Acknowledgements: I compared my results with those from Rafael Agiro and Nathan Timmers and Rutger Scheurs. Rick Waasdorp pointed out which equation to use for the Gauss-Newton method.

My homework is entirely in accordance with the acedemic integrity.

Homework Assignment 8

A new system is analyzed, namely the EzyRoller shown in figure 1.

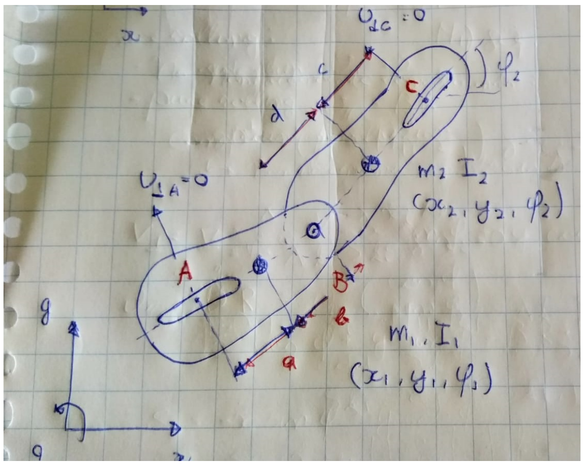

Figure 1: Problem overview

1.1 Equations of motion

The nonholonomic constraints can be determined by noting that $v_{A_\perp} = 0$ and $v_{C_\perp} = 0$. The velocity in A can be expressed in terms of the velocities at cm$_1$,

$$\begin{aligned}
\mathbf{v}_A &= \mathbf{v}_1 + \omega_1 \times \mathbf{r}_{A/cm} \\
&= \begin{pmatrix} \dot{x}_1 \\ \dot{y}_1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dot{\varphi}_1 \end{pmatrix} \times \begin{pmatrix} -a\cos\varphi_1 \\ -a\sin\varphi_1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} \dot{x}_1 + a\dot{\varphi}_1 \sin\varphi_1 \\ \dot{y}_1 - a\dot{\varphi}_1\cos\varphi_1 \\ 0 \end{pmatrix}
\end{aligned} \tag{1}$$

Furthermore

$$\begin{aligned}
v_{A_\perp} &= \mathbf{v}_A \begin{pmatrix} -\sin\varphi_1 \\ \cos\varphi_1 \\ 0 \end{pmatrix} \\
&= -\dot{x}_1 \sin\varphi_1 + \dot{y}_1\cos\varphi_1 - a\dot{\varphi}_1
\end{aligned} \tag{2}$$

In an identical manner the perpendicalur volocity for the second wheel can be determined,

$$\begin{aligned}\mathbf{v}_C &= \mathbf{v}_2 + \omega_2 \times \mathbf{r}_{C/cm_2} \\ &= \begin{pmatrix}\dot{x}_2\\ \dot{y}_2\\ 0\end{pmatrix} + \begin{pmatrix}0\\ 0\\ \dot{\varphi}_2\end{pmatrix} \times \begin{pmatrix}c\cos\varphi_2\\ c\sin\varphi_2\\ 0\end{pmatrix} \\ &= \begin{pmatrix}\dot{x}_2 - c\dot{\varphi}_2\sin\varphi_2\\ \dot{y}_2 + c\dot{\varphi}_2\cos\varphi_2\\ 0\end{pmatrix}\end{aligned} \tag{3}$$

Therefore

$$\begin{aligned} v_{C_\perp} &= \mathbf{v}_C \begin{pmatrix}-\sin\varphi_2\\ \cos\varphi_2\\ 0\end{pmatrix} \\ &= -\dot{x}_2\sin\varphi_2 + \dot{y}_2\cos\varphi_2 + c\dot{\varphi}_2\end{aligned} \tag{4}$$

Therefore the the zero sideslip constraints are:

$$\begin{aligned} S_1 &= -\dot{x}_1\sin\varphi_1 + \dot{y}_1\cos\varphi_1 - a\dot{\varphi}_1 = 0 \\ S_2 &= -\dot{x}_2\sin\varphi_2 + \dot{y}_2\cos\varphi_2 + c\dot{\varphi}_2 = 0\end{aligned} \tag{5}$$

In addition also the holonomic constraints are imposed by the hinge B, which connects both bodies. Form the geometry of the problem we can see that these constraints are,

$$\begin{aligned} C_1 &= x_1 + b\cos(\phi_1) + d\cos(\phi_2) - x_2 \\ C_2 &= y_1 + b\sin(\phi_1) + d\sin(\phi_2) - y_2\end{aligned} \tag{6}$$

With these equations we can determine the EOM of this system

$$\begin{pmatrix} M_{ij} & C_{k,i} & S_{mi} \\ C_{k,j} & \mathbf{0} & \mathbf{0} \\ S_{mj} & \mathbf{0} & \mathbf{0}\end{pmatrix}\begin{pmatrix}\ddot{x}_j \\ \lambda_k \\ \lambda_m\end{pmatrix} = \begin{pmatrix} F_i \\ -C_{k,jl}\dot{x}_j\dot{x}_l \\ -S_{mj,l}\dot{x}_j\dot{x}_l\end{pmatrix} \tag{7}$$

1.2 Numeric Intergration

The motion of the mechanism can be computed by numerical integrating the result. For this the classical Runge-Kutta 4^{th} order method is used:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)\end{aligned} \tag{8}$$

1.3 Coordinate projection method

In this assignment two sets of constraints are used. The holonomic constraints constrain two of the six coordinates and two velocities. The nonholonomic constraints put two constraints on the velocities. However the approximate position $\bar{q}_{n+1}$ and velocity $\bar{\dot{q}}_{n+1}$ will generally not fulfill these constraints.

This will cause the mechanism to fly apart and the wheels to slip. Therefore the Gauss-Newton method is used to transform the solution back to the constrained surface.

$$\begin{aligned} q_{n+1} &= \bar{q}_{n+1} + \Delta q_{n+1} \\ \dot{q}_{n+1} &= \bar{\dot{q}}_{n+1} + \Delta \dot{q}_{n+1} \end{aligned} \tag{9}$$

where Δq_{n+1} is determined itteritevly with:

$$\begin{aligned} &e = -C \\ &\Delta q_{n+1} = \text{pinv}\left(C_{,q}\right) e \end{aligned} \tag{10}$$

The speeds fulfil a linear constraint, therefore $\Delta \dot{q}_{n+1}$ can be determined with one step. Note that we now need to take the holonomic and non holonomic constraints into account

$$\begin{aligned} \dot{e} &= -\begin{pmatrix} C_{,q}\dot{q} \\ S_{,q}\dot{q} \end{pmatrix} \\ \Delta \dot{q}_{n+1} &= \text{pinv}\begin{pmatrix} C_{,q} \\ S_{,q} \end{pmatrix} \dot{e} \end{aligned} \tag{11}$$

An error will be made of the same order as the order made in the numerical integration step.

1.4 First results

The algorithm described above is implemented in MATLAB. For verification we run a simple test with some nonzero initial conditions, so we can inspect whether the system is moving properly. The EzyRoller is given an intial velocity in the positive x direction, while the from wheel is given a negative steering angle, the results are shown in figure 2.

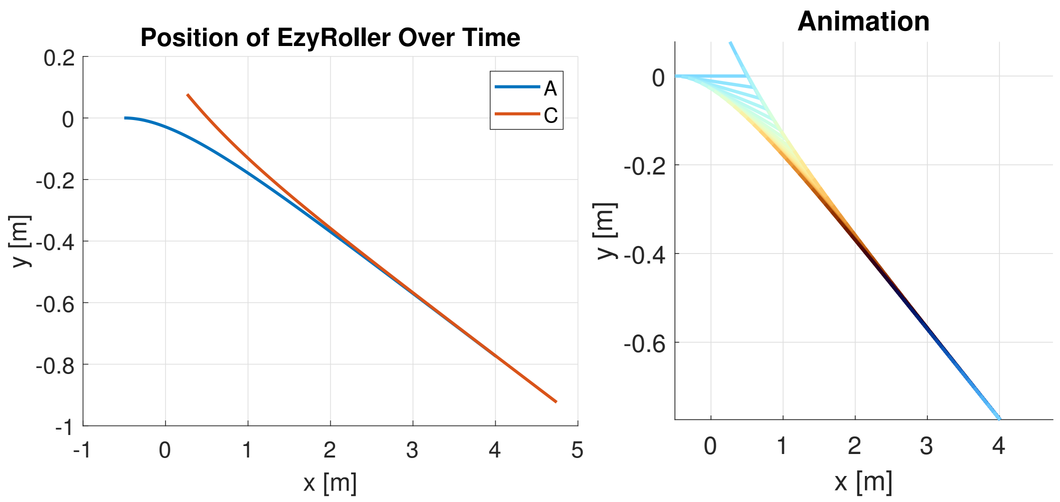

Figure 2: The bath followed by the EzyRoller

We can see that the path followed by the EzyRoller is as expected, thus the alogrithm is working.

1.5 Action reaction torque

An action reaction torque is added to the hinge such that it resembles an oscillating torque as applied by the rider on the steering assembly. The following torque function is used

$$M = M_0 \cos(\omega t) \tag{12}$$

Where $M_0 = 0.1$ [Nm], and $\omega = \pi$ [rad/s]. This torque can easily be added to the force vector, the path fro the first 100 s is shown in figure 3. We can see that initially the EzyRunner turns about

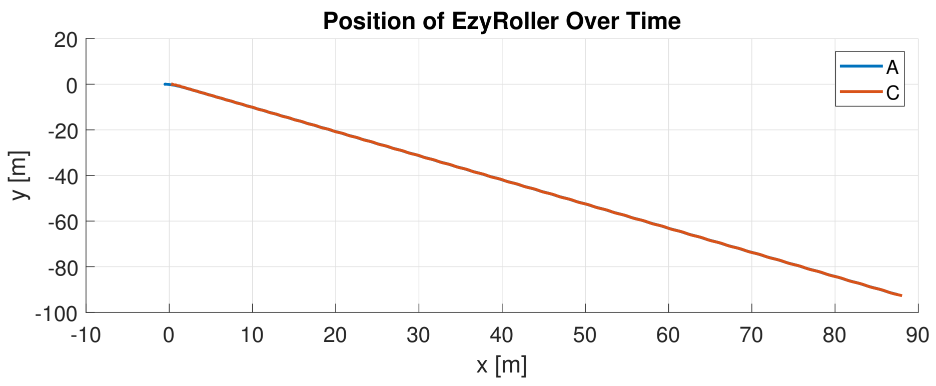

Figure 3: The path followed by the EzyRoller

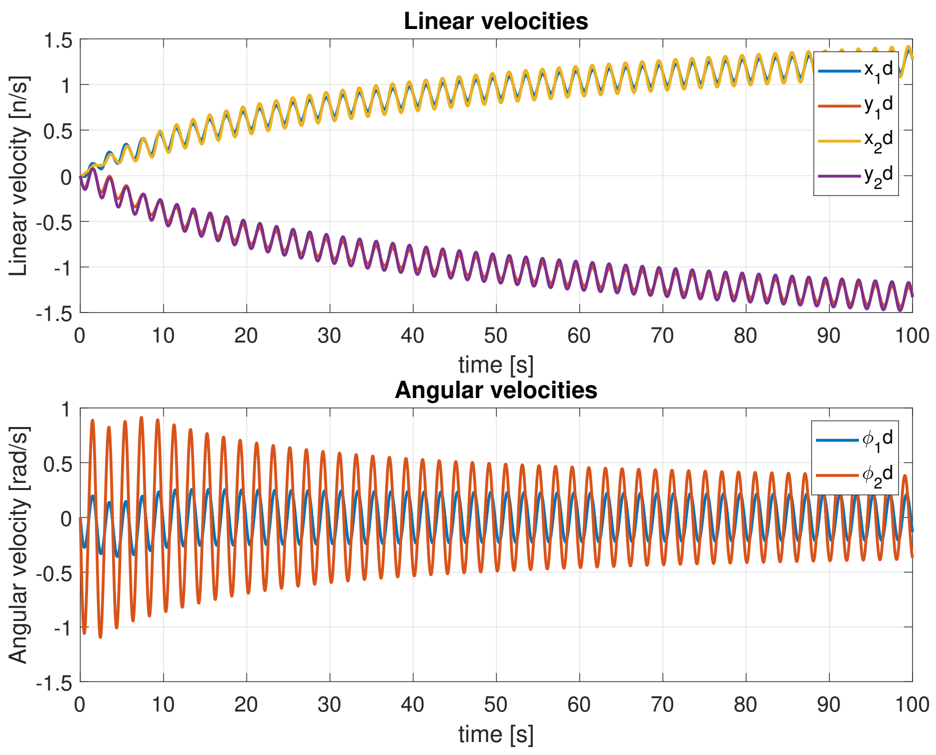

Figure 4: Caption

45 degrees in the negative direction, and then follows a fairly straight line. The linear and angular velocities of both bodies are shown in figure 4.

The velocities of both bodies are opposite since the second body is rotated by approximately 180 degrees. Finally the kinetic energy and work are determined with,

$$\begin{aligned} E_{kin} &= \frac{1}{2} v^T M v \\ W &= \int_{t_0}^{t_{end}} F ds \end{aligned} \tag{13}$$

Where the intergration is done numericly. They are plotted in figure 5.

We can see that the kinetic energy in the system steadily increases together with the work done. The are both approximately the same which is as expected due to lag of losses. A small error is made

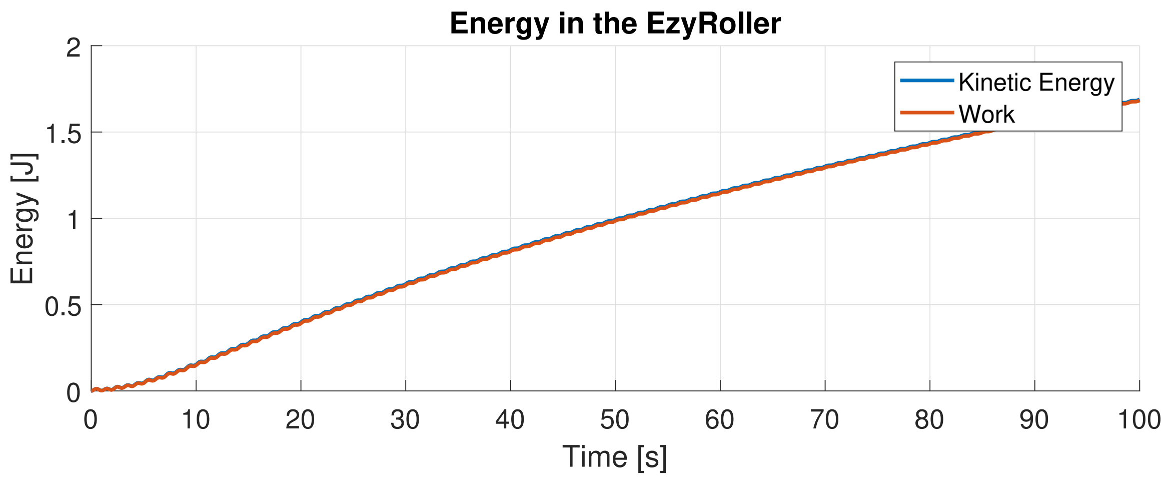

Figure 5: Kinetic energy in the EzyRoller with the work done by the torque

which causes the work sometimes to be slightly lower than the kinetic energy in the system. This due to numeric integration.

Appendix A

All code can be found in this section.

main.m

```
clear all
close all
clc
% Author: Taeke de Haan
% Date: 30-05-2018

%% init
% simulation
dt = 0.00001;               %[s]
t_end = 100;              %[s]
t = 0:dt:t_end;          %[s]
itterations = t_end/dt; %[-]

% pl0t
dim = [200, 200, 1000, 350];

% animation
fline = 10;
static = true;

% dim
paramDim.a = 0.5;   % [m]
paramDim.b = 0.5;   % [m]
paramDim.c = 0.125;   % [m]
paramDim.d = 0.125; % [m]
[a, b, c, d] = unfold_param_dim(paramDim);

% mass/ inirtia
paramMass.m1 = 1;     % not specified
paramMass.m2 = 0;   % [kg]
paramMass.I1 = 0.1;     % [kgm2]
paramMass.I2 = 0;     % [kgm2]
[m1, m2, I1, I2] = unfold_param_mass(paramMass);

M = [m1, m1, I1, m2, m2, I2];
M = diag(M);

% torque and force
M0 = 0.1;
omega = pi;
force = 0;   % [N]
torque = M0 * cos(omega * t);   % [Nm]

%% init
% q
x1 = 0;
y1 = 0;
phi1 = 0;
phi2 = pi;
```

```matlab
% body 2
x2 = x1 + a * cos(phi1) + d * cos(phi2);
y2 = y1 + a * sin(phi1) + d * sin(phi2);

q = [x1, y1, phi1, x2, y2, phi2];

% qd
x1d = 0;
y1d = 0;
phi1d = 0;
phi2d = 0;
x2d = 0; %xid +
y2d = 0;

qd = [x1d y1d phi1d, x2d, y2d, phi2d];

% pack
y = [q, qd];

% transform to possible space
y(1,:) = gauss_newton(y(1,:), paramDim);

for i= 1:itterations

    % constrain
    y(i,:) = gauss_newton(y(i,:), paramDim);
    [yd(i,:), lambda(i,:)] = compute_system(y(i,:), paramDim,
       paramMass, force, torque(i));

    % intergrate numericly
    y(i + 1,:) = int_rk4(y(i,:), dt, paramDim, paramMass, force,
       torque(i));

    % unpack
    q = y(i,1:4);

    % store loactions
    A(i,:) = [y(i,1) - a * cos(y(i,3)), y(i,2) - a * sin(y(i,3))];
    B1(i,:) = [y(i,1) + b * cos(y(i,3)), y(i,2) + b * sin(y(i,3))];
    B2(i,:) = [y(i,4) - d * cos(y(i,6)), y(i,5) - d * sin(y(i,6))];
    C(i,:) = [y(i,4) + c * cos(y(i,6)), y(i,5) + c * sin(y(i,6))];

end

i = i + 1;
y(i,:) = gauss_newton(y(i,:), paramDim);
[yd(i,:), lambda(i,:)] = compute_system(y(i,:), paramDim, paramMass,
    force, torque(i));


%% compute enrgy
for i = 1:(itterations + 1)
    % determine enrgy
    Ekin(i) = 1/2 * y(i,7:end) * M * y(i,7:end)';

    diff(i) = y(i,3) - y(i,6); % phi1 - phi2
    if i ~= 1
        W(i) = torque(i) * (diff(i) - diff(i-1));
    else
        W(i) = 0;
    end
end

W = cumsum(W);

%% animate
animate_bar(A, B1, B2 , C, dt, fline, static)
saveas(gca, 'fig/animate', 'jpg')
saveas(gca, 'fig/animate', 'epsc')


%% Path of A and B
figure('Position', dim)
hold on
plot(A(:,1), A(:,2))
plot(C(:,1), C(:,2))
legend('A', 'C')
xlabel('x [m]')
ylabel('y [m]')
title('Position of EzyRoller Over Time')
saveas(gca, 'fig/path', 'jpg')
saveas(gca, 'fig/path', 'epsc')

%% anguylar velocities of body 1 and 2
figure('Position', dim)
plot(t, y(:,[9, 12]))
legend('\phi_1d', '\phi_2d')

title('Angular velocities')
xlabel('time [s]')
ylabel('Angular velocity [rad/s]')

saveas(gca, 'fig/ang_vel', 'jpg')
saveas(gca, 'fig/ang_vel', 'epsc')


%% Linear velocities of body 1 and 2
figure('Position', dim)
plot(t, y(:,[7, 8, 10, 11]))
legend('x_1d', 'y_1d', 'x_2d', 'y_2d')

title('Linear velocities')
xlabel('time [s]')
ylabel('Linear velocity [n/s]')

saveas(gca, 'fig/lin_vel', 'jpg')
saveas(gca, 'fig/lin_vel', 'epsc')

%% Kin energy
figure('Position', dim)
hold on

plot(t, Ekin)
plot(t, W)

title('Energy in the EzyRoller')
xlabel('Time [s]')
ylabel('Energy [J]')

legend('Kinetic Energy', 'Work')

saveas(gca, 'fig/energy', 'jpg')
saveas(gca, 'fig/energy', 'epsc')
```

gen_eq.m.m

```
%% Generate equations
% Author: Taeke de Haan
% Date: 30-05-2018

clear all; close all; clc

%% Pendulum Constants (symbolic definition)
% length
syms a b c d   real

% mass
syms m1 m2 I1 I2   real

% other
syms g torque force           real

%% Generalized coordinates and their derivatives
syms x1 y1 phi1 x2 y2 phi2 real
syms x1d y1d phi1d x2d y2d phi2d real
syms x1dd y1dd phi1dd x2dd y2dd phi2dd real

q   = [x1 y1 phi1, x2, y2, phi2].';
qd  = [x1d y1d phi1d, x2d, y2d, phi2d].';
qdd  = [x1dd y1dd phi1dd, x2dd, y2dd, phi2dd].';

%% Mapping
% center of mass positions as function of the generalized
   coordinates

% body 1
% -

% body 2
% -

% holonomic constraints
C1 = x1 + b * cos(phi1) + d * cos(phi2) - x2;
C2 = y1 + b * sin(phi1) + d * sin(phi2) - y2;

% non-holonomic constraints
S1 = - x1d * sin(phi1) + y1d * cos(phi1) - a * phi1d; % tire A
S2 = - x2d * sin(phi2) + y2d * cos(phi2) + c * phi2d; % tire C
```

```
% mass
M = [m1, m1, I1, m2, m2, I2];
M = diag(M);

%% Differentiate constraints
% holonomic
C = [C1; C2];
Cq  = simplify(jacobian(C,q));
Cd = Cq * qd; % this is dC/dt=dC/dx*xd
C2 = simplify(jacobian(Cd,q) * qd);

% non holonomic
S = [S1; S2];
Sq  = simplify(jacobian(S,qd));
Sd = Sq * qd; % this is dC/dt=dC/dx*xd
S2 = simplify(jacobian(Sd,q) * qd);

%% EOM in matrix form
% forces
F = [0, 0, torque, 0, 0, -torque]';

% zero matrix
O = zeros(2,2);

LHS = [M, Cq', Sq'; Cq, O, O; Sq, O, O];

% export to latex
disp(latex_pretify(LHS));

% Compute answer
result = LHS \ [F; -C2; -S2];
qdd = simplify(result(1:6));
lambda = simplify(result(7:length(result)));

% export to latex
% disp(latex_pretify(LHS));
%disp(latex_pretify(Q_bar_text));

%% Save symbolic derivation to script file.
% Use Diary function, save the symbolicly derived functions to file.

% qdd
if exist('symb_qdd.m', 'file')
    ! del symb_qdd.m
end
diary symb_qdd.m
    disp('qdd = ['), disp(qdd), disp('];');
diary off

% lambda
if exist('symb_lambda.m', 'file')
    ! del symb_lambda.m
end
diary symb_lambda.m
    disp('lambda = ['), disp(lambda), disp('];');
diary off
```

```
% S
if exist('symb_S.m', 'file')
    ! del symb_S.m
end
diary symb_S.m
    disp('S = ['), disp(S), disp('];');
diary off

% Sq
if exist('symb_Sq.m', 'file')
    ! del symb_Sq.m
end
diary symb_Sq.m
    disp('Sq = ['), disp(Sq), disp('];');
diary off

% C
if exist('symb_C.m', 'file')
    ! del symb_C.m
end
diary symb_C.m
    disp('C = ['), disp(C), disp('];');
diary off

% Cq
if exist('symb_Cq.m', 'file')
    ! del symb_Cq.m
end
diary symb_Cq.m
    disp('Cq = ['), disp(Cq), disp('];');
diary off
```

animate_bar.m

```
function animate_bar(A, B1, B2, C, dt, fline, static)
    % Author: Taeke de Haan
    % Date: 30-05-2018

    P = size(A,1);

    p = figure('Position', [200,200,500,400]); %create figure
    grid on

    xlabel('x [m]')
    ylabel('y [m]')
    title('Animation')
    axis([min(A(:,1)) max(C(:,1)), min(A(:,2)) max(C(:,2))])
    axis square

    plotStep = round((1/dt) / fline);

    for i = 1 : plotStep : size(A,1)
        if ishandle(p)
            figure(p)
            hold on
            if static
                color = [1/2 * sin(i * 2 * pi / P) + 1/2, 1/2 * sin(
                   i * 2 * pi / P + pi/4) + 1/2, 1/2 * sin(i * 2 *
                   pi / P + pi/2) + 1/2];
            else
                color = [1, 0.2, 0.2];
            end

            if i ~= 1 && ~static
                 delete(l_current)
            end
            l_current(1) = plot([A(i, 1), B1(i, 1)], [A(i, 2), B1(i,
                2)], 'Color',color);
            l_current(2) = plot([B2(i, 1), C(i, 1)], [B2(i, 2), C(i,
                2)], 'Color',color);

            tic
            drawnow

            %wait some time, so the gait is played back at an
               accurate speed.
            while true
                if ~(toc < dt * plotStep)
                    break
                end
            end
        else %If figure p is closed, stop animation.
            break
        end
    end
end
```

compute_system.m

```
function [yd, lambda] = compute_system(y, paramDim, paramMass, force
   , torque)
    % Author: Taeke de Haan
    % Date: 30-05-2018
% unpack
qd = y(7:end);

[a, b, c, d] = unfold_param_dim(paramDim);
[m1, m2, I1, I2] = unfold_param_mass(paramMass);

%unpack y
[x1, y1, phi1, x2, y2, phi2, x1d, y1d, phi1d, x2d, y2d phi2d] =
   unfold(y);

symb_qdd;
symb_lambda;

yd = [qd, qdd'];
%ydot = [phi1d; phi2d; qdd];
end
```

gauss_newton.m

```
function y = gauss_newton(y, param)
% Author: Taeke de Haan
% Date: 30-05-2018
q = y(1:6);
qd = y(7:12);

iterate = 0;      % itterations
tol = 1e-12;     % tolrance
maxiterate = 10;  % maximum itterations

% unfold parameters
[a, b, c, d] = unfold_param_dim(param);

while true
    % unfold
    [x1, y1, phi1, x2, y2, phi2, x1d, y1d, phi1d, x2d, y2d phi2d] =
       unfold([q, qd]);

    symb_C; % C = C(q_n1 )
    symb_Cq;

    % Cq' * (Cq * Cq')^(-1) ?
    e = -C;                    % error
    delta = pinv(Cq) * e;    % Dq_n1 = -C,q^T(C,q*C,q^T)*C
    q = q + delta';          % q_n1 = q_n1 + Dq_n1

        iterate = iterate +1;

    if( max(abs(C)) < tol || iterate > maxiterate)
        break;
    end
end

[x1, y1, phi1, x2, y2, phi2, x1d, y1d, phi1d, x2d, y2d phi2d] =
   unfold([q, qd]);
symb_Cq;
symb_Sq;  % C = C(q_n1 )

ed = - ([Cq *  qd' ; Sq * qd']);        % 1 Cd = C,q*C*qd_n1
deltad = pinv([Cq; Sq]) * ed;      % 2 Dqd_n1 = -C,q^T(C,q*C,q^T)*Cd
qd = qd + deltad';                % 3 set qd_n1 = qd_n1 + Dqd_n1

y = [q, qd];
end
```

int_rk4.m

```
function y = int_rk4(y, dt, paramDim, paramMass, force, torque)
% Author: Taeke de Haan
% Date: 30-05-2018
%rk4 intergration
%   [y] = int_rk4(y, dt, t)

k1 = compute_system(y, paramDim, paramMass, force, torque);
```

```
k2 = compute_system(y + 1/2 * dt * k1, paramDim, paramMass, force,
    torque);
k3 = compute_system(y + 1/2 * dt * k2, paramDim, paramMass, force,
    torque);
k4 = compute_system(y + dt * k3, paramDim, paramMass, force, torque)
    ;
y = y + 1/6 * dt * (k1 + 2 * k2 + 2 * k3 + k4);
end
```

latex_pretify.m

```
function string = latex_prefify(eq)
%Pretify equagtions for LateX
%   Author: Taeke de Haan
%   Date: 27-03-2017
string = latex(eq);
string = strrep(string,'alphadd','\ddot{\alpha}');
string = strrep(string,'alphad','\dot{\alpha}');

string = strrep(string,'betadd','\ddot{beta}');
string = strrep(string,'betad','\dot{beta}');
string = strrep(string,'beta','\beta');

string = strrep(string,'gamma','\gamma');
end
```

RPM2rads.m

```
function [rads] = RPM2rads(RPM)
    %% Converts for RPM to rad/s
    % Author: Taeke de Haan (4316568)
    % Date: 25-02-2018
    rads = RPM/60*2*pi; %[rad/s]
end
```

unfold.m

```
function [x1, y1, phi1, x2, y2, phi2, x1d, y1d, phi1d, x2d, y2d,
    phi2d] = unfold(y)
    % Author: Taeke de Haan
    % Date: 30-05-2018
    x1      = y(1);
    y1      = y(2);
    phi1    = y(3);
    x2      = y(4);
    y2      = y(5);
    phi2    = y(6);

    x1d     = y(7);
    y1d     = y(8);
    phi1d   = y(9);
    x2d     = y(10);
    y2d     = y(11);
    phi2d   = y(12);
end
```

unfold_param_dim.m

```
function [a, b, c, d] = unfold_param_dim(paramDim)
% Author: Taeke de Haan
% Date: 30-05-2018
a = paramDim.a;
b = paramDim.b;
c = paramDim.c;
d= paramDim.d;

end
```

unfold_param_J.m

```
function [J2, J3, J4, J5, J6] = unfold_param_J(paramJ)
    % Author: Taeke de Haan
    % Date: 30-05-2018
    J2 = paramJ.J2;
    J3 = paramJ.J3;
    J4 = paramJ.J4;
    J5 = paramJ.J5;
    J6 = paramJ.J6;
end
```

unfold_param_mass.m

```
function [m1, m2, I1, I2] = unfold_param_mass(paramMass)
% Author: Taeke de Haan
% Date: 30-05-2018
m1 = paramMass.m1;
m2 = paramMass.m2;
I1 = paramMass.I1;
I2 = paramMass.I2;

end
```