7.5

I used [1] in making this assignment when finished I compared initial values with Prajish Kumar (4743873).	**Rick Staa, #4511328** **HW set 3 due 13/03/2018** **ME41060**

1. Statement of integrity

my homework is completely in accordance with the Academic Integrity

2. Problem statement

In this assignment we use the TMT method to model a simple passive dynamic walker (PDW). This model consists of two slender legs hinged at the hip. The two legs are identical, with length l and mass m (distributed equally around the legs). The body of this PDW is modelled as an extra point mass M at the hip. Further he ground this model is walking on is tilted by an angle γ. The generalized coordinates used in this model are the angle of the stance leg α w.r.t. the ground normal and the angle of the swing leg β w.r.t. the ground normal. This results in the following state $q_i = [\alpha, \beta]$.

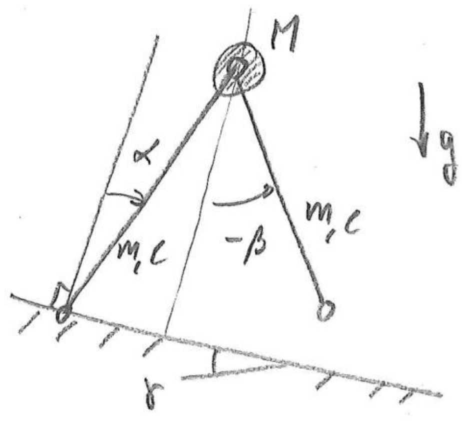

Figure 1: A sketch of the Passive Dynamic walker used in this assignment.

3. Method

We were asked to derive the equations of motion of the PDW by means of two methods, The LaGrange method and the Virtual power TMT method. In this section both methods will be discussed. However since we already extensively reviewed the Lagrange method only a brief overview of this method will be given.

3.1 Lagrange Method

This method makes use of the conservation of energy principle and Lagrange Multipliers to derive the equations of motions. This is done by Solving the following LaGrange constrain equation:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = Q_j \qquad (1)$$

Where T is the kinetic energy of the system, V the potential energy and Q_j generalised forces. This last one is equal to zero in this problem. When we rewrite this formula in a matrix vector product with the known terms at the right side we get:

$$\frac{\partial}{\partial \dot{q}}\left(\frac{\partial T}{\partial \dot{q}_i}\right)\ddot{q} = Qi - \frac{\partial}{\partial q}\left(\frac{\partial T}{\partial \dot{q}_i}\right)\dot{q} - \frac{\partial V}{\partial q_i} + \frac{\partial T}{\partial q_i} \qquad (1)$$

This results in the following matrix vector product:

$$M_{ij}\ddot{q}_j = F_i \qquad (2)$$

The kinetic energy and potential energy can be calculated as follows:

<u>Kinetic energy</u>

$$T = \frac{1}{2}x_j^T M_{ij} x_j \qquad (2)$$

Where M_{ij} is the mass matrix:

$$M_{ij} = [m\; m\; I\; M\; M\; 0\; m\; m\; I] \qquad (3)$$

And x_j are the positions of the COM's of the segments expressed in the generalised coordinates: The global coordinate system was said to be perpendicular to the slope since this made the calculations easier.

$$x_j = \begin{bmatrix} x_1 \\ y_1 \\ \varphi_1 \\ x_2 \\ y_2 \\ \varphi_2 \\ x_3 \\ y_3 \\ \varphi_3 \end{bmatrix} = \begin{bmatrix} \frac{l}{2}\sin\alpha \\ \frac{l}{2}\cos\alpha \\ \frac{\pi}{2}-\alpha \\ l\sin\alpha \\ l\cos\alpha \\ 0 \\ l\sin\alpha + \frac{l}{2}\sin\left(\frac{\pi}{2}+\beta\right) \\ l\cos\alpha + \frac{l}{2}\cos\left(\frac{\pi}{2}+\beta\right) \\ \frac{\pi}{2}+\beta \end{bmatrix} \tag{4}$$

No EOM given -1

<u>Potential energy</u>

The potential energy of the system can be calculated as follows:

$$V = -Fx_j \tag{5}$$

In which F is the vector with potential forces as described in the global coordinate system:

$$F = [mgsin(\gamma) \quad -mgcos(\gamma) \quad 0 \quad Mgsin(\gamma) \quad -Mgcos(\gamma) \quad 0 \quad 0 \quad mgsin(\gamma) \quad -mgcos(\gamma) \quad 0\,] \tag{6}$$

For the implementation you are referred to appendix A.

3.2 Virtual Power TMT method

The Virtual Power TMT method is like the Lagrange method but differs in the fact that it doesn't go into the energy domain. Instead it stays in the forces domain and uses an incremental approach to obtain the equations of motion. We can derive the method by first looking at the virtual power equation with included D'Alembert forces:

$$\delta P = (F - M\ddot{\boldsymbol{x}})\delta\dot{\boldsymbol{x}} \tag{7}$$

Following the TMT method makes use of a transformation matrix T to transform the COM positions to the generalised coordinates. As a result we obtain the following equation:

$$\delta P = \delta\dot{x}_i(F_i - M_{ik}\ddot{\boldsymbol{x}}_{\boldsymbol{k}}) + \delta\dot{q}_k Q_j \tag{8}$$

no EOM given -1

Following we can derive the virtual accelerations which fulfil the constraints in equation 3 and fill this in in equation 8. After noting that this equation must hold for all virtual velocities and rearranging the equation a bit we obtain:

$$\bar{M}\ddot{q} = \bar{f} \tag{9}$$

Where:

$$\bar{M} = T^T MT \text{ and } \bar{f} = T^T(F - mG) \tag{10}$$

In this T represents the transformation matrix which can be calculated by taking the Jacobian of the states x w.r.t. the general coordinates. The M is again the mass matrix and the F contains the applied forces and moment at the COM's of the segments. The new G matrix is a convective term that arises due to deriving the virtual accelerations. The MATLAB code implementing the TMT method can be found in Appendix B.

4. Results and discussion

0.5

To rule out random effects two sets of parameters and initial states were used to compare the both methods. This comparison can be found below:

4.1 Results and discussion

For the first test the following parameters were used:

Parameter	Value
L	$0.6\ m$
m	$0.3\ kg$
M	$1\ kg$
g	$9.81\ m/s$
γ	$5\ deg$

Table 1: Parameters used

The following initial states were used:

Initial states	Value
α	$25\ deg$
β	$40\ deg$
$\dot{\alpha}$	$-180\frac{deg}{s}$
$\dot{\beta}$	$0\frac{deg}{s}$

Table 2: Initial states used

The results for both methods are shown in table 3 on the next page. From this table both methods give the same result.

	Lagrange	*TMT*
$\ddot{x}_1$	$1.09\,\frac{m}{s^2}$	$1.09\,\frac{m}{s^2}$
$\ddot{y}_1$	$-3.78\,\frac{m}{s^2}$	$-3.78\,\frac{m}{s^2}$
$\ddot{\varphi}_1$	$-8.65\,\frac{rad}{s^2}$	$-8.65\,\frac{rad}{s^2}$
$\ddot{x}_2$	$2.20\,\frac{m}{s^2}$	$2.20\,\frac{m}{s^2}$
$\ddot{y}_2$	$-7.59\,\frac{m}{s^2}$	$-7.59\,\frac{m}{s^2}$
$\ddot{\varphi}_2$	$0\,\frac{rad}{s^2}$	$0\,\frac{rad}{s^2}$
$\ddot{x}_3$	$0.96\,\frac{m}{s^2}$	$0.96\,\frac{m}{s^2}$
$\ddot{y}_3$	$-9.03\,\frac{m}{s^2}$	$-9.03\,\frac{m}{s^2}$
$\ddot{\varphi}_3$	$6.40\,\frac{rad}{s^2}$	$6.40\,\frac{rad}{s^2}$

Table 3: Results for given initial conditions

4.1 Results and discussion

For the first test the following parameters were used:

Parameter	**Value**
L	$1.2m$
m	$12\,kg$
M	$38\,kg$
g	$9.81\,m/s$
γ	$15\,deg$

Table 4: Parameters used

The following initial states were used:

Initial states	**Value**
α	$20\,deg$
β	$30\,deg$
$\dot{\alpha}$	$-90\,\frac{deg}{s}$
$\dot{\beta}$	$10\,\frac{deg}{s}$

Table 5: Initial states used

The results for both methods are shown in table 6 on the next page. From this table both methods give the same result.

does it make sense that the results are the same? no discussion -0.5

	Lagrange	

$\ddot{x}_1$	$8.50\,\frac{m}{s^2}$	$8.50\,\frac{m}{s^2}$
$\ddot{y}_1$	$-4.67\,\frac{m}{s^2}$	$-4.67\,\frac{m}{s^2}$
$\ddot{\varphi}_1$	$-15.98\,\frac{rad}{s^2}$	$-15.98\,\frac{rad}{s^2}$
$\ddot{x}_2$	$17.01\,\frac{m}{s^2}$	$17.01\,\frac{m}{s^2}$
$\ddot{y}_2$	$-9.3418\,\frac{m}{s^2}$	$-9.3418\,\frac{m}{s^2}$
$\ddot{\varphi}_2$	$0\,\frac{rad}{s^2}$	$0\,\frac{rad}{s^2}$
$\ddot{x}_3$	$-37.71\,\frac{m}{s^2}$	$-37.71\,\frac{m}{s^2}$
$\ddot{y}_3$	$15.89\,\frac{m}{s^2}$	$15.89\,\frac{m}{s^2}$
$\ddot{\varphi}_3$	$9.19\,\frac{rad}{s^2}$	$9.19\,\frac{rad}{s^2}$

Table 6: Results for given initial conditions

4. Cons and Pros

Without looking at adding additional passive and active elements the TMT method is easier to implement since you only are calculating the transformation matrix. As a result, the TMT method is less prone to errors and it is easier to add additional segments. Although here was no noticeable difference for this small system it is possibly also computationally faster when working with larger systems since less differentials need to be evaluated. Further the TMT method is easier to interpret compared to the LaGrange method you now stay in the state and force space. The downside is however that you don't directly get any information about the energy of the system, but 1 this can be easily calculated within the algorithm. 3

Appendix A (Lagrange)

MATLAB CODE

1

```
%% MBD_B: Assignment 5 - Passive dynamic walker (Lagrange and TMT aproach)
%  Rick Staa (4511328)
%  Last edit: 27/03/2018
% - Question A: Lagrange method -

% clear all; close all; clc;

%% Parameters and variables
% Initialise parameters and variables
syms l m I M                                    % Initialize model Parameters
syms x1 y1 phi1 x2 y2 phi2 x3 y3 phi3                % Initialize variables
syms x1p y1p phi1p x2p y2p phi2p x3p y3p phi3p            % Initialize derivatives
syms alpha beta alphap betap                      % Initialize generalised coordinates
syms gamma g                                      % Initialize enviroment variables

% Calculate additional parameters
I     = (1/12)*m*(l^2);                            % Mass moment of inertia of each leg about the COM
```

```
% Devine state and its derivative
q       = [alpha;beta];
qp      = [alphap;betap];

%% Generate equations of motion

% Express coordinates of the COM in terms of generalised coordinates
x1      = 0.5*l*sin(alpha);
y1      = 0.5*l*cos(alpha);
phi1    = 0.5*pi - alpha;
x2      = l*sin(alpha);
y2      = l*cos(alpha);
phi2    = 0;
x3      = l*sin(alpha) + 0.5*l*sin(0.5*pi + beta);
y3      = l*cos(alpha) + 0.5*l*cos(0.5*pi + beta);
phi3    = 0.5*pi + beta;

% Put in one state vector
x       = [x1;y1;phi1;x2;y2;phi2;x3;y3;phi3];

% Compute the jacobian of these expressions
Jx_q    = simplify(jacobian(x,q));

% Calculate derivative of the state vector
xp      = Jx_q*qp;

% Compute energies
T       = 0.5*xp.'*diag([m;m;I;M;M;0;m;m;I])*xp;          % Kinetic energy
V       = -([m*g*sin(gamma) -m*g*cos(gamma) 0 M*g*sin(gamma) -M*g*cos(gamma) 0 m*g*sin(gamma) -m*g*cos(gamma) 0]*x);          % Potential energy

%% Now compute the lagrangian constraint equations of motion

% Compute partial derivatives w.r.t q
T_q     = simplify(jacobian(T,q.')).';                 % Term 2 in Lagrange equation reader
V_q     = simplify(jacobian(V,q.')).';                 % Term 3 ...

% Compute partial derivatives w.r.t. qp
T_qp    = simplify(jacobian(T,qp.')).';

% Compute terms of total derivative
T_qp_qp  = simplify(jacobian(T_qp,qp.'));
T_qp_q   = simplify(jacobian(T_qp,q.'));

% Combine matrixes to get the lagrangian euqations of motion in matrix
% vector form
Qj      = 0;                                           % Non-conservative forces
Q       = T_qp_qp;
F       = (-T_qp_q*qp + T_q - V_q + Qj);

% Calculate result expressed in generalized coordinates
qdp     = Q\F;

% Get result back in COM coordinates
xdp     = simplify(jacobian(xp,qp.'))*qdp + simplify(jacobian(xp,q.'))*qp;

%% Calculate for a initial state
x0     = [deg2rad(20),deg2rad(30),deg2rad(-90),10];
parms = [1.2,12,9.81,38,deg2rad(15)];                  % parms = [l,m,g,M,gamma];
xdp = subs(xdp, [l,m,g,M,gamma],parms);
xdp = double(subs(xdp, [alpha,beta,alphap,betap],[x0]));
disp(xdp)
```

Appendix B (TMT)

MATLAB CODE

```
%% MBD_B: Assignment 5 - Passive dynamic walker (Lagrange and TMT aproach)
%  Rick Staa (4511328)
%  Last edit: 27/03/2018
% - Question A: Lagrange method -

clear all; close all; clc;

%% Parameters and variables
% Initialise parameters and variables
syms l m I M                                    % Initialize model Parameters
syms x1 y1 phi1 x2 y2 phi2 x3 y3 phi3           % Initialize variables
syms x1p y1p phi1p x2p y2p phi2p x3p y3p phi3p  % Initialize derivatives
syms alpha beta alphap betap                    % Initialize generalised coordinates
syms gamma g                                    % Initialize enviroment variables

% Calculate additional parameters
I       = (1/12)*m*(l^2);                       % Mass moment of inertia of each leg about the COM

% Devine state and its derivative
q       = [alpha;beta];
qp      = [alphap;betap];

%% Generate equations of motion

% Express coordinates of the COM in terms of generalised coordinates
x1      = 0.5*l*sin(alpha);
y1      = 0.5*l*cos(alpha);
phi1    = 0.5*pi - alpha;
x2      = l*sin(alpha);
y2      = l*cos(alpha);
phi2    = 0;
x3      = l*sin(alpha) + 0.5*l*sin(0.5*pi + beta);
y3      = l*cos(alpha) + 0.5*l*cos(0.5*pi + beta);
phi3    = 0.5*pi + beta;

% Put in one state vector
x       = [x1;y1;phi1;x2;y2;phi2;x3;y3;phi3];

% Compute the jacobian of these expressions
Jx_q    = simplify(jacobian(x,q.'));

% Calculate derivative of the state vector
xp      = Jx_q*qp;

% Solve with virtual power
M_bar   = Jx_q.'*diag([m,m,I,M,M,0,m,m,I])*Jx_q;
F       = [m*g*sin(gamma), -m*g*cos(gamma), 0, M*g*sin(gamma), -M*g*cos(gamma), 0, m*g*sin(gamma), -m*g*cos(gamma), 0];
T_qq    = simplify(jacobian(Jx_q*qp,q)*qp);
Q       = simplify(Jx_q.'*(F.' - diag([m,m,I,M,M,0,m,m,I])*T_qq));

% Calculate result expressed in generalized coordinates
qdp     = M_bar\Q;
```

```
% Get result back in COM coordinates
xdp     = simplify(jacobian(xp,qp.'))*qdp + simplify(jacobian(xp,q.'))*qp;

%% Calculate for a initial state
x0    = [deg2rad(20),deg2rad(30),deg2rad(-90),10];
parms = [1.2,12,9.81,38,deg2rad(15)];                                   % parms = [l,m,g,M,gamma];
xdp = subs(xdp, [l,m,g,M,gamma],parms);
xdp = double(subs(xdp, [alpha,beta,alphap,betap],[x0]));
disp(xdp)
```