School of Electrical and Electronic Engineering

EEEN 30150

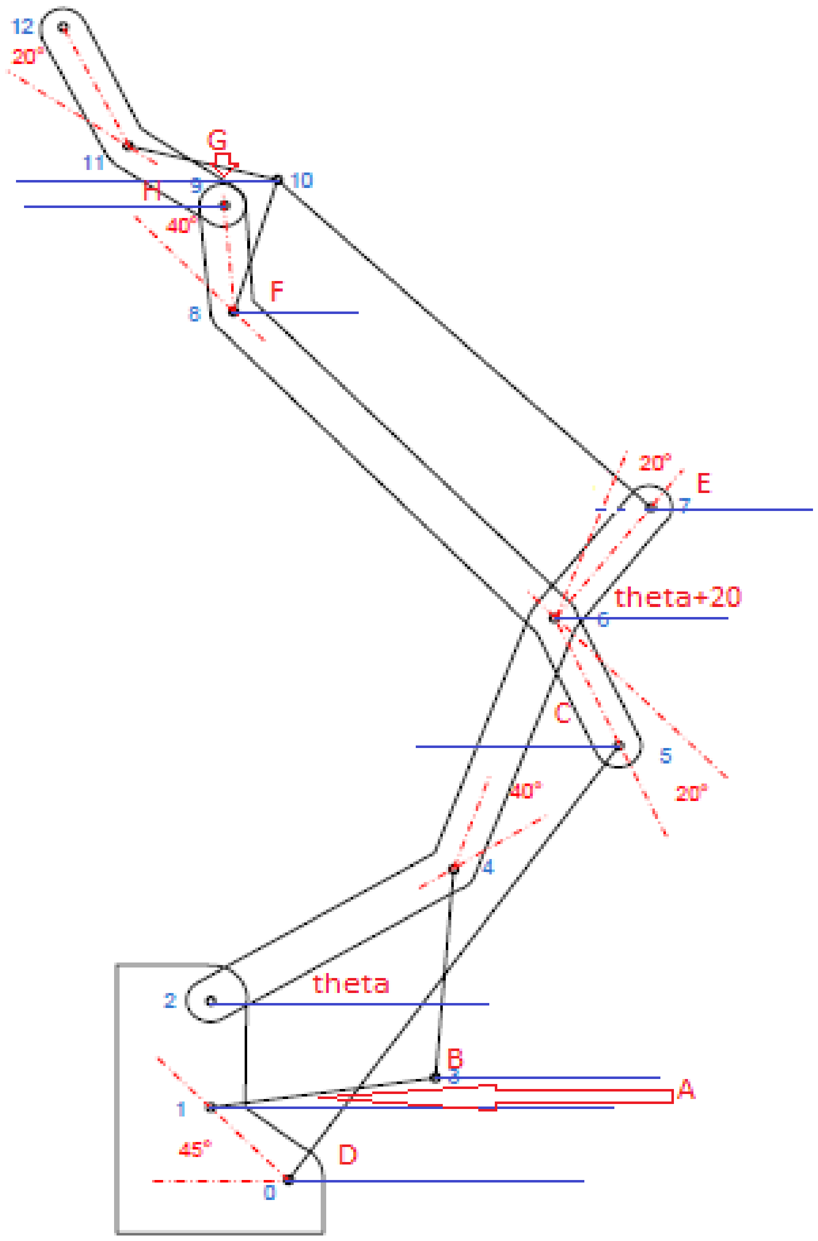
Modelling and Simulation

Minor Project: Static Equations

Student Name:	Paul Doherty
Student Number:	10387129
Date:	25/03/2013
Minor Project:	Primary Problem 2

Introduction:

The aim of this project is to solve a system of unknown angles and points for a range of the known variable θ using numerical methods . Fig. 1 shows the mechanical linkage for storage and deployment of the roof of a 1991 Ford Mustang and also shows the known angles, the unknown angles (A,B,C,D,E,F,G,H), the variable angle θ (theta) and unknown points 0 - 12.



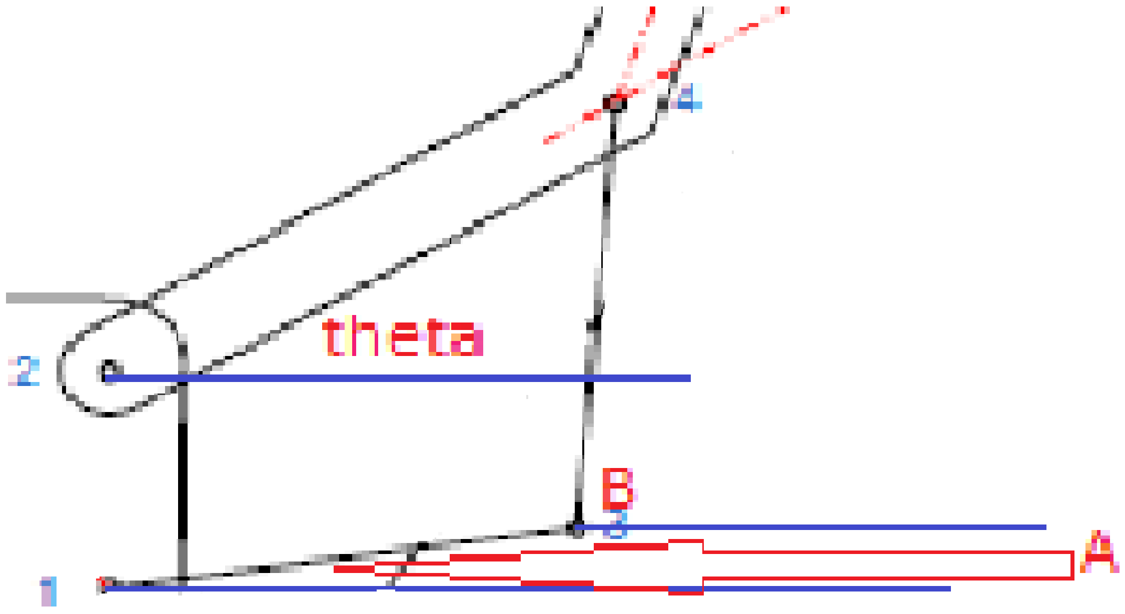
o

Fig 1

In this project the following link lengths were provided:
$l_{0\to1} = 1$, $l_{1\to2} = 1$, $l_{1\to3} = 2$, $l_{3\to4} = 2$, $l_{0\to5} = 5$, $l_{2\to4} = 2.5$, $l_{4\to6} = 2.5$, $l_{6\to7} = 4/3$, $l_{5\to6} = 4/3$, $l_{6\to8} = 4$, $l_{8\to9} = 1$, $l_{7\to10} = 4.5$, $l_{10\to8} = 4/3$, $l_{10\to11} = 4/3$, $l_{9\to11} = 1$, $l_{11\to12} = 4/3$

Finding expressions for unknown angles and solving numerically:

In order to find the unknown angles by numerical methods first basic kinematics was applied to find all the angles of all the links and subsequently all the all the positions of all the joints. The angles A and B were found by analysing the quadrilateral between points 1, 2, 4, and 3.



By employing kinematics to the quadrilateral and finding the i and j coordinates, angles A and B can be expressed as the matrix:

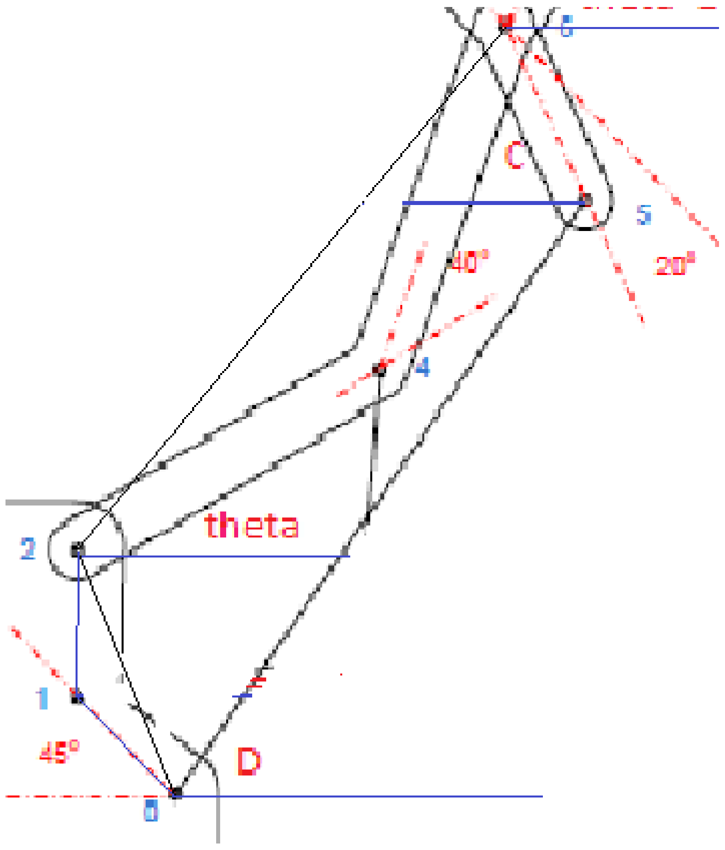
$$f1 = \begin{bmatrix} f1i \\ f1j \end{bmatrix} = \begin{bmatrix} 2\cos(A) + 2\cos(B - 2.5\cos(\theta)) \\ 2\sin(A) + 2\sin(B) - 2.5\sin(\theta) - 1 \end{bmatrix}$$

From this matrix, solutions can now be found for both A and B by numerical methods i.e. the Newton-Raphson (NR) algorithm. This was done by using MATLAB. Below is the code that was employed to achieve this:

```
%Part 1 - Solution to angles A and B

v1 = [pi/18; pi/2];      %initial guess for solutions of newton-raphson (NR)
%function for part 1
f1_i = 2*cos(v1(1)) + 2*cos(v1(2)) - (2.5)*cos(theta); %i component
f1_j = 2*sin(v1(1)) + 2*sin(v1(2)) - (2.5)*sin(theta) - 1; %j component
sum1 = f1_i + f1_j;          %initialising test for convergence of NR
test1 = abs(sum1);

%initiate while loop to iterate NR
while test1 > 0.001
```

```
%update function with new guess
f1_i = 2*cos(v1(1)) + 2*cos(v1(2)) - (2.5)*cos(theta);
f1_j = 2*sin(v1(1)) + 2*sin(v1(2)) - (2.5)*sin(theta) - 1;
sum1 = f1_i + f1_j; %update test
test1 = abs(sum1);
%find components of Jacobian matrix for NR
Df1_i = [(-2*sin(v1(1))) (-2*sin(v1(2)))]; %i component
Df1_j = [(2*cos(v1(1))) (2*cos(v1(2)))];   %j component

f1 = [f1_i;f1_j];  %create matrix to hold function
Df1 = [Df1_i;Df1_j]; %create matrix to hold jacobian

%the following three lines of code are to find the inverse matrix of the
%jacobian
matrix_pivoted1 = [Df1(4) -Df1(3);-Df1(2) Df1(1)]; %swas first and fourth
%elements of jacobian and changes sign of second end third elements
det1 = (1/(Df1(1)*Df1(4)-Df1(2)*Df1(3))); %finds determinant of jacobian
inverse1 = matrix_pivoted1*det1; %finds inverse of jacobian
%NR algorithm:
v1 = v1 - (inverse1*f1);
end     %end while loop
```

Angle C and D were found similarly. Kinematics was used to analyse the quadrilateral between points 0, 2, 6, and 5.



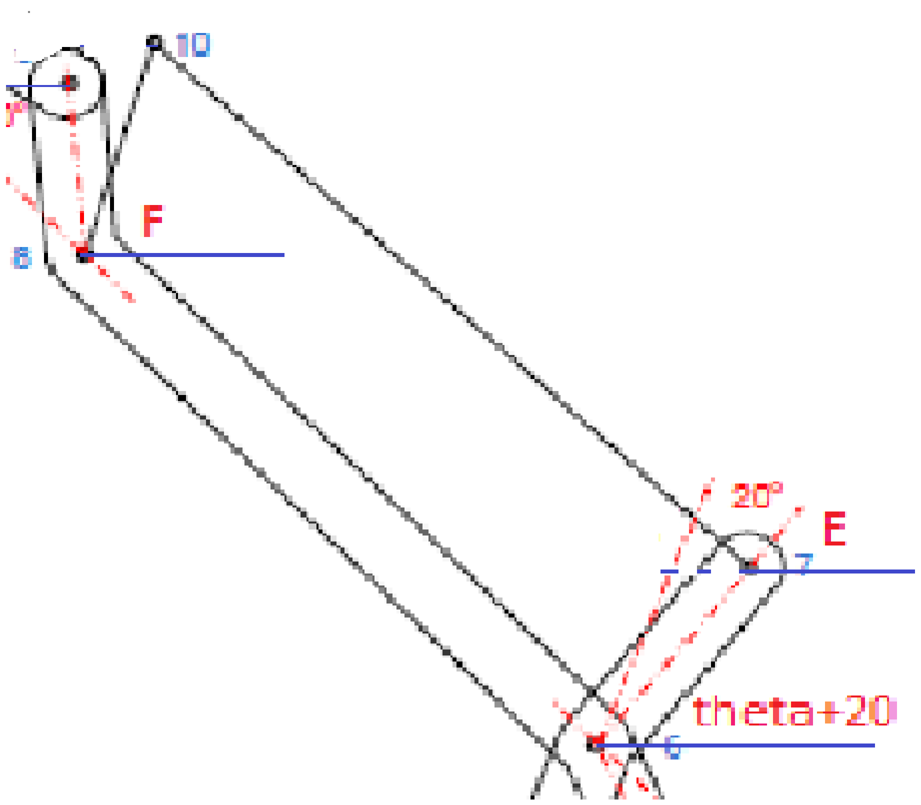
Using kinematics around triangles (0,1,2) and (2,4,6) and the quadrilateral (0,2,6,5) the unknown angles C and D can be expressed in the following matrix in their i and j components.

$$f2 = \begin{bmatrix} f2i \\ f2j \end{bmatrix} = \begin{bmatrix} 4.698\cos(\theta + 20) - 5\cos(D) + \frac{4}{3}\cos(C) - \sqrt{0.5} \\ 4.698\sin(\theta + 20) - 5\sin(D) - \frac{4}{3}\sin(C) + \sqrt{0.5} + 1 \end{bmatrix}$$

The NR algorithm was then used in MATLAB to solve for C and D. This was done by using the following lines of code.

```
%Part 2 - Solution to angles C and D

v2 = [pi/18; pi/2]; %initial guess
%function for Part 2 split into i and j components
f2_i = 4.698*cos(theta+(pi/9))-sqrt(0.5)-5*cos(v2(2))+(4/3)*cos(v2(1));
f2_j = sqrt(0.5)+1+4.698*sin(theta+(pi/9))-5*sin(v2(2))-(4/3)*sin(v2(1));
sum2 = f2_i + f2_j; %initialise test for NR
test2 = abs(sum2);

while test2 > 0.001  %initialise while loop

    %update function with new  guess
    f2_i = 4.698*cos(theta+(pi/9))-sqrt(0.5)-5*cos(v2(2))+(4/3)*cos(v2(1));
    f2_j = sqrt(0.5)+1+4.698*sin(theta+(pi/9))-5*sin(v2(2))-
(4/3)*sin(v2(1));
    sum2 = f2_i + f2_j;      %update test
    test2 = abs(sum2);
    %components of jacobian matrix for NR
    Df2_i = [((-4/3)*sin(v2(1))) (5*sin(v2(2)))]; %i component
    Df2_j = [((-4/3)*cos(v2(1))) (-5*cos(v2(2)))];%j component

    f2 = [f2_i;f2_j];        %create matrix to hold function
    Df2 = [Df2_i;Df2_j];     %create matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted2 = [Df2(4) -Df2(3);-Df2(2) Df2(1)];
    det2 = (1/(Df2(1)*Df2(4)-Df2(2)*Df2(3)));
    inverse2 = matrix_pivoted2*det2;

    %NR algorithm
    v2 = v2 - (inverse2*f2);
end %end while loop
```

Angles E and F were found using kinematics around the quadrilateral (6, 7, 10, 8):



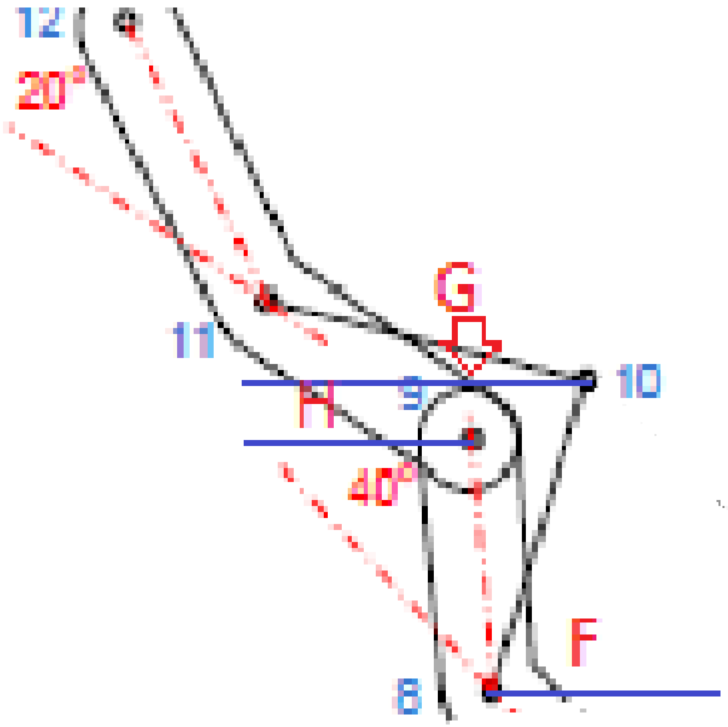
Expressions for angles E and F were worked out by using previously unknown angle C. They are shown below in matrix form:

$$f3 = \begin{bmatrix} f3i \\ f3j \end{bmatrix} = \begin{bmatrix} 4\cos(C-20) + \frac{4}{3}\cos(\theta+20) - \frac{4}{3}\cos(F) - 4.5\cos(E) \\ -4\sin(C-20) + \frac{4}{3}\sin(\theta+20) - \frac{4}{3}\sin(F) + 4.5\sin(E) \end{bmatrix}$$

The NR algorithm was then used in MATLAB to solve for E and F. This was done by using the following lines of code:

```
%Part 3 - solutions to E and F

v3 = [pi/18; pi/2]; %initial guess
%function for Part3 split into i and j components
f3_i = 4*cos(v2(1)-(pi/9))+(4/3)*cos(theta+(pi/9))-(4/3)*cos(v3(2))-
4.5*cos(v3(1));
f3_j = -4*sin(v2(1)-(pi/9))+(4/3)*sin(theta+(pi/9))-
(4/3)*sin(v3(2))+4.5*sin(v3(1));
sum3 = f3_i + f3_j; %test for while loop
test3 = abs(sum3);

while test3 > 0.001  %initialise while loop
    %updated function components
    f3_i = 4*cos(v2(1)-(pi/9))+(4/3)*cos(theta+(pi/9))-(4/3)*cos(v3(2))-
4.5*cos(v3(1));
    f3_j = -4*sin(v2(1)-(pi/9))+(4/3)*sin(theta+(pi/9))-
(4/3)*sin(v3(2))+4.5*sin(v3(1));
    sum3 = f3_i + f3_j;%updated test
    test3 = abs(sum3);
    %compenents of jacobian matrix
    Df3_i = [4.5*sin(v3(1)) (4/3)*sin(v3(2))];  %i component
    Df3_j = [-4.5*cos(v3(1)) (4/3)*cos(v3(2))]; %j component

    f3 = [f3_i;f3_j];     %creates matrix to hold function
    Df3 = [Df3_i;Df3_j]; %creates matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted3 = [Df3(4) -Df3(3);-Df3(2) Df3(1)];
    det3 = (1/(Df3(1)*Df3(4)-Df3(2)*Df3(3)));
    inverse3 = matrix_pivoted3*det3;
    %NR algorithm
    v3 = v3 - (inverse3*f3);
end
```

Angles G and H were found by employing kinematics around the quadrilateral (8, 10, 11, 9).



Expressions for angles G and H were worked out by using previously unknown angles C and F. They are shown below in matrix form:

$$f4 = \begin{bmatrix} f4i \\ f4j \end{bmatrix} = \begin{bmatrix} \cos(C+20) + \frac{4}{3}\cos(F) + \cos(H) - \frac{4}{3}\cos(G) \\ -\sin(C+20) + \frac{4}{3}\sin(F) - \sin(H) + \frac{4}{3}\sin(G) \end{bmatrix}$$

The NR algorithm was then used in MATLAB to solve for G and H. This was done by using the following lines of code:

```
%Part 4 - solutions for G and H

v4 = [pi/18; pi/2]; %initial guess

%functions for Part 4, split into i and j components
f4_i = cos(v2(1)+(pi/9))+(4/3)*cos(v3(2))+cos(v4(2))-(4/3)*cos(v4(1));
f4_j = -sin(v2(1)+(pi/9))+(4/3)*sin(v3(2))-sin(v4(2))+(4/3)*sin(v4(1));
sum4 = f4_i + f4_j; %initial test
test4 = abs(sum4);

while test4 > 0.001 %initialise while loop NR
    %updated function, i and j components
    f4_i = cos(v2(1)+(pi/9))+(4/3)*cos(v3(2))+cos(v4(2))-(4/3)*cos(v4(1));
    f4_j = -sin(v2(1)+(pi/9))+(4/3)*sin(v3(2))-sin(v4(2))+(4/3)*sin(v4(1));
    sum4 = f4_i + f4_j;     %updated test
    test4 = abs(sum4);
    %components of  jacobian matrix
    Df4_i = [((4/3)*sin(v4(1))) (-sin(v4(2)))]; %i component
    Df4_j = [((4/3)*cos(v4(1))) (-cos(v2(2)))]; %j component

    f4 = [f4_i;f4_j];   %creates matrix to hold function
    Df4 = [Df4_i;Df4_j];%creates matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted4 = [Df4(4) -Df4(3);-Df4(2) Df4(1)];
    det4 = (1/(Df4(1)*Df4(4)-Df4(2)*Df4(3)));
    inverse4 = matrix_pivoted4*det4;

    %initialise NR algorithm
    v4 = v4 - (inverse4*f4);
end  %terminate while loop
```

Once all 8 unknown angles were found for the given range of values θ they were stored in a matrix:

```
angles = [v1 v2 v3 v4]; %creates 4x2 matrix to hold all computed angles
```

The following check was then placed on the values to ensure that the matrix only contained values within the range 0° - 360° after the angles were converted from radians to degrees.

```
angle_deg = angles*180/pi; %converts angles in radians to degrees
rev = angle_deg/360;       %calcualates how revolutions the angle is around
                           %the unit circle
whole = floor(rev);        %finds whole number of number of revolutions
angle_deg = angle_deg - (whole*360); %calculates angle in term of 0-360deg
```

This check needs to be in place because there is a chance that the NR algorithm may converge to a value that exists beyond one revolution of the unit circle.

There is a slight error in analyse of the unknown angles that now needs to be corrected. That is that the unknown angles C, G, and H were found with respect to the negative x-axis. This is inconsistent with the other unknown angles as they are in terms of the positive x-axis. This error is corrected with the next few lines of code:

```
%the following three angles C, G and H were found with respect to the
%negative x-axis and now have to be converted to the positive
angle_deg(3) = 180-angle_deg(3);
angle_deg(7) = 180-angle_deg(7);
angle_deg(8) = 180-angle_deg(8);
angle_deg;
```

Vector Points:

All the angles of the system are now known and expressions for the points 0 - 12 can now be found by using kinematics. Each point is represented by an *i* and *j* component and is inserted into MATLAB in the following way:

```
%points to represent position vectors
point0=[sqrt(0.5);0];
point1=[0;sqrt(0.5)];
point2=[0;1+sqrt(0.5)];
point3=[2*cos(v1(1));2*sin(v1(1))];
point4=[2.5*cos(theta);2.5*sin(theta)+point2(2)];
point5=[sqrt(0.5)+5*cos(v2(2));5*sin(v2(2))];
point6=[4.698*cos(theta+(pi/9));point2(2)+4.698*sin(theta+(pi/9))];
point7=[6.032*cos(theta+(pi/9));point2(2)+6.032*sin(theta+(pi/9))];
point8=[point6(1)-4*cos(v2(1)-(pi/9));point6+4*sin(v2(1)-(pi/9))];
point9=[point8(1)-cos(v2(1)+(pi/9));point8(2)+sin(v2(1)+(pi/9))];
point10=[point8(1)+(4/3)*cos(v3(2));point8(2)+(4/3)*sin(v3(2))];
point11=[point10(1)-(4/3)*cos(v4(1));point10(2)+(4/3)*sin(v4(1))];
point12=[point9(1)-
2.506*cos(v4(2)+(pi/9));point9(2)+2.506*sin(v4(2)+(pi/9))];
```

Simulation:

All angles vector points are now known with respect to the angle θ. The next step is simulation. I decided the best way to simulate the system would be to plot a figure in MATLAB that contained all the major links and minor links. This was done with the following code:

```
%Simulation

hold on  %command to hold a figure so that other plots can be added to it

%code to create vectors to hold points to plot the first major link
major_link1x=[point2(1) point4(1) point6(1) point7(1)];%vector to hold x
                                                      %components
major_link2y=[point2(2) point4(2) point6(2) point7(2)];%vector to hold y
                                                      %component
p1= plot(major_link1x, major_link2y); %code to plot first major link and
set(p1,'Color','red','LineWidth',8)   %set its properties

%code to create vectors to hold points to plot the second major link
major_link2x=[point5(1) point6(1) point8(1) point9(1)];%vector to hold x
                                                      %components
major_link2y=[point5(2) point6(2) point8(2) point9(2)];%vector to hold y
                                                      %component
p2=plot(major_link2x, major_link2y);    %code to plot second major link and
set(p2,'Color','magenta','LineWidth',8);%set its properties
```

```
%code to create vectors to hold points to plot the third major link
major_link3x=[point9(1) point11(1) point12(1)];%vector to hold x
                                               %components
major_link3y=[point9(2) point11(2) point12(2)];%vector to hold y
                                               %component
p3=plot(major_link3x, major_link3y);%code to plot third major link and
set(p3,'Color','cyan','LineWidth',8);%set its properties

%code to create vectors to hold points to plot the first minor link
minor_link1x=[point0(1) point5(1)];%x component
minor_link1y=[point0(2) point5(2)];%y component
p4=plot(minor_link1x, minor_link1y);%code to plot first minor link and
set(p4,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the second minor link
minor_link2x=[point1(1) point3(1) point4(1)];%x component
minor_link2y=[point1(2) point3(2) point4(2)];%y component
p5=plot(minor_link2x, minor_link2y);%code to plot second minor link and
set(p5,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the third minor link
minor_link3x=[point7(1) point10(1)];%x component
minor_link3y=[point7(2) point10(2)];%y component
p6=plot(minor_link3x, minor_link3y);%code to plot third major link and
set(p6,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the fourth minor link
minor_link4x=[point10(1) point8(1)];%x component
minor_link4y=[point10(2) point8(2)];%y component
p7=plot(minor_link4x, minor_link4y);%code to plot fourth major link and
set(p7,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the fifth minor link
minor_link5x=[point10(1) point11(1)];%x component
minor_link5y=[point10(2) point11(2)];%y component
p8=plot(minor_link5x, minor_link5y);%code to plot fifth major link and
set(p8,'Color','black','LineWidth',1);%set its properties
```

This code creates a figure of the static system with respect to a single value of θ. The overall goal of the project is to make a MATLAB movie that simulates the system over a range of values of θ. This was done by making a movie were each frame consisted of a figure plot of the system for a different angle of θ. This was achieved by employing the following lines of code:

```
%code to make movie

%initiate for loop to create vector to hold handles of figures to make
%movie
for i = gcf                  %sets i equal to handle of currrent figure
    handlefig(i) = getframe([i]);%stores movie frame as an element of
                                 %vector handlefig
end %end for loop
end %end for loop
close all                    %command to close all current figures
figure;                      %open new figure
axis off                     %turn off axis for film

M = handlefig;               %sets handlefig equal to M
```

```
FPS=1;                       %sets movie speed to one frame per second
N=-2;                        %plays movie forward then backwards TWICE!!!:D
axis([-2 4 0 10]);           %sets axis

movie(M,N,FPS)               %plays movie
```

Errors in Code:

While I do have a working simulation of the system it is clear that it is not perfect. There are several errors both in the simulation and in my angle finding algorithm:

1. the simulation is not centred, i.e. cannot see the whole simulation when movie is running
2. NR algorithm only converges for certain ranges of θ. Works best for angles 30°-60°.
3. The algorithm that I have in place to decide when the NR algorithm has converged enough does seem to sufficient enough to cover all angles of θ even between 30°-60°. My range of θ increments in steps of 0.05 radians, when this is changed to a smaller increment e.g. 0.01 the compiler will not exit the while loop. Because I have to choose such a large increment to makes the movie appear random and not smooth. However this could also be due to errors in expressions for the unknown angles.

M- file Function solution():

The function I have written to solve this problem is designed to take two arguments – the lower limit of θ and the upper limit of θ. There is error checking code in place to ensure that the upper limit is not inputted as the lower limit and visa-versa. There is also error code in place to ensure that the lower limit does not go less than 30° and the upper limit does not go above 60°. This was implemented as follows:

```
function [angle_deg] = solution(X1,X2)     %declares function

%function error messaging
if X1>X2
   error('Invalid input arguments: argument 2 must be greater than argument 
1')
end
if X1<pi/6
   error('Invalid input arguments: recommended input for argument 1: 
X1>=pi/6')
end
if X2>pi/3
   error('Invalid input arguments: recommended input for argument 2: 
X2<=pi/3')
end
```

The Functions use is to take the two input arguments, calculate and output the unknown angles with respect to the lower limit right through to the upper limit in increments of 0.05. This was done by implementing the following for loop:

```
for theta=X1:0.05:X2  %initialise for loop for range of inputted values
```

The function then takes these values, calculates the vector points for each iteration with respect to these values and carries out the simulation, outputting the movie at the end.

I also wrote a simple help file for my function Solution() briefly explaining how to use it and what it does. It can be seen below:

```
%function [angle_deg] = solution()
%solution() outputs unknown values of theta with respect to range inputted
%and creates movie of system for this inputted range

%solution(X1,X2) computes the unknown angles of the system and calculates
%the vector positions for a range of angles X1 to X2 before creating a
%visualisation of the systems behaviour for this range.
%
%X1 and X2 are input radians and their expected range is pi/6-pi/3
%An Error message will occur if X1 is not less than X2
%Function only works for X1>=pi/6 and X2<=pi/3
```

Conclusion:

I genuinely enjoyed this project as it required me to employ my knowledge of kinematics as well as testing my ability to write efficient algorithms to solve static equations numerically. Looking back on the project I see several faults as well as areas that need improvement that I will address in the future, especially as I find MATLAB a very exciting tool that I wish someday to have a very high knowledge off.

Below is my m-file in its entirety:

```
%function [angle_deg] = solution()
%solution() outputs unknown values of theta with respect to range inputted
%and creates movie of system for this inputted range

%solution(X1,X2) computes the unknown angles of the system and calculates
%the vector positions for a range of angles X1 to X2 before creating a
%visualisation of the systems behaviour for this range.
%
%X1 and X2 are input radians and their expected range is pi/6-pi/3
%An Error message will occur if X1 is not less than X2
%Function only works for X1>=pi/6 and X2<=pi/3

function [angle_deg] = solution(X1,X2)    %declares function

%function error messaging
if X1>X2
   error('Invalid input arguments: argument 2 must be greater than argument
1')
end
if X1<pi/6
   error('Invalid input arguments: recommended input for argument 1:
X1>=pi/6')
end
if X2>pi/3
   error('Invalid input arguments: recommended input for argument 2:
X2<=pi/3')
end
```

```
close all   %command to close all figures before running code

for theta=X1:0.05:X2  %initialise for loop for range of inputted values

figure;     %command to create new figure for each iteration of theta

%Part 1 - Solution to angles A and B

v1 = [pi/18; pi/2];     %initial guess for solutions of newton-raphson (NR)
%function for part 1
f1_i = 2*cos(v1(1)) + 2*cos(v1(2)) - (2.5)*cos(theta); %i component
f1_j = 2*sin(v1(1)) + 2*sin(v1(2)) - (2.5)*sin(theta) - 1; %j component
sum1 = f1_i + f1_j;         %initialising test for convergence of NR
test1 = abs(sum1);

%initiate while loop to iterate NR
while test1 > 0.001

%update function with new guess
f1_i = 2*cos(v1(1)) + 2*cos(v1(2)) - (2.5)*cos(theta);
f1_j = 2*sin(v1(1)) + 2*sin(v1(2)) - (2.5)*sin(theta) - 1;
sum1 = f1_i + f1_j; %update test
test1 = abs(sum1);
%find components of Jacobian matrix for NR
Df1_i = [(-2*sin(v1(1))) (-2*sin(v1(2)))]; %i component
Df1_j = [(2*cos(v1(1))) (2*cos(v1(2)))];   %j component

f1 = [f1_i;f1_j];  %create matrix to hold function
Df1 = [Df1_i;Df1_j]; %create matrix to hold jacobian

%the following three lines of code are to find the inverse matrix of the
%jacobian
matrix_pivoted1 = [Df1(4) -Df1(3);-Df1(2) Df1(1)]; %swas first and fourth
%elements of jacobian and changes sign of second end third elements
det1 = (1/(Df1(1)*Df1(4)-Df1(2)*Df1(3))); %finds determinant of jacobian
inverse1 = matrix_pivoted1*det1; %finds inverse of jacobian
%NR algorithm:
v1 = v1 - (inverse1*f1);
end     %end while loop

%Part 2 - Solution to angles C and D

v2 = [pi/18; pi/2]; %initial guess
%function for Part 2 split into i and j components
f2_i = 4.698*cos(theta+(pi/9))-sqrt(0.5)-5*cos(v2(2))+(4/3)*cos(v2(1));
f2_j = sqrt(0.5)+1+4.698*sin(theta+(pi/9))-5*sin(v2(2))-(4/3)*sin(v2(1));
sum2 = f2_i + f2_j; %initialise test for NR
test2 = abs(sum2);

while test2 > 0.001  %initialise while loop

    %update function with new  guess
    f2_i = 4.698*cos(theta+(pi/9))-sqrt(0.5)-5*cos(v2(2))+(4/3)*cos(v2(1));
    f2_j = sqrt(0.5)+1+4.698*sin(theta+(pi/9))-5*sin(v2(2))-
(4/3)*sin(v2(1));
    sum2 = f2_i + f2_j;     %update test
    test2 = abs(sum2);
    %components of jacobian matrix for NR
    Df2_i = [((-4/3)*sin(v2(1))) (5*sin(v2(2)))]; %i component
    Df2_j = [((-4/3)*cos(v2(1))) (-5*cos(v2(2)))];%j component

    f2 = [f2_i;f2_j];        %create matrix to hold function
    Df2 = [Df2_i;Df2_j];     %create matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted2 = [Df2(4) -Df2(3);-Df2(2) Df2(1)];
    det2 = (1/(Df2(1)*Df2(4)-Df2(2)*Df2(3)));
    inverse2 = matrix_pivoted2*det2;

    %NR algorithm
    v2 = v2 - (inverse2*f2);
end %end while loop

%Part 3 - solutions to E and F

v3 = [pi/18; pi/2]; %initial guess
%function for Part3 split into i and j components
f3_i = 4*cos(v2(1)-(pi/9))+(4/3)*cos(theta+(pi/9))-(4/3)*cos(v3(2))-
4.5*cos(v3(1));
f3_j = -4*sin(v2(1)-(pi/9))+(4/3)*sin(theta+(pi/9))-
(4/3)*sin(v3(2))+4.5*sin(v3(1));
sum3 = f3_i + f3_j; %test for while loop
test3 = abs(sum3);

while test3 > 0.001  %initialise while loop
    %updated function components
    f3_i = 4*cos(v2(1)-(pi/9))+(4/3)*cos(theta+(pi/9))-(4/3)*cos(v3(2))-
4.5*cos(v3(1));
    f3_j = -4*sin(v2(1)-(pi/9))+(4/3)*sin(theta+(pi/9))-
(4/3)*sin(v3(2))+4.5*sin(v3(1));
    sum3 = f3_i + f3_j;%updated test
    test3 = abs(sum3);
    %compenents of jacobian matrix
    Df3_i = [4.5*sin(v3(1)) (4/3)*sin(v3(2))];  %i component
    Df3_j = [-4.5*cos(v3(1)) (4/3)*cos(v3(2))]; %j component

    f3 = [f3_i;f3_j];     %creates matrix to hold function
    Df3 = [Df3_i;Df3_j]; %creates matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted3 = [Df3(4) -Df3(3);-Df3(2) Df3(1)];
    det3 = (1/(Df3(1)*Df3(4)-Df3(2)*Df3(3)));
    inverse3 = matrix_pivoted3*det3;
    %NR algorithm
    v3 = v3 - (inverse3*f3);
end

%Part 4 - solutions for G and H

v4 = [pi/18; pi/2]; %initial guess

%functions for Part 4, split into i and j components
f4_i = cos(v2(1)+(pi/9))+(4/3)*cos(v3(2))+cos(v4(2))-(4/3)*cos(v4(1));
f4_j = -sin(v2(1)+(pi/9))+(4/3)*sin(v3(2))-sin(v4(2))+(4/3)*sin(v4(1));
sum4 = f4_i + f4_j; %initial test
test4 = abs(sum4);

while test4 > 0.001 %initialise while loop NR
```

```
    %updated function, i and j components
    f4_i = cos(v2(1)+(pi/9))+(4/3)*cos(v3(2))+cos(v4(2))-(4/3)*cos(v4(1));
    f4_j = -sin(v2(1)+(pi/9))+(4/3)*sin(v3(2))-sin(v4(2))+(4/3)*sin(v4(1));
    sum4 = f4_i + f4_j;     %updated test
    test4 = abs(sum4);
    %components of  jacobian matrix
    Df4_i = [((4/3)*sin(v4(1))) (-sin(v4(2)))]; %i component
    Df4_j = [((4/3)*cos(v4(1))) (-cos(v2(2)))]; %j component

    f4 = [f4_i;f4_j];   %creates matrix to hold function
    Df4 = [Df4_i;Df4_j];%creates matrix to hold jacobian

    %get inverse of jacobian matrix
    matrix_pivoted4 = [Df4(4) -Df4(3);-Df4(2) Df4(1)];
    det4 = (1/(Df4(1)*Df4(4)-Df4(2)*Df4(3)));
    inverse4 = matrix_pivoted4*det4;

    %initialise NR algorithm
    v4 = v4 - (inverse4*f4);
end  %terminate while loop

angles = [v1 v2 v3 v4]; %creates 4x2 matrix to hold all computed angles

%the following lines of code are a check to ensure that the output displays
%the angles in terms of less than one evolution of the unit circle. this is
%needed because at times the NR algorithm may approximate the angle at an
%angle greater than 360 degrees or 2pi radians
angle_deg = angles*180/pi; %converts angles in radians to degrees
rev = angle_deg/360;       %calcualates how revolutions the angle is around
                           %the unit circle
whole = floor(rev);        %finds whole number of number of revolutions
angle_deg = angle_deg - (whole*360); %calculates angle in term of 0-360deg
%the following three angles C, G and H were found with respect to the
%negative x-axis and now have to be converted to the positive
angle_deg(3) = 180-angle_deg(3);
angle_deg(7) = 180-angle_deg(7);
angle_deg(8) = 180-angle_deg(8);
angle_deg;

%points to represent position vectors
point0=[sqrt(0.5);0];
point1=[0;sqrt(0.5)];
point2=[0;1+sqrt(0.5)];
point3=[2*cos(v1(1));2*sin(v1(1))];
point4=[2.5*cos(theta);2.5*sin(theta)+point2(2)];
point5=[sqrt(0.5)+5*cos(v2(2));5*sin(v2(2))];
point6=[4.698*cos(theta+(pi/9));point2(2)+4.698*sin(theta+(pi/9))];
point7=[6.032*cos(theta+(pi/9));point2(2)+6.032*sin(theta+(pi/9))];
point8=[point6(1)-4*cos(v2(1)-(pi/9));point6+4*sin(v2(1)-(pi/9))];
point9=[point8(1)-cos(v2(1)+(pi/9));point8(2)+sin(v2(1)+(pi/9))];
point10=[point8(1)+(4/3)*cos(v3(2));point8(2)+(4/3)*sin(v3(2))];
point11=[point10(1)-(4/3)*cos(v4(1));point10(2)+(4/3)*sin(v4(1))];
point12=[point9(1)-
2.506*cos(v4(2)+(pi/9));point9(2)+2.506*sin(v4(2)+(pi/9))];

%Simulation

hold on  %command to hold a figure so that other plots can be added to it
```

```
%code to create vectors to hold points to plot the first major link
major_link1x=[point2(1) point4(1) point6(1) point7(1)];%vector to hold x
                                                       %components
major_link2y=[point2(2) point4(2) point6(2) point7(2)];%vector to hold y
                                                       %component
p1= plot(major_link1x, major_link2y); %code to plot first major link and
set(p1,'Color','red','LineWidth',8)   %set its properties

%code to create vectors to hold points to plot the second major link
major_link2x=[point5(1) point6(1) point8(1) point9(1)];%vector to hold x
                                                       %components
major_link2y=[point5(2) point6(2) point8(2) point9(2)];%vector to hold y
                                                       %component
p2=plot(major_link2x, major_link2y);    %code to plot second major link and
set(p2,'Color','magenta','LineWidth',8);%set its properties

%code to create vectors to hold points to plot the third major link
major_link3x=[point9(1) point11(1) point12(1)];%vector to hold x
                                               %components
major_link3y=[point9(2) point11(2) point12(2)];%vector to hold y
                                               %component
p3=plot(major_link3x, major_link3y);%code to plot third major link and
set(p3,'Color','cyan','LineWidth',8);%set its properties

%code to create vectors to hold points to plot the first minor link
minor_link1x=[point0(1) point5(1)];%x component
minor_link1y=[point0(2) point5(2)];%y component
p4=plot(minor_link1x, minor_link1y);%code to plot first minor link and
set(p4,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the second minor link
minor_link2x=[point1(1) point3(1) point4(1)];%x component
minor_link2y=[point1(2) point3(2) point4(2)];%y component
p5=plot(minor_link2x, minor_link2y);%code to plot second minor link and
set(p5,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the third minor link
minor_link3x=[point7(1) point10(1)];%x component
minor_link3y=[point7(2) point10(2)];%y component
p6=plot(minor_link3x, minor_link3y);%code to plot third major link and
set(p6,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the fourth minor link
minor_link4x=[point10(1) point8(1)];%x component
minor_link4y=[point10(2) point8(2)];%y component
p7=plot(minor_link4x, minor_link4y);%code to plot fourth major link and
set(p7,'Color','black','LineWidth',1);%set its properties

%code to create vectors to hold points to plot the fifth minor link
minor_link5x=[point10(1) point11(1)];%x component
minor_link5y=[point10(2) point11(2)];%y component
p8=plot(minor_link5x, minor_link5y);%code to plot fifth major link and
set(p8,'Color','black','LineWidth',1);%set its properties

%code to make movie

%initiate for loop to create vector to hold handles of figures to make
%movie
for i = gcf                  %sets i equal to handle of currrent figure
```

```
    handlefig(i) = getframe([i]);%stores movie frame as an element of
                                 %vector handlefig
end %end for loop
end %end for loop
close all                    %command to close all current figures
figure;                      %open new figure
axis off                     %turn off axis for film

%handlefig = figure('position', [10000, 10000, 400, 300]);
M = handlefig;               %sets handlefig equal to M

FPS=1;                       %sets movie speed to one frame per second
N=-2;                        %plays movie forward then backwards. TWICE!!!:D
axis([-2 4 0 10]);           %sets axis

movie(M,N,FPS)               %plays movie
```