Table of Contents


```
function dynamicsModel(parameter)

mkdir('generated_codes/matlab');
mkdir('generated_codes/cpp');
mkdir('utils/casadi');

generateLegDynamics(parameter);
loadDynamics(parameter);

end

function loadDynamics(parameter)

addpath('./utils')
addpath('./generated_codes/matlab')

% Require CasADi for autodiff
addpath('./utils/casadi')
import casadi.*

x0 = SX.sym('x0', [parameter.n, 1]); % First finite element
x1 = SX.sym('x1', [parameter.n, 1]); % Second finite element
u = SX.sym('u', [parameter.m, 1]); % Input
feet_location = SX.sym('feet_location', [12, 1]); % Foot vector
dt = SX.sym('dt'); % Finite element length
mu = SX.sym('mu'); % Friction coefficient
lambda = SX.sym('lambda', [parameter.n+16, 1]); % Constraint
 multipliers

[M, h, J_u, q_dot] = legDynamics(x1, feet_location); % Backward Euler
EOM = [(x1(1:parameter.n/2)-x0(1:parameter.n/2))-dt*q_dot;
    M*(x1(parameter.n/2+1:end)-x0(parameter.n/2+1:end))+dt*(h-
J_u*u)]; % Equation of motion

tmp = [1, 0, -mu;
    -1, 0, -mu;
    0, 1, -mu;
    0, -1, -mu];
friction_cone = kron(diag([1;1;1;1]), tmp);
friction = friction_cone*u; % Friction pyramid

p = [dt; mu; feet_location]; % Optimization parameter
w = [x0; u; x1]; % Decision variable
```

```
g = [EOM; friction]; % Constraints including dynamics and friction
eval_g = Function(['eval_g_', convertStringsToChars(parameter.name)],
{w, p},{g},{'w','p'},{'g'}); % CasADi function

jac_g = jacobian(g, w); % Constraint jacobian
[hess_g, ~] = hessian(g'*lambda, w);
hess_g = tril(hess_g); % Constraint hessian

eval_jac_g = Function(['eval_jac_g_',
 convertStringsToChars(parameter.name)], {w, p}, {jac_g}, {'w', 'p'},
 {'jac_g'});
eval_hess_g = Function(['eval_hess_g_',
 convertStringsToChars(parameter.name)], {w, lambda, p}, {hess_g},
 {'w', 'lambda', 'p'}, {'hess_g'});

% Generate cpp codes
opts = struct('cpp', true, 'with_header', true);
eval_g.generate(['eval_g_',
 convertStringsToChars(parameter.name), '.cpp'], opts);
eval_jac_g.generate(['eval_jac_g_',
 convertStringsToChars(parameter.name), '.cpp'], opts);
eval_hess_g.generate(['eval_hess_g_',
 convertStringsToChars(parameter.name), '.cpp'], opts);

movefile('eval*', 'generated_codes/cpp');
end

function generateLegDynamics(parameter)

addpath('./utils')

syms p theta p_dot omega [3, 1] real % Body positions, RPY angles,
 linear velocities in world frame and angular velocities in body frame
syms feet_location [12, 1] real % Foot to body vectors in world frame

q = [p; theta];
```

Kinematics

```
R_wb = [cos(theta(3)), -sin(theta(3)), 0;
    sin(theta(3)), cos(theta(3)), 0;
    0, 0, 1]*...
    [cos(theta(2)), 0, sin(theta(2));
    0, 1, 0;
    -sin(theta(2)), 0, cos(theta(2))]*...
    [1, 0, 0;
    0, cos(theta(1)), -sin(theta(1));
    0, sin(theta(1)), cos(theta(1))]; % Rotation matrix

J_wb__b=jacobian(reshape(R_wb, 9, 1), theta);
J_wb__b=[skew2angvel(R_wb'*reshape(J_wb__b(:, 1), 3, 3)), ...
    skew2angvel(R_wb'*reshape(J_wb__b(:, 2), 3, 3)), ...
    skew2angvel(R_wb'*reshape(J_wb__b(:, 3), 3, 3))]; % Jacobian
```

```
theta_dot=J_wb__b\omega; % Euler angle rates
V_wb__b = [R_wb'*p_dot; omega]; % Body twist
```

Dynamics

```
M_b = blkdiag(diag(repmat(parameter.physics.mass_body, 3, 1)),
 parameter.physics.inertia_body); % Body inertia

T = V_wb__b'*M_b*V_wb__b./2; % Kinematic energy
V =
 (parameter.physics.mass_body)*parameter.physics.gravitational_constant*p(3); %
 Potential energy

L = T-V; % Lagrangian
L = elementwiseSimplify(L);

q_dot = [p_dot; theta_dot];
velocities = [p_dot; omega];

% Lagrange's equations
tmp = jacobian(L, velocities)*blkdiag(eye(3), J_wb__b);
tmp = elementwiseSimplify(tmp);

M = jacobian(tmp, velocities); % Inertia matrix

tmp2 = jacobian(reshape(J_wb__b, 9, 1), theta);
tmp2 = [reshape(tmp2(:, 1), 3, 3)*theta_dot, ...
    reshape(tmp2(:, 2), 3, 3)*theta_dot, ...
    reshape(tmp2(:, 3), 3, 3)*theta_dot];
tmp2 = blkdiag(zeros(3, 3), tmp2);
tmp2 = elementwiseSimplify(tmp2);

h = jacobian(tmp, q)*q_dot-...
    (jacobian(L, q))'-...
    (jacobian(L, velocities)*tmp2)'; % Coriolis and potential energy
 terms

% Input jacobian
J_feet = [];

for i = 1:4
    g_lb = [R_wb, feet_location(1+3*(i-1):3*i); 0, 0, 0, 1];
    V_lb__s = tform2adjoint(g_lb)*V_wb__b;
    J_lb__s = jacobian(V_lb__s, velocities)*blkdiag(eye(3), J_wb__b);

    J_feet = [J_feet, J_lb__s'*[eye(3); zeros(3)]];
end
```

Export

```
x = [q; velocities]; % State space

M = elementwiseSimplify(M);
```

```
h = elementwiseSimplify(h);
J_feet = elementwiseSimplify(J_feet);
q_dot = elementwiseSimplify(q_dot);

matlabFunction(M, h, J_feet, q_dot, 'File', 'generated_codes/matlab/
legDynamics', 'Vars', {x, feet_location});

end
```

```
Warning: Directory already exists.
Warning: Directory already exists.
Warning: Directory already exists.
```

```
Not enough input arguments.

Error in dynamicsModel (line 7)
generateLegDynamics(parameter);
```

Published with MATLAB® R2021a