WTZMIC001 Mechatronics II report

Introduction

Due to the huge amount of rockets and satellites that inhabit the space close to our earth, there are big danger of colliding with space debris. The aim of this project was to model and design a controller for a 3-armlink debris collector in space. Each link has a motor attached to it that is able to produce a torque. By varying the torque one is able to control the arms. The system had an unknown coefficient of friction for each arm which had to be identified. The system will operate in 2 modes and the aim was for the end of the arm (the debris collector) to be able to catch space debris at all times. The system was non-linear and a controller had to be designed to produce the right control action.

Requirements

The arm is required to :

1. Swing from a vertically down position to a vertically up position
2. Intercept debris by performing the following:
    a. Have the end-effector <0.1m to the debris
    b. End with velocity vector at angle of 30degrees relative to axis of 3rd link

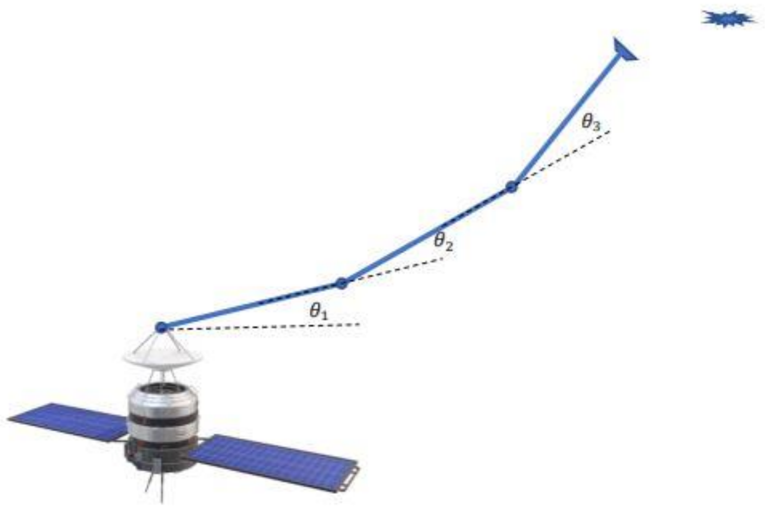


Figure 1.1: robot arm

There are 2 modes in which the arm must be able to operate:

- Mode 1: Speed of debris ranges between -1ms and 1ms
- Mode 2: Speed of debris ranges between -5ms and 5ms
-

Dynamics of System

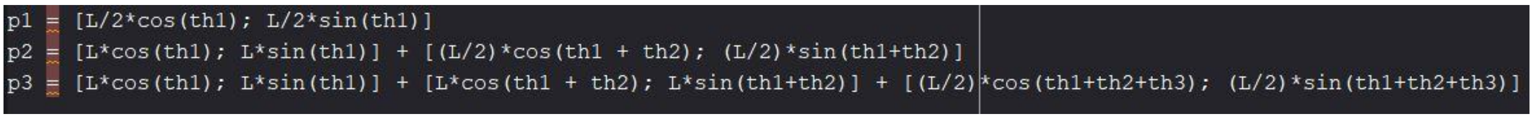
The system can be modelled by the following equation

$$M * ddq + C = B * tau + Q$$

Where M = mass matrix; C = Centrifugal/velocity vector; B = eigen matrix; Q = Friction force

Position Vectors

Each position is calculated for each link around the COM. Assume that the COM is in the middle of the link



```
p1 = [L/2*cos(th1); L/2*sin(th1)]
p2 = [L*cos(th1); L*sin(th1)] + [(L/2)*cos(th1 + th2); (L/2)*sin(th1+th2)]
p3 = [L*cos(th1); L*sin(th1)] + [L*cos(th1 + th2); L*sin(th1+th2)] + [(L/2)*cos(th1+th2+th3); (L/2)*sin(th1+th2+th3)]
```

Velocity Vectors:

$$v1 = jacobian(p1, q) * dth1$$

$$v2 = jacobian(p3, q) * dth2$$

$$v3 = jacobian(p2, q) * dth3$$

Energy

Kinetic:

$$T = \frac{1}{2} q^T M(q) dq$$

See Energy code at end of document

Potential:

The system was modelled in place and therefore had no potential energy

Mass matrix:

$$T = \frac{1}{2} q^T M(q) dq$$

Rewrite

$$M(q)_{i,j} = \frac{\partial T}{\partial dqi * \partial qj}$$

M =

$$\begin{pmatrix} 3\,\mathrm{I} + \frac{15\,L^2\,m}{4} + L^2\,m\cos(\mathrm{th}_2+\mathrm{th}_3) + 3\,L^2\,m\cos(\mathrm{th}_2) + \sigma_4 & \sigma_1 & \sigma_2 \\ \sigma_1 & 2\,\mathrm{I} + \frac{3\,L^2\,m}{2} + \sigma_4 & \sigma_3 \\ \sigma_2 & \sigma_3 & \frac{m\,L^2}{4} + \mathrm{I} \end{pmatrix}$$

where

$$\sigma_1 = 2\,\mathrm{I} + \frac{3\,L^2\,m}{2} + \frac{L^2\,m\cos(\mathrm{th}_2+\mathrm{th}_3)}{2} + \frac{3\,L^2\,m\cos(\mathrm{th}_2)}{2} + \sigma_4$$

$$\sigma_2 = \mathrm{I} + \frac{L^2\,m}{4} + \frac{L^2\,m\cos(\mathrm{th}_2+\mathrm{th}_3)}{2} + \frac{\sigma_4}{2}$$

$$\sigma_3 = \mathrm{I} + \frac{L^2\,m}{4} + \frac{\sigma_4}{2}$$

Centrifugal matrix

$$C = dM * dq - \frac{\partial T}{\partial q}$$

See code at end of document

Friction Force

$$Q = \frac{b1}{-dth1} = \frac{b2}{-dth2} = \frac{b2}{-dth2}$$

To find the friction force we were tasked with finding the unknown coiefficient b

The following method was used:

$$Q = M * ddq + C - B * tau$$

All the parameters of Q could be calculated. This was done in matlab using 1st symbols. And then with simulation results to get q and dq

Tau was set manually to 0.1

After taking all the values of Q for and averaging it over all time steps a value of B was found

b = 4.3179

Controlling the system

The design process which was followed was to identify which parameter would be the system output – in the sence, which parameter would be calculated in the error.

Options for error calculations:

- Error could be calculated based upon the distance between the debris and this end of arm3. However this approach produced a very big error in the start of the simulation as the debris is far away from the arm. After many changing of gains I could not find an optimal situation where the link consistently catches the debris. In this case the link would go around and keep going around and not settling on one spot
- Error could be calculated based upon angle between the 1st link and the vector angle of the debris. This turned out to be a robust method for 2 reasons
    - The end of the arm could efficiently catch debris
    - It was easy to code in the manual position mode

The system is however non linear and before linear control can be applied the system had to be linearised. The method that was used was feedback linearization

Feedback linearisation

To linearize the system a dummy variable v was used.

$$ddq = M^{-1}(-C + B * tau + Q)$$

Rearrange:

$$tau = B^{-1}(M * ddq + C - Q)$$

System parameter to control:

$$y = \begin{bmatrix} th1 \\ th2 \\ th3 \end{bmatrix}$$

$$dy = J * dq$$

$$ddy = dJ * dq + J * ddq$$

Where J is the Jacobian and dJ is the derivative of the Jacobian

To linearize we set: $ddy = v$

We do this so that we can control V and then we map it back to tau and ultimately control tau – the control action

Rearrange: $ddq = \frac{v - dJ*dq}{J}$

We can then sub back into tau equation :

$$tau = B^{-1}\left(M * \left(\frac{v - dJ*dq}{J}\right) + C - Q\right)$$

We then set $v \;=\; Kp * e \;+\; Kd * de$

Kp and Kd are our controller gains which I will speak about in the next section

Ultimately we get a tau equation to control the control action:

$$tau = B^{-1}\left(M * \left(\frac{Kp * e + Kd * e - dJ * dq}{J}\right) + C - Q\right)$$

Controller

An attempt was made to make a state space controller however it did not work so well.

Here are the State Space equations:

$$\frac{d}{dt}\begin{pmatrix} dx \\ dy \\ dz \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} dx \\ dy \\ dz \\ x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} v1 \\ v2 \\ v3 \end{pmatrix}$$

$$y = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} dx \\ dy \\ dz \\ x \\ y \\ z \end{pmatrix}$$

Using the Ackerman Formula to compute the gains

$$|sI - A + bK^T| = s^3(s+k1)(s+k2)(s+k3)$$

This required allot of pole placement and I therefore changed tactics

PID controller

This controller was implemented using code. There were 2 gains used. A proportional gain and a derivative gain

The formula was:

$$v = Kp * e + Kd * de$$

$$e = [e1; e2; e3] and\ de\ derivative\ of\ error = [de1; de2; de3]$$

The error was calculated as follows

a setpoint(angle was calculated):

$$setpoint = \tan^{-1}(\frac{debris_y}{debris_x})$$

Using this formula the angle of attack was calculated and the aim was for the robot arm to math this.

To calculate the error terms there were 2 scenarios:

- The debris approached from the left

$$e = \begin{bmatrix} setpoint - th1 \\ -th2 \\ -th3 \end{bmatrix}$$

- The debris approached from the right

$$e = \begin{bmatrix} -pi + setpoint - th1 \\ -th2 \\ -th3 \end{bmatrix}$$

the de term was calculated as follows:

$$de = \begin{bmatrix} -dth1 \\ -dth2 \\ -dth3 \end{bmatrix}$$

$$These\ terms\ were\ collected\ from\ model$$

Gains were chosen with a trial and error method and settled on

Kp = 65

Kd = 13

This was subbed into the control action equation and the tau's were calculated:

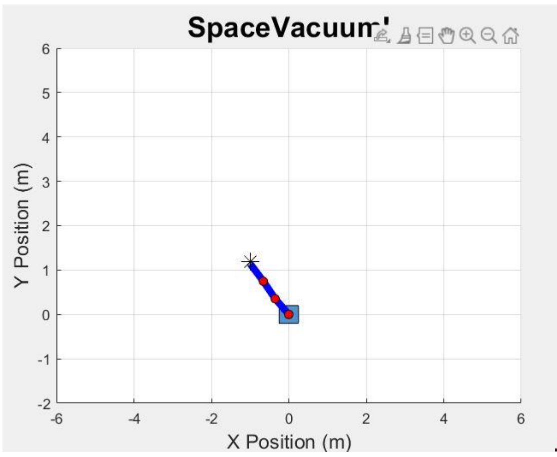
$$tau = B^{-1}\left(M * \left(\frac{Kp * e + Kd * e - dJ * dq}{J}\right) + C - Q\right)$$

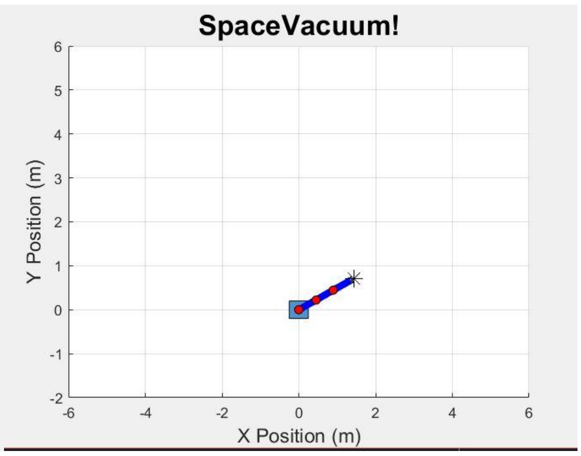
The controller received th and dth values from the model

See controller code at end of document

Experimental results

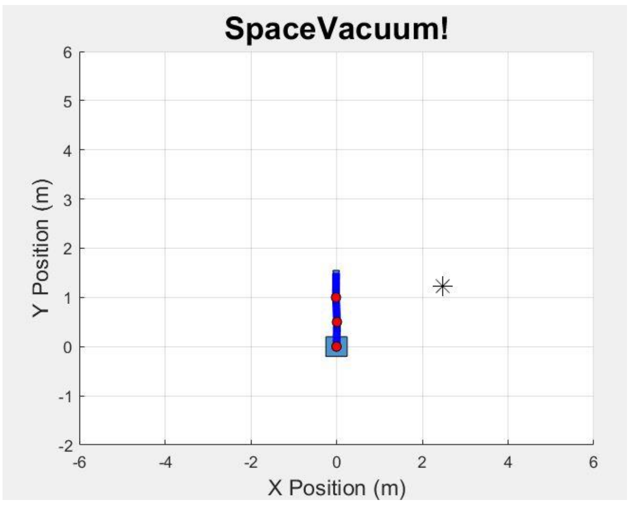
The system performs very well. The robot arm is able to track the position of debris with any seed value I proposed. It tracks debree coming from all angles and all speeds.





Manual Mode:

If the manual_mode constant is set to 1 in the control block the controller gives the right control action and successfully rotates the arm from vertical down to a vertical up position



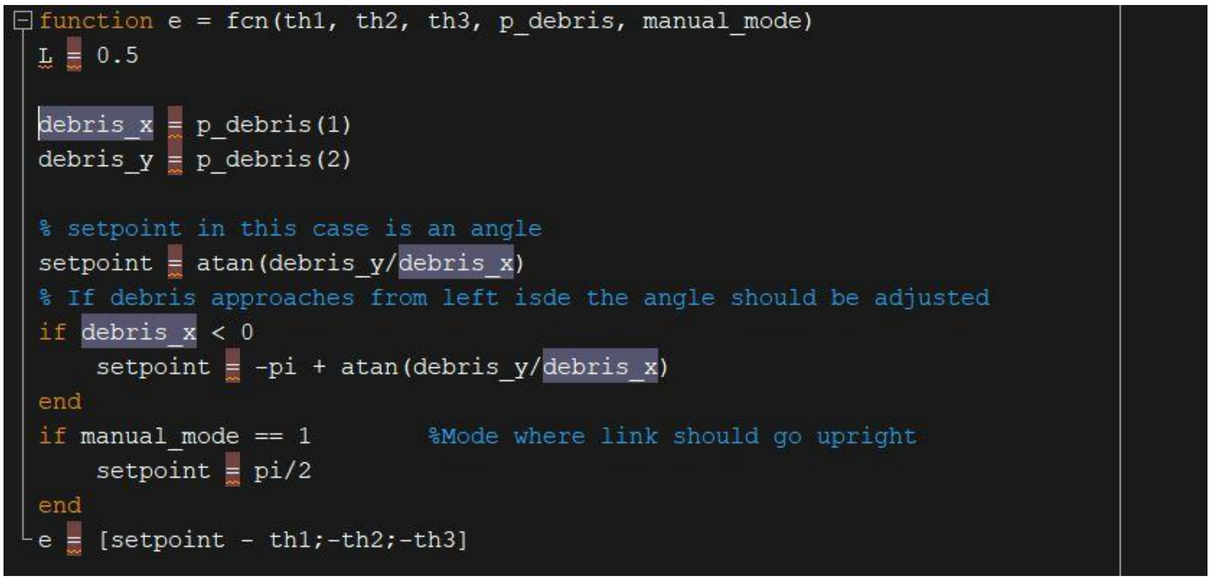
Conclusion:

The controller works wonderful. I would've liked to put the gains a little bit lower. But it was hard to find the right seed to bring about the worst conditions. If given more time I would have recreated the system and manually given the worst conditions and then chosen appropriate gains. This is because of the cost function that one wants to minimise. Overall The project was a great success and I was able to model the system, find the unknown b-coefficient and create a controller that accurately caught debris. I am proud.

Improvements was gain tuning. Next time I would model the system properly in Simulink that I am able to use the automatic gain tuner software.

Code Sections

Error calc:

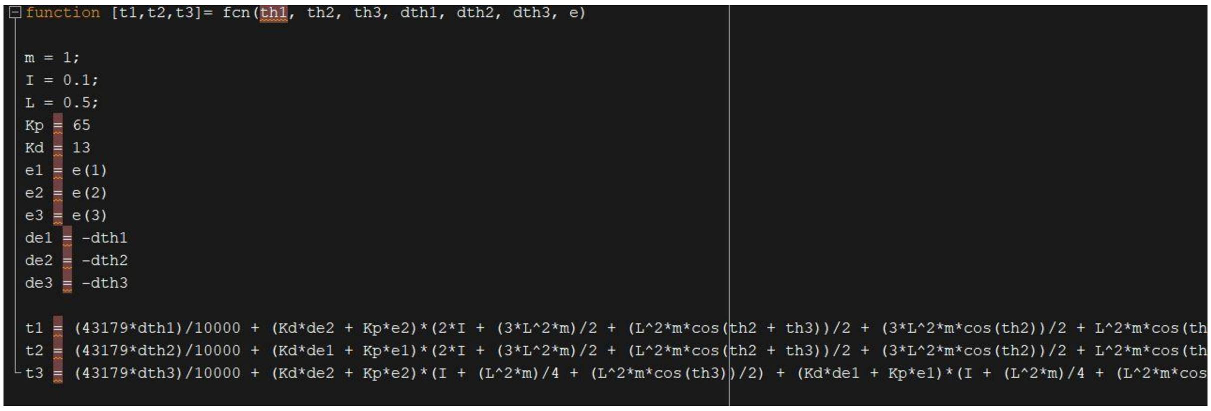


```
function e = fcn(th1, th2, th3, p_debris, manual_mode)
L = 0.5

debris_x = p_debris(1)
debris_y = p_debris(2)

% setpoint in this case is an angle
setpoint = atan(debris_y/debris_x)
% If debris approaches from left isde the angle should be adjusted
if debris_x < 0
    setpoint = -pi + atan(debris_y/debris_x)
end
if manual_mode == 1         %Mode where link should go upright
    setpoint = pi/2
end
e = [setpoint - th1;-th2;-th3]
```

Controller:

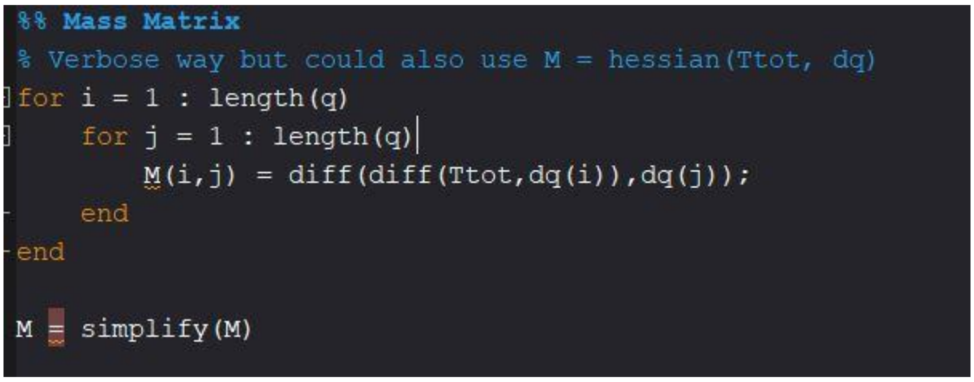


```
function [t1,t2,t3]= fcn(th1, th2, th3, dth1, dth2, dth3, e)

m = 1;
I = 0.1;
L = 0.5;
Kp = 65
Kd = 13
e1 = e(1)
e2 = e(2)
e3 = e(3)
de1 = -dth1
de2 = -dth2
de3 = -dth3

t1 = (43179*dth1)/10000 + (Kd*de2 + Kp*e2)*(2*I + (3*L^2*m)/2 + (L^2*m*cos(th2 + th3))/2 + (3*L^2*m*cos(th2))/2 + L^2*m*cos(th
t2 = (43179*dth2)/10000 + (Kd*de1 + Kp*e1)*(2*I + (3*L^2*m)/2 + (L^2*m*cos(th2 + th3))/2 + (3*L^2*m*cos(th2))/2 + L^2*m*cos(th
t3 = (43179*dth3)/10000 + (Kd*de2 + Kp*e2)*(I + (L^2*m)/4 + (L^2*m*cos(th3))/2) + (Kd*de1 + Kp*e1)*(I + (L^2*m)/4 + (L^2*m*cos
```

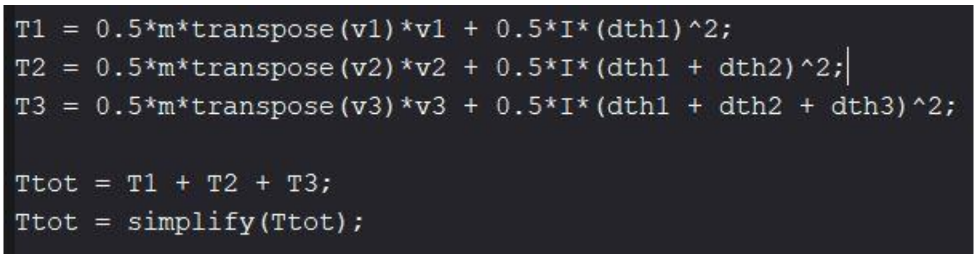
Mass matrix Code:



```
%% Mass Matrix
% Verbose way but could also use M = hessian(Ttot, dq)
for i = 1 : length(q)
    for j = 1 : length(q)
        M(i,j) = diff(diff(Ttot,dq(i)),dq(j));
    end
end

M = simplify(M)
```

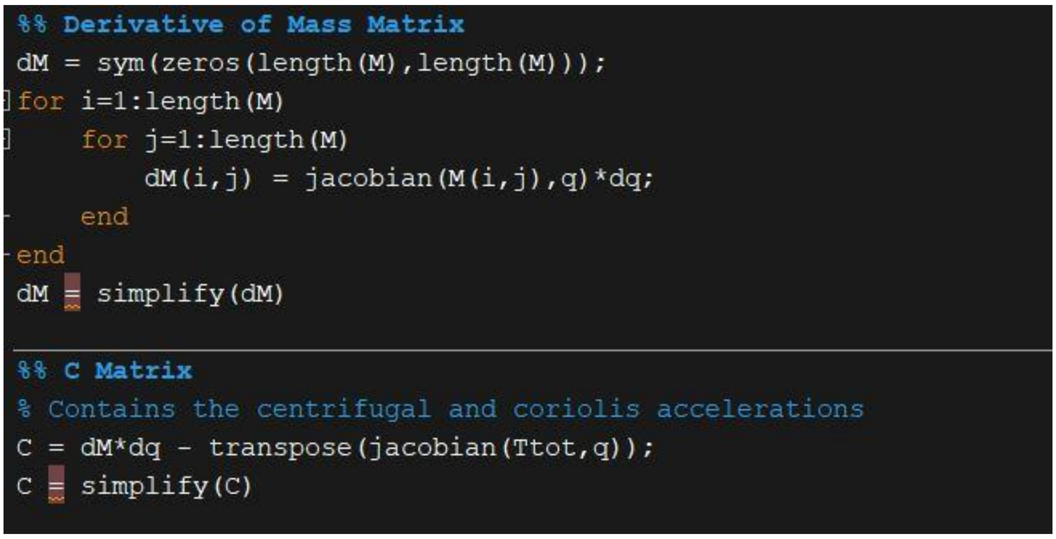
Code to find Energy:



```
T1 = 0.5*m*transpose(v1)*v1 + 0.5*I*(dth1)^2;
T2 = 0.5*m*transpose(v2)*v2 + 0.5*I*(dth1 + dth2)^2;
T3 = 0.5*m*transpose(v3)*v3 + 0.5*I*(dth1 + dth2 + dth3)^2;

Ttot = T1 + T2 + T3;
Ttot = simplify(Ttot);
```

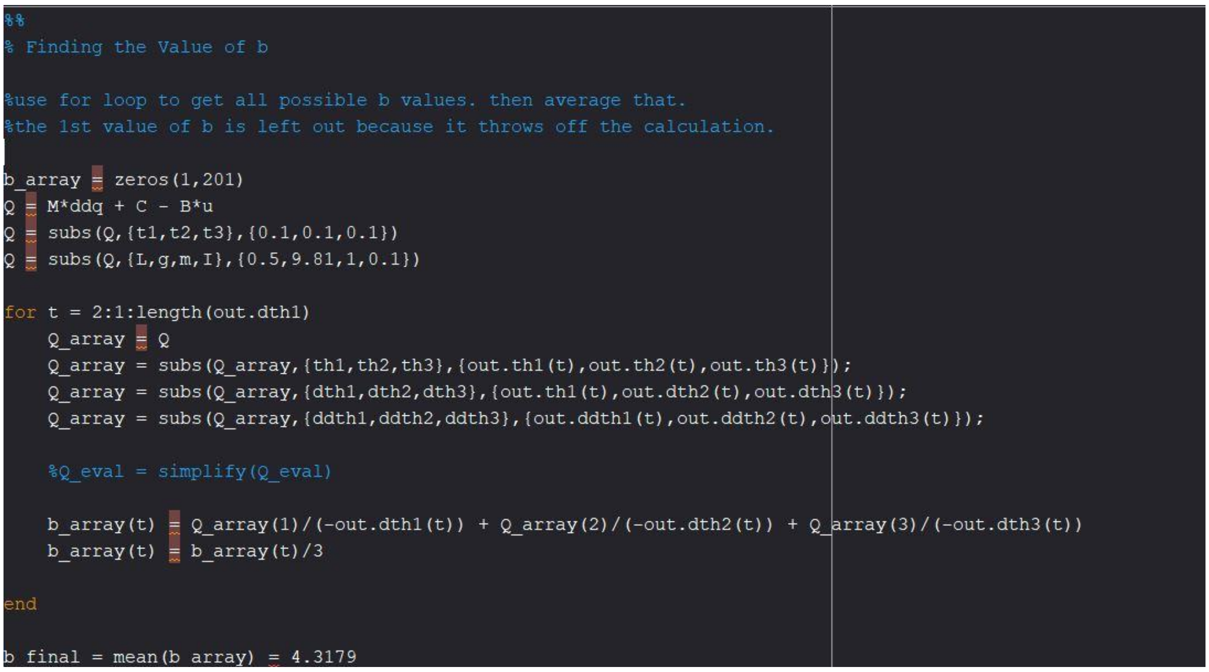
Centrifugal Matrix:



```
%% Derivative of Mass Matrix
dM = sym(zeros(length(M),length(M)));
for i=1:length(M)
    for j=1:length(M)
        dM(i,j) = jacobian(M(i,j),q)*dq;
    end
end
dM = simplify(dM)

%% C Matrix
% Contains the centrifugal and coriolis accelerations
C = dM*dq - transpose(jacobian(Ttot,q));
C = simplify(C)
```

Code to find B:



```
%%
% Finding the Value of b

%use for loop to get all possible b values. then average that.
%the 1st value of b is left out because it throws off the calculation.

b_array = zeros(1,201)
Q = M*ddq + C - B*u
Q = subs(Q,{t1,t2,t3},{0.1,0.1,0.1})
Q = subs(Q,{L,g,m,I},{0.5,9.81,1,0.1})

for t = 2:1:length(out.dth1)
    Q_array = Q
    Q_array = subs(Q_array,{th1,th2,th3},{out.th1(t),out.th2(t),out.th3(t)});
    Q_array = subs(Q_array,{dth1,dth2,dth3},{out.th1(t),out.dth2(t),out.dth3(t)});
    Q_array = subs(Q_array,{ddth1,ddth2,ddth3},{out.ddth1(t),out.ddth2(t),out.ddth3(t)});

    %Q_eval = simplify(Q_eval)

    b_array(t) = Q_array(1)/(-out.dth1(t)) + Q_array(2)/(-out.dth2(t)) + Q_array(3)/(-out.dth3(t))
    b_array(t) = b_array(t)/3

end

b_final = mean(b_array) = 4.3179
```