised coordinates. The method of calculating each of the terms in the above is briefly outlined below

2.2.1 Mass Matrix

The mass matrix contains the coefficients of the accelerations of each generalised coordinate, it thus contains inertial information in a broad sense.

Practically, the mass matrix is the hessian matrix, square matrix of second order partial derivatives of a function [3]. In our case the scalar valued function is the sum of all kinetic energy in the system and the partial derivatives are taken with regards to the first derivatives of our generalised coordinates. The mass matrix is thus described by

$$M_{i,j} = \frac{\partial T}{\partial \dot{q}_i \partial \dot{q}_j} \tag{4}$$

2.2.2 Coriolis Matrix

The Coriolis matrix deals encapsulates the information on all apparent centrifugal and coriolis forces present in the system, this matrix can be calculated as follows [4]

$$C = \dot{M}\dot{q} - \frac{\partial T}{\partial q} \tag{5}$$

2.2.3 Gravity Matrix

The gravity matrix (vector), contains information on the constant forces exerted on the system, Thus it is simply the jacobian of the potential energy with regards to each generalised coordinate:

$$G = \frac{\partial V}{\partial q} \tag{6}$$

2.3 State Space Control

2.3.1 Linear Quadratic Regulator (LQR)

The linear quadratic regulator algorithm is a an algorithm which optimises state gains in a state space control system [2] in accordance with the following cost function

$$J(u) = \int_0^\infty (x^T Q x + u^T R u) dt \tag{7}$$

In this cost function, Q is a diagonal matrix which acts as a penalisation on certain states and R is a factor which acts as a penalisation on control action.

This algorithm additionally requires that the controlled system be linear.

3 System Modelling

3.1 Equations of Motion

In order to determine the equations of motion to model the system. The manipulator equation parameters are calculated as described above. The variables are as shown below:

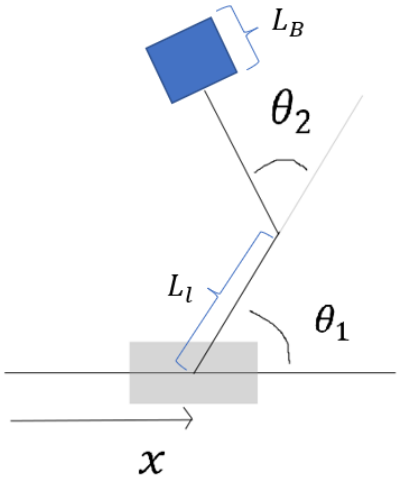

Figure 2. Illustration of Simplified Manipulator

First, the link and box positions and velocities are calculated (in the inertial frame) :

$$rs = \begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} \quad r1 = \begin{bmatrix} x + \frac{cos(\theta_1)}{8} \\ 0 \\ \frac{sin(\theta_1)}{8} \end{bmatrix} \quad r2 = \begin{bmatrix} x + \frac{cos(\theta_1+\theta_2)}{8} + \frac{cos(\theta_1)}{4} \\ 0 \\ \frac{sin(\theta_1+\theta_2)}{8} + \frac{cos(\theta_1)}{4} \end{bmatrix} \quad rB = \begin{bmatrix} x + \frac{11cos(\theta_1+\theta_2)}{20} + \frac{cos(\theta_1)}{4} \\ 0 \\ \frac{11sin(\theta_1+\theta_2)}{20} + \frac{cos(\theta_1)}{4} \end{bmatrix} \tag{8}$$

$vs = \frac{\partial rs}{q} dq \quad v1 = \frac{\partial r1}{q} dq \qquad v2 = \frac{\partial r2}{q} dq \qquad vB = \frac{\partial rB}{q} dq$

with q being the vector of $[x, \theta_1, \theta_2]$ dq naturally being the derivative of each of these generalised coordinates.

At this point, Kinetic, T, and potential, V energies are calculated as follows:

$$\begin{aligned} Ts &= 0.5 \times m_s \times [vs]^T \times vs \\ T1 &= 0.5 \times m_L \times [v1]^T \times v1 + 0.5 \times [\omega_1]^T \times I_L \times \omega_1 \\ T2 &= 0.5 \times m_L \times [v2]^T \times v2 + 0.5 \times [\omega_2]^T \times I_L \times \omega_1 \\ TB &= 0.5 \times m_B \times [vB]^T \times vB + 0.5 \times [\omega_2]^T \times I_B \times \omega_1 \\ Vs &= 0 \\ V1 &= m_L \times g \times [r1]^T \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ V2 &= m_L \times g \times [r2]^T \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ VB &= m_L \times g \times [rB]^T \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

These energy terms can be summed together and used to calculate the manipulator equation matrices as described in section 2.2.

3.2 Drag Coefficient Identification

The package handler system will have friction present in its the motion of the rotation as well as the motion of the slider on the rail. For the purposes of this project, the friction is assumed to be linear viscous damping where $F_{drag} = -b\dot{x}$. Additionally, we have that the damping factor, b, is the same across each element.

The presence of this damping means that the manipulator equation needs to be slightly modified as follows:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau - b\dot{q} \tag{9}$$

Looking at the above it is clear that the damping factor can be calculated by rearranging the above equation for the damping constant. This can be done for each equation of motion individually or as a collective using matrix manipulation.

A wise method to achieve the following is to solve the equations of motion calculated with equation 9 for steady state conditions.

The above process is done experimentally by simply allowing the package handler system to drop from a reference position with no input forces and solving for the vector of b. Due to the nature of this being a simulated system the output of these calculations immediately yields a result for b. The calculated damping coefficient value was calculated to be **b=0.9412**.

4 Design

4.1 Feedback Linearisation of System

Control methods are designed for linear systems, it is clear from 9 that our input to the system is the torque vector τ and that this is not a linear system. We need a way to linearly control the position of the end effector, y which is a function $f(q)$. Thus, we can differentiate this function until we get the second derivative $\ddot{q}$ which we can see to be present in 9

$$\begin{aligned} y &= f(q) \\ \dot{y} &= J\dot{q} \\ \ddot{y} &= \dot{J}\dot{q} + J\ddot{q} \end{aligned}$$

Where J is the jacobian $\frac{\partial f(q)}{q}$. We can thus assign a "sudo input" ,w to $\ddot{y}$. Thus our system will appear from a control standpoint to be a double integrator. We can then map our sudo input to our actual input as follows:

$$\tau = M\left[\frac{J}{w - \dot{w}\dot{q}}\right] + C + G + b\dot{q} \tag{10}$$

4.2 Position Control for Navigating to Box

With a linearised system, a suitable control system needs to be desired. This controller is designed using state space methods. The first thing that needs to be chosen is the states to be controlled. Since the manipulator is constrained to the xz plane, the x and z position of the end affector should naturally be states. However, these cannot be the only states as this is a three degree of freedom system and thus we cannot solve for a particular position with only two states (there are multiple orientations of thr manipulator that will result with the end affector in any given position). Thus we add a third state being the angle of the second link to the horizontal axis. This is a beneficial state to control as it is also a system requirement that manipulated packages are help upright throughout their motion. Thus our state and output vectors are as follows:

$$X = \begin{bmatrix} x \\ y \\ \theta_{sum} \\ \dot{x} \\ \dot{y} \\ \dot{\theta}_{sum} \end{bmatrix} \quad Y = \begin{bmatrix} x \\ y \\ \theta_{sum} \end{bmatrix}$$

Where $\theta_{sum} = \theta_1 + \theta_2$ and all states are functions of our generalised coordinates q.

Since our system now appears as a simple double integrator, our state space matrices are very simple and exist to extract our desirable states

State gains were chosen using the LQR algorithm 7 such that they minimised the force over time of our motors as illustrated by the cost function:

$$J = \int_0^{t_{final}} [\tau_1 + \tau_2 + F_1] dt$$

. In choosing Q and R values there is some trade off between system response time and control action. Thus there is naturally some iterative design in the optimal choice of these values. For this project, the Q matrix was chosen so that each state was equally penalised in the LQR algorith. This is definitely a design decision around which there can be optimisation as there are definitely some states that require more control action than others to meet specifications.

4.3 Position Control of Box with Known Weight

The process for controlling a box of known weight was very similar to controlling the end affector of the robot. In this case, the energies of the box are included in the calculation of the manipulator matrices which affect the non-linear mapping of the feedback linearisation as shown in equation 10.

Other than including the box mass in the matrix calculations, the only difference to the end affector controller is that we are controlling the position of the box's center of mass half a box length away from the position of the end affector. Thus the X and Z values of our control states are slightly offset.This is visualised in figure 2

4.4 Position Control Of Box With Unknown Weight

4.4.1 Mass Estimator

In order to accurately control the box, the mass of the box needs to be calculated. This can be done in a number of ways, the simplest way is to note that the manipulator equation matrices are a function of the mass of the box, thus we can rearrange the manipulator to solve for the mass of the box using the known inputs to the system and its observed outputs.

4.4.2 Dealing with Mass Estimator Lag

Due to the dynamics of the system, the mass estimator is not able to converge on a correct mass instantly, thus a problem arises where the robot needs to control the box while the mass estimator converges on an estimated mass.

There are several solutions to this. Initial designs had the manipulator attempt to simply hold the box stationary while the mass estimate converged. However, this was not a robust design because if the box was too heavy, the system dynamics would shift enough that the original controller would not be able to hold the box in place.

The final design used a relatively high gain controller to start the box manipulation while the mass estimate converged. Then once the mass estimate had sufficiently converged, the estimated mass was used to determine the non-linear mapping of the controller and a more relaxed low gain controller was able to be used for optimal control of the box.

4.4.3 Robust Control Of Estimated Box

An issue with the above system is that the mass estimator is sensitve to large input forces to the system. Thus if the reference position is quickly changed, the control action will cause the mass

estimator to deviate from the correct value. This causes an unstable positive feedback loop where the control action causes incorrect mass estimation which in turn causes incorrect control action.

The solution to this was to track the rate of change of the mass estimate and once this rate is sufficiently low, "lock in" the mass estimate and use this for the rest of the reference tracking. This way the system is robust to rapid reference changes or output disturbances.

The one trade off with this is that the mass used in the control system will never be the fully estimated mass. However, the control system is robust enough with sufficient gain that it can operate within this uncertainty.

4.4.4 Robust Control For Large Mass Boxes

The system described above is sufficient to meet the project specifications, however it is not robust enough to handle boxes with a large mass. A solution to this problem was to use a very high gain controller during the mass estimation phase. Using this method, the control system is able to manipulate boxes well beyond the system specifications. This process is illustrated in the block diagram below

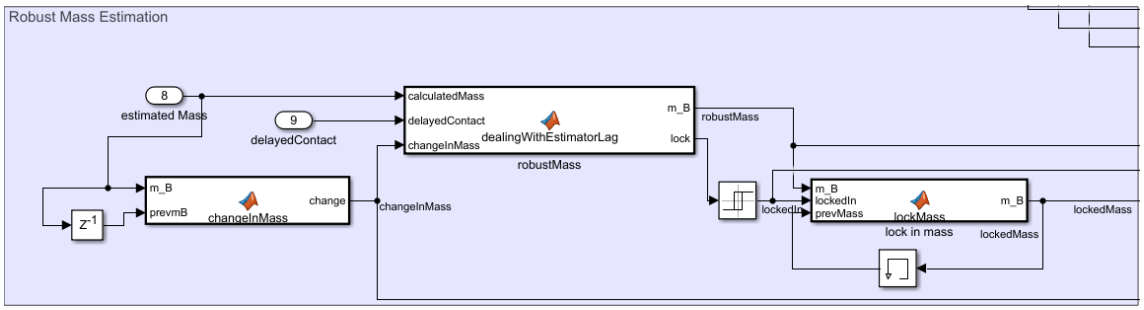

Figure 3. Mass "Lock in" System Block Diagram

The negatives of this approach include the large amount of control action that comes with high gain controllers. Additionally, the result of this aggressive control action is that the mass estimator never fully converges, there is always slight oscillation (in time with the control action) about the true mass. This is however successfully mitigated through the "lock in" strategy as described above.

5 Simulation Results

5.1 Estimating Drag Coefficient

Due to the nature of this being a simulated system, solving for the drag coefficient using system inputs and outputs as discussed in section 3.2 is an instantaneous process. This is shown in the plot below.

Figure 4. Drag Estimation Plot

5.2 Cost Function of Navigating To Box

The cost function for the robot navigating to, and manipulating a box of mass 1kg is shown below: It is clear from this plot that most of the optimisation is to be done around the navigation

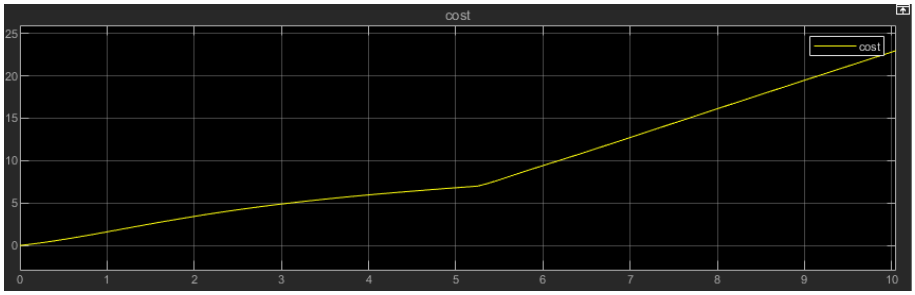

Figure 5. Cost Function Of Navigating to and manipulating box

to the box as required forces once the box has been collected are dictated by the mass of the box more than by optimised motion.

5.3 Convergence Of Mass Estimator

The behaviour of the mass estimator is illustrated in the following plot. This plot illustrates

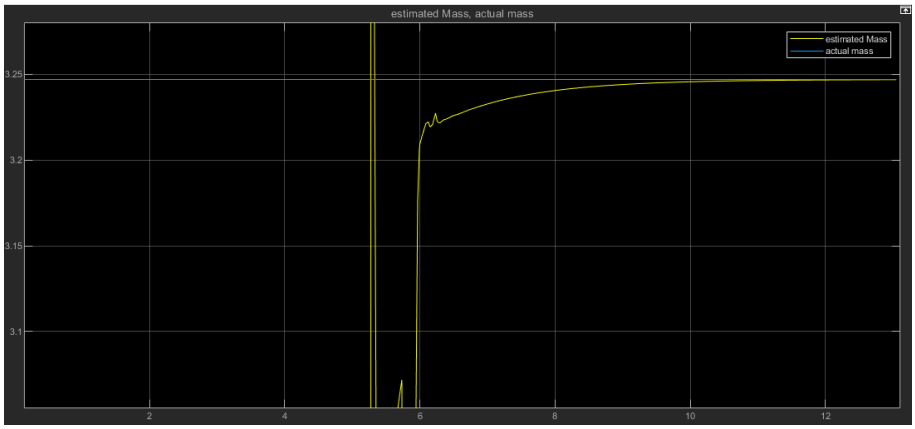

Figure 6. Mass Estimator Behaviour

the need for a delay before the estimated mass is used for control action. Using the estimated during the shown singularities causes the simulation to crash. The mass actually used in the controller feedback linearisation is shown below

5.4 Robust Control of Estimated Box

As described by section 4.4.3, the actual mass used in the controller dynamics starts at a nominal value, then "locks in" once the change in mass has decreased to an acceptable level. This behaviour is shown in the plot below which displays the mass actually used by the controller:

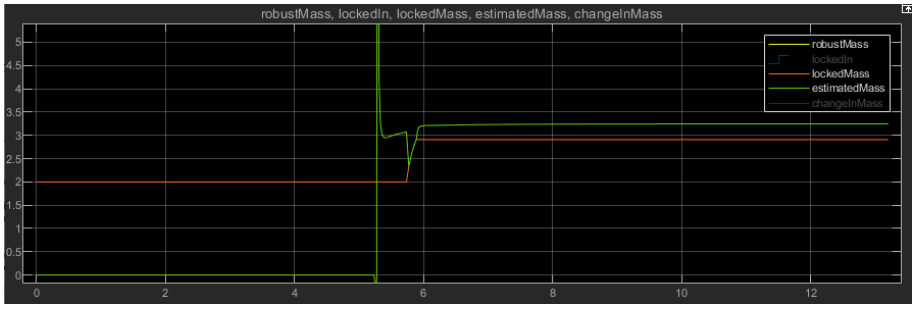

Figure 7. Displaying Mass

5.5 Robust Control Of Heavy Boxes

Section 4.4.4 describes how the use of a high gain controller enables the manipulation of large mass boxes beyond the system specifications. The plot below shows how the "lock in" system enables this by rejecting the oscillations that occur due to the high system gain.

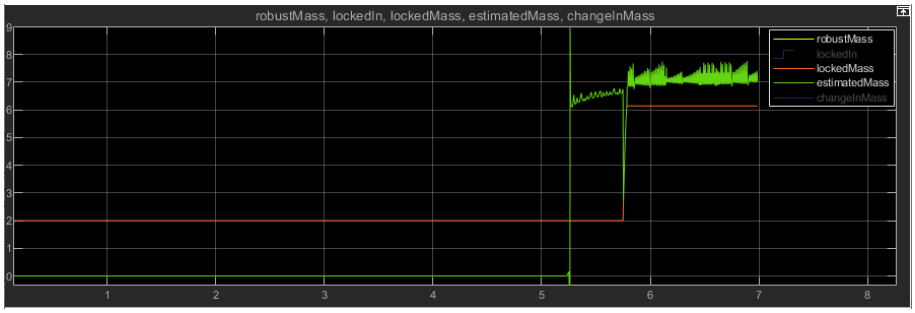

Figure 8. Plot Demonstrating efficacy of "lock in" strategy

6 Conclusion

This project was an effective learning experience in the control of non-linear systems. Overall it was a resounding success. The successful design of a controller for a non-linear system was achieved and the system specifications were exceeded as described in section 4.4.4. There are however optimisations that can be made as far as minimising required control action. In particular, there is definitely room for optimisation around choice of controller gain as discussed in section 4.2.

7 References

References

[1] Dr. Amir Patel, "Lagrange Mechanics II", University of Cape Town, 2022.

[2] Dr. Amir Patel, "Linearization (Static)", University of Cape Town, 2022.

[3] "Hessian matrix - Wikipedia", En.wikipedia.org, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Hessian_matrix. [Accessed: 15- May- 2022].

[4] M. Bjerkeng and K. Y. Pettersen, "A new Coriolis matrix factorization," 2012 IEEE International Conference on Robotics and Automation, 2012, pp. 4974-4979, doi: 10.1109/ICRA.2012.6224820.

8 Matlab Code

8.1 Modelling System

breaklines

```
1
2  clear;
3  %% defining system parameters
4
5  %commenting out the following because I am going to try to solve the whole
6  %thing symbolically to see if the results vary
7
8  % (the results were exactly the same)
9
10 %slider parameters
11 m_s=0.25;            %sliding base mass
12 %link parameters
13 m_l= 0.25;           %link mass
14 L_l= 0.25;           %link length
15 syms Jxx Jzz     %link inertias
16 Jyy=(1/3)*m_l*L_l^2;
17 I_L=diag([Jxx Jyy Jzz]);
18 syms b               % unknown drag coefficient on joints
19 %box params (to be picked up)
20 syms m_B ;           %box mass (variable)
21 L_B = 0.1;           %length of box side
22 %other
23 g=9.81;              %gravitational acceleration
24 %motor parameters
25 syms t1 t2           %torques on each joint
26 syms f1             %force applied by motors in slider
27
28
29
30
31 %% Generalised Coordinates
32 %{
33 The system is characterised by three generalised coordinates
34 x    : The position of the slider along the rail
35 th1 : The angle of the first link to the negative x-axis direction
36 th2 : The angle of the second link relative to the first link
37 %}
38 syms x th1 th2 dx dth1 dth2 ddx ddth1 ddth2
39 q= [x; th1; th2];
40 dq=[dx; dth1; dth2];
41 ddq= [ddx; ddth1; ddth2];
42
43 %% Rotation Matrices
44 %{
45 Note definition of angles above, th1 defined as rotation from negative x
46 horizontal. Note has been taking of the relevant rotation matrix needed
47 %}
48 R01=Roty(-th1);          %from 1 according to RHR (ccw)
49 R10=transpose(R01);     %
```

```
50 R12=Roty(-th2);           %from 1 to 2 according to RHR (ccw)
51 R02=R12*R01;              %from inertial to 2
52 R20=transpose(R02);       %
53
54
55 %% Mass positions
56 rs_0= [x;0;0];            % position of slider
57
58 rL1_1 = [L_l/2;0;0];       % link 1 in frame 1
59 rL1_0 = rs_0+R10*rL1_1; %         in inertial frame
60
61 rL2_2 = [L_l/2;0;0];       % link 2 in frame 2
62 rL2_0 = rs_0+R10*rL1_1*2+ R20*rL2_2;%in inertial frame
63
64 rB_2= [(L_B/2);0;0];     %box position in frame 2 (half a side length away from end of L2)
65 rB_0= rs_0+R10*rL1_1*2+ R20*rL2_2*2+R20*rB_2; %
66
67 %% Mass velocities
68 vs = jacobian(rs_0,q)*dq;        %slider
69 vL1 = jacobian(rL1_0,q)*dq;     %link 1
70 vL2 = jacobian(rL2_0,q)*dq;     %link 2
71 vB  = jacobian(rB_0,q)*dq;      %box
72 %% Angular Velocities
73 %scalars
74 wL1_0=[0;dth1;0];         %
75 wL2_0=[0;dth1+dth2;0]; %conviniently
76 %angular velocities in body coordinates
77 w01_2=R12*[0;dth1;0];
78 w12_2=[0;dth2;0];
79 w02_2=w01_2+w12_2;        %should simply be dth1+dth2
80 %% Kinetic Energy
81 %using vector angular velocities
82 I_B=diag([0;(1/6)*m_B*L_B^2;0]);
83 Ts=  0.5*m_s*transpose(vs)*vs;
84 TL1= 0.5*m_l*transpose(vL1)*vL1+0.5*transpose(w01_2)*I_L*w01_2; %link1
85 TL2= 0.5*m_l*transpose(vL2)*vL2+0.5*transpose(w02_2)*I_L*w02_2; %link2
86 TB=       0.5*m_B*transpose(vB)*vB+0.5*transpose(w02_2)*I_B*w02_2; %box has same w as link 2
87 Ttot=simplify(Ts+TL1+TL2+TB);
88 %{
89 % using scalar angular velocities
90 % I tested it and yields same results
91 Ts=  0.5*m_s*transpose(vs)*vs;
92 TL1= 0.5*m_l*transpose(vL1)*vL1+0.5*transpose(wL1_0)*Jyy*wL1_0; %link1
93 TL2= 0.5*m_l*transpose(vL2)*vL2+0.5*transpose(wL2_0)*Jyy*wL2_0; %link2
94 TB=0;% box ignored for this milestone
95 Ttot=simplify(Ts+TL1+TL2+TB);
96 %}
97 %% Potential Energy (height represented in z axis)
98 % slider assumed to have no potential energy
99 VL1=m_l*g*transpose(rL1_0)*[0;0;1]; %Link 1
100 VL2=m_l*g*transpose(rL2_0)*[0;0;1]; %Link 2
101 VB= m_B*g*transpose(rB_0)*[0;0;1]; %box
102 Vtot=simplify(VL1+VL2+VB);
103
104 %% Define Mass Matrix
```

```
105 M = hessian(Ttot,dq);
106
107 %% Define Mass Matrix Deriv
108 dM = sym(zeros(length(M),length(M)));
109 for i=1:length(M)
110     for j=1:length(M)
111         dM(i,j) = jacobian(M(i,j),q)*dq;
112     end
113 end
114 dM = simplify(dM);
115
116 %% Define Gravity Matrix
117 G = jacobian(Vtot,q);
118 G = simplify(G);
119
120 %% Define Coriolis Matrix
121 C = dM*dq - transpose(jacobian(Ttot,q));
122 C = simplify(C);
123
124 %% Using calculated drag coeffients
125 b=0.9412;
126
127 %% inputs vector
128 T=[f1,t1,t2]';
129
130
131
132 %% Getting EOMs
133
134 manip= M*(ddq) + C + transpose(G) -T + b.*dq==0;
135
136 col=collect(manip,m_B)
137 mVect=[m_B,m_B,m_B];
138 %assert(isequal(manip,mVect*col))
139
140 m1=solve(manip(1),m_B);
141 m2=solve(manip(2),m_B);
142 m3=solve(manip(3),m_B);
143
144 Stdnum = 'WHTDYL001'; % Student number HERE
145
146 %simulink agrees
147
148 %% Helper Functions
149 function A = Roty(th)
150     A = [cos(th)   0   -sin(th);...
151          0         1    0;...
152          sin(th)   0    cos(th)];
153 end
```

8.2 Designing Controller without Box

breaklines

```
%% admin
clear;clc;

%% feedback linearisation
%{
y=[x;y];
where x and y are the positions of the end effector
%}
load("dynamics_no_box.mat")

%rend_0 is of form x,y,z for the end effector
syms x th1 th2
Y=[rend_0(1); rend_0(3); th1+th2];
%Y=rend_0; %pretty sure

J=simplify(jacobian(Y,q));

%shamelessly yoinked from Prof's code
dJ(:,1) = jacobian(J(:,1),q)*dq;
dJ(:,2) = jacobian(J(:,2),q)*dq;
dJ(:,3) = jacobian(J(:,3),q)*dq;
dJ
J
%% Going to SS representation
%X=[x1,Z1,th1+th2,dx,dz,dth1+dth2]
%U=[f1,t1,t2]
% Y=[x_end,y_end,th1+th2]

A=zeros(6);
A(1,4)=1; % for dx
A(2,5)=1; % for dth1
A(3,6)=1; % for dth2

B=[0,0,0;
    0,0,0;
    0,0,0;
    1,0,0;
    0,1,0;
    0,0,1];

C=[1,1,1,0,0,0];
D=0;

%% getting controller gains
Q = 1.8*eye(6);
R = 2.5;
Klqr = lqr(A,B,Q,R)

%% step testing
%% step testing loop
Acl=A-B*Klqr;
cl=ss(Acl,B,C,D);
%figure()
%step(cl,'K')

```

```
Kdc = dcgain(cl);
Kr =1./Kdc

%% Step scale
% Closed loop system
syscl_scaled = ss(Acl,B.*Kr,C,D);
figure()
step(syscl_scaled, 'r');
```

8.3 Designing Controller with Box

breaklines

```
%% admin
clear;clc;

%% feedback linearisation
%{
y=[x;y];
where x and y are the positions of the end effector
%}
load("dynamics_with_box.mat")

%rend_0 is of form x,y,z for the end effector
syms x th1 th2
% note controlling position of center of mass of box
Y=[rB_0(1); rB_0(3); th1+th2];
%Y=rend_0; %pretty sure

J=simplify(jacobian(Y,q));

%shamelessly yoinked from Prof's code
dJ(:,1) = jacobian(J(:,1),q)*dq;
dJ(:,2) = jacobian(J(:,2),q)*dq;
dJ(:,3) = jacobian(J(:,3),q)*dq;
dJ
J
%% Going to SS representation
%X=[x1,Z1,th1+th2,dx,dz,dth1+dth2]
%U=[f1,t1,t2]
% Y=[x_end,y_end,th1+th2]

A=zeros(6);
A(1,4)=1; % for dx
A(2,5)=1; % for dth1
A(3,6)=1; % for dth2

B=[0,0,0;
    0,0,0;
    0,0,0;
    1,0,0;
    0,1,0;
    0,0,1];

```

```
42 C=[1,1,1,0,0,0];
43 D=0;
44
45 %% getting controller gains
46 Q = 5*eye(6);
47 R = 2.5;
48 Klqr = lqr(A,B,Q,R)
49
50 %% step testing loop
51 Acl=A-B*Klqr;
52 cl=ss(Acl,B,C,D);
53 %figure()
54 %step(cl,'K')
55
56 Kdc = dcgain(cl);
57 Kr =1./Kdc
58
59 %% Step scale
60 % Closed loop system
61 syscl_scaled = ss(Acl,B.*Kr,C,D);
62 figure()
63 step(syscl_scaled, 'r');
```

9 Simulink Model

The following images of the simulink model used illustrate

1. The Overall Model Structure
2. The Gain Scheduled Controller
3. The Feedback Linearised Full State Feedback Controller

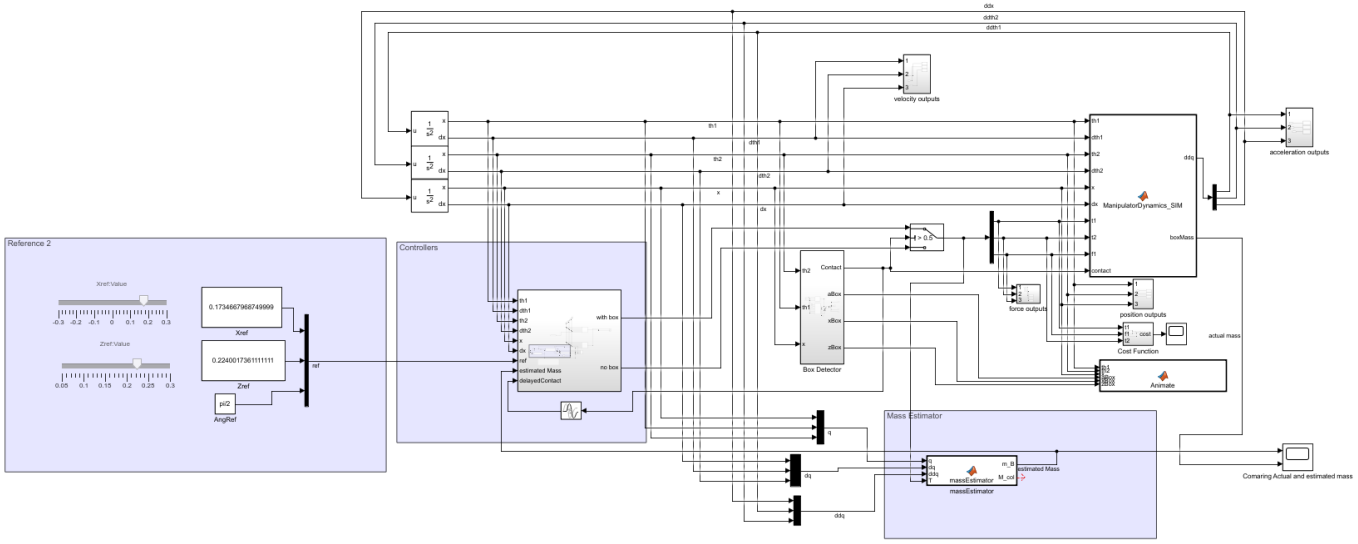

Figure 9. Model Overview

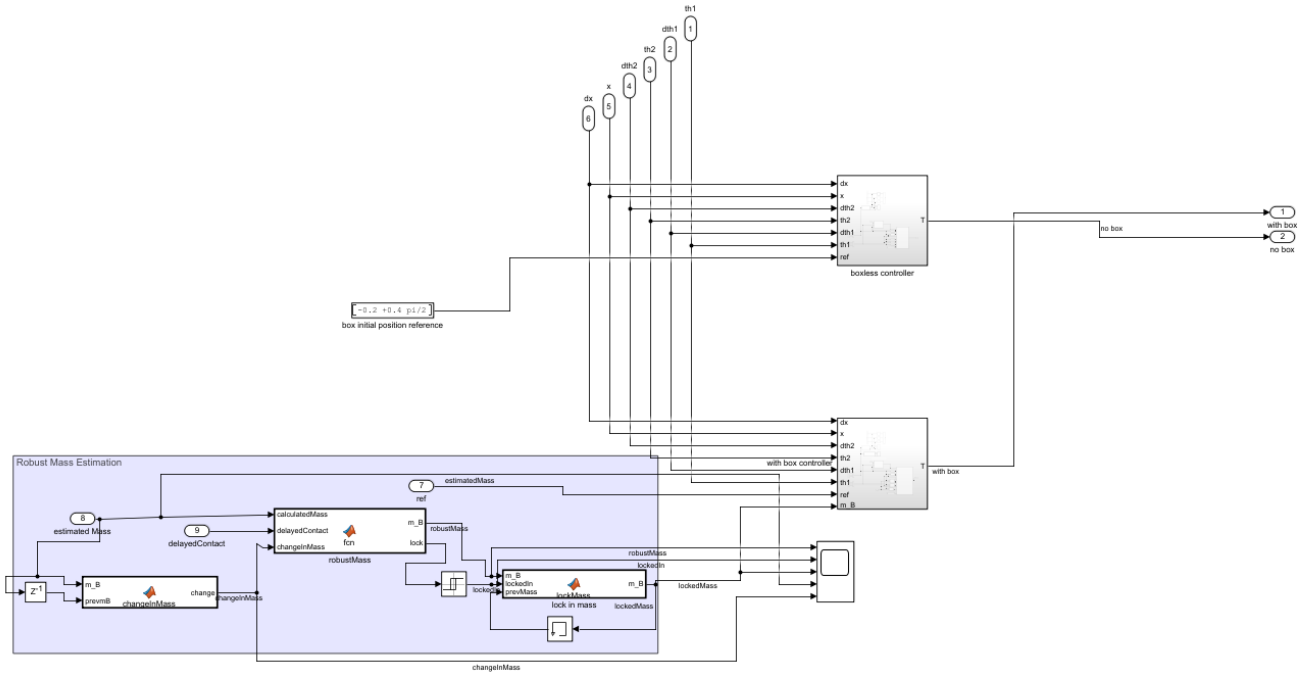

Figure 10. Gain Scheduled Controller

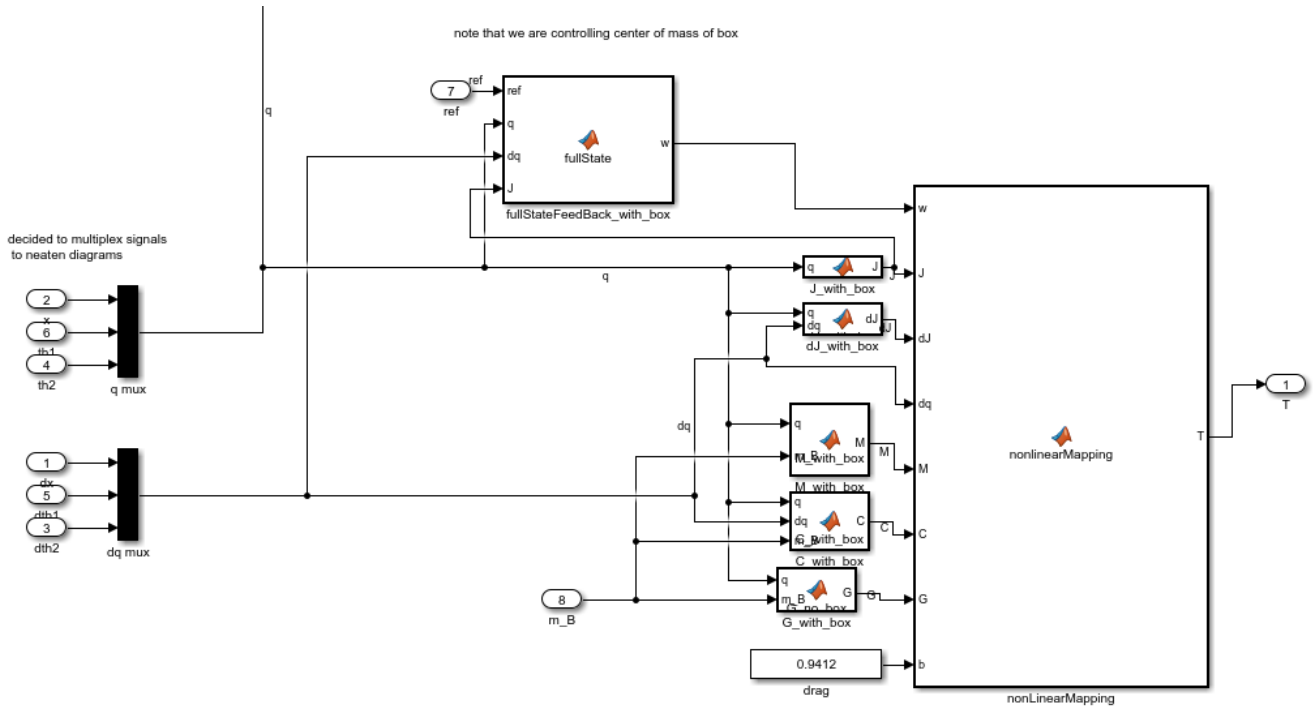

Figure 11. Feedback Linearised Full State Feedback Controller