Dynamic model of Otbot

This live script develops the dynamic model of Otbot and the solution to its forward and inverse dynamics problems.

Table of Contents

[1. Initializations](#MW_H_75165D1E)   
[2. Computation of the mass matrix](#MW_H_C1624CC5)   
[3. Computation of the Coriolis matrix](#MW_H_3E18453E)   
[4. Generalized force of actuation](#MW_H_C853F4AF)   
[5. Acceleration constraint](#MW_H_C1819C81)   
[6. Final model in explicit first-order form](#MW_H_07BCD941)   
[7. Solution to the forward dynamics problem](#MW_H_44475EE0)   
[8. Solution to the inverse dynamics problem](#MW_H_8980F79A)   
[9. Save wokspace and main matrices to file](#MW_H_8C496859)   
[10. Print ellapsed time](#MW_H_A30E5B9A)   
[Appendix: Function kin\_en\_trans](#MW_H_FBEF0ADE)

Recall that the equation of motion of a robot takes the form



where:

*  is the configuration vector of the robot (of size  in our case)
*  is the vector of motor torques (the right and left wheel torques and the platform torque).
*  is the mass matrix of the unconstrained system (positive-definite of size 
* is the generalized Coriolis and centrifugal force matrix
*  is the generalized gravity force
*  is the constraint Jacobian of the robot
*  is a vector of Lagrange mulitpliers.
*  is the generalized force of actuation, which can be written in the form 
*  is the generalised force modelling all friction forces in the system

Note that, since the Otbot will move on flat terrain, . Friction forces will also be neglected for the moment, so  will be zero initially. The constraint Jacobian  was already obtained in kinematic\_model.mlx and we will simply load it from an appropriate mat file.

Our task thus boils down to obtaining , , and  initially. After that, we will assemble the full dynamic model using the acceleration-level constraint (the time derivative of ).

Hereafter, when we say "chassis" we mean the robot chassis **including** the wheels.

# 1. Initializations

% Clear all variables, close all figures, and clear the command window

clearvars

close all

clc

% Start stopwatch

tic;

% Display the matrices with rectangular brackets

sympref('MatrixWithSquareBrackets',true);

% Turn off abbreviated output format to avoid Matlab's own

% substitution of long expressions)

% sympref('AbbreviateOutput',false);

% Symbolic variables to be used (see the figures and explanations below)

syms x y % Absolute coords of the pivot joint

syms x\_dot y\_dot % Absolute velocity components of the pivot joint

syms alpha % Absolute angle of the platform

syms alpha\_dot % Absolute angular velocity of the platform

syms varphi\_r % Angle of right wheel relative to chassis

syms varphi\_l % Angle of left wheel relative to chassis

syms varphi\_p % Pivot joint angle (platform wrt chassis)

syms varphi\_dot\_l % Angular velocity of left wheel

syms varphi\_dot\_r % Angular velocity of right wheel

syms varphi\_dot\_p % Angular velocity of the platform relative to the chassis

syms l\_1 % Pivot offset relative to the wheels axis

syms l\_2 % One half of the wheels separation

syms m\_c % Mass of the chassis (including wheels)

syms m\_p % Mass of the platform

syms x\_F y\_F % Coords of F (c.o.m. of the platform) in platform frame

syms x\_B y\_B % Coords of B (c.o.m. of the chassis) in chassis frame

syms I\_p % Vertical moment of inertia of the platform at its c.o.m.

syms I\_a % Axial moment of inertia of one wheel

syms I\_c % Vertical moment of inertia of the chassis at its c.o.m.

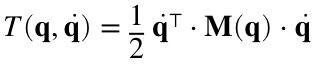
# 

# 2. Computation of the mass matrix

Recall that the robot configuration is given by



To find the mass matrix  we first write the kinetic energy  of the robot as a function of  and  and then express it as:



We have to compute the translational and rotational kinetic energies of all bodies in the Otbot, and add them all to obtain .

% Chassis angle theta, and its derivative, in terms of qdot

theta = alpha - varphi\_p;

theta\_dot = alpha\_dot - varphi\_dot\_p;

% Translational kinetic energies

% Of the chassis

T\_tra\_c = kin\_en\_trans(x\_B, y\_B, m\_c, x\_dot, y\_dot, theta, theta\_dot);

% Of the platform

T\_tra\_p = kin\_en\_trans(x\_F, y\_F, m\_p, x\_dot, y\_dot, alpha, alpha\_dot);

% Rotational kinetic energies

% Whole chassis including wheels

T\_rot\_c = 1/2 \* I\_c \* (theta\_dot)^2;

% Right wheel when turning about its axis alone

T\_rot\_r = 1/2 \* I\_a \* (varphi\_dot\_r)^2;

% Left wheel when turing about its axis alone

T\_rot\_l = 1/2 \* I\_a \* (varphi\_dot\_l)^2;

% Platform

T\_rot\_p = 1/2 \* I\_p \* (alpha\_dot)^2;

% Total kinetic energy

T = T\_tra\_c + T\_tra\_p + ...

T\_rot\_c + T\_rot\_r + T\_rot\_l + T\_rot\_p;

% Display the results

display(T\_tra\_c); display(T\_tra\_p);

display(T\_rot\_c); display(T\_rot\_r); display(T\_rot\_l); display(T\_rot\_p);

display(T);

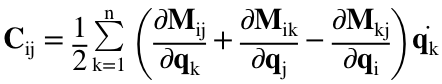
Since  is a quadratic form, the Hessian of  gives the desired mass matrix:

Mmat = hessian(T, ...

[x\_dot y\_dot alpha\_dot varphi\_dot\_r varphi\_dot\_l varphi\_dot\_p])

# 3. Computation of the Coriolis matrix

Recall that the  element of  is given by the following formula (see Murray, Sastry, Lee)

,

which in Matlab can be implemented as follows:

qvec = [x y alpha varphi\_r varphi\_l varphi\_p].';

qdotvec = [x\_dot y\_dot alpha\_dot varphi\_dot\_r varphi\_dot\_l varphi\_dot\_p].';

nq = length(qvec);

Cmat = sym(zeros(nq,nq));

for i=1:nq

for j=1:nq

for k=1:nq

Cmat(i,j) = Cmat(i,j) + qdotvec(k) \* 0.5 \* ( ...

diff(Mmat(i,j),qvec(k)) + ...

diff(Mmat(i,k),qvec(j)) - ...

diff(Mmat(k,j),qvec(i)) );

end

end

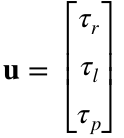
end

Cmat = simplify(Cmat);

display(Cmat)

# 4. Generalized force of actuation

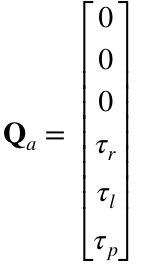
Our vector  of motor torques is defined as



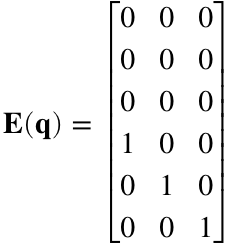
syms tau\_r tau\_l tau\_p

uvec = [tau\_r; tau\_l; tau\_p];

Each of the torques in  acts directly on a  coordinate and, therefore, the generalized force of actuation is given by



To rewrite this force in the form  we only have to define



Ematrix = [zeros(3);eye(3)];

This completes our derivation of all matrices involved in the equation of motion.

# 5. Acceleration constraint

We now wish to obtain the robot model in the form  This requires writing  and  as a function of , , and  Recall that the Euler-Lagrange equation is

 (we omit the dependencies on  and  for simplicity)

Since this equation is a system of 6 equations in 9 unknowns (6 coordinates in  and 3 in ) we need 3 additional equations to determine and  for a given . These equations can be obtained by taking the time derivative of the kinematric constraint

,

which gives

,

or, equivalently,



The latter equation is called the acceleration constraint of the robot.

Recall that  was obtained in kinematic\_model.mlx. Therefore, we only need to calculate its time derivative now:

% We first import J\_of\_q (obtained by an earlier run of kinematic\_model.mlx)

load("constraint\_jacobian.mat")

% Substitute its variables by functions of t and take the time derivative

syms f1(t) f2(t)

J\_of\_t = subs(J\_of\_q,[alpha varphi\_p],[f1(t) f2(t)]);

dJdt = diff(J\_of\_t,t);

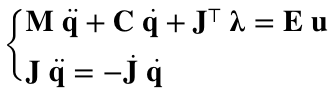
% Substitute d/dt of f1 and f2 by the original variables using dot notation

Jdot = subs(dJdt,[f1(t) f2(t) diff(f1,t) diff(f2,t)], ...

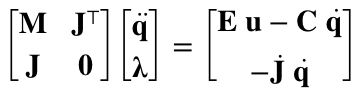
[alpha varphi\_p alpha\_dot varphi\_dot\_p])

# 6. Final model in explicit first-order form

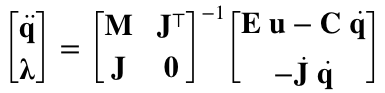
The equations



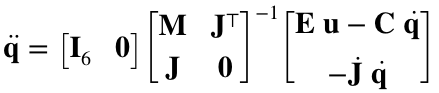
can be written as a linear system with the form



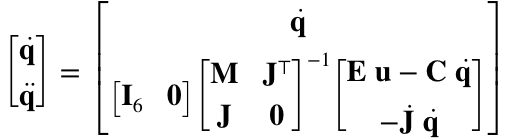
and since  is positive-definite and  is full row rank, the matrix on the left-hand side can be inverted to write

.

Therefore

.

Finally, by considering the trivial equation  in conjunction with the earlier equation we arrive at

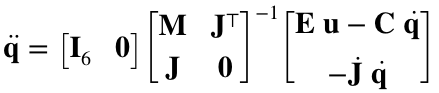
,

which gives the robot model in the usual control form

.

# 7. Solution to the forward dynamics problem

The forward dynamics problem consists in finding the acceleration  that corresponds to a given . Such a  is given by the earlier expression

.

# 8. Solution to the inverse dynamics problem

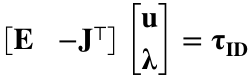
The inverse dynamics problem consists in finding the torques  that produce a desired . These torques are obtained by solving



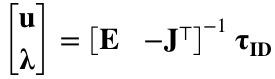
for  and  (6 equations and 6 unknowns). For this we define  and rewrite the previous equation as

,

or, equivalently, as

.

It is easy to see that  is a  full rank matrix irrespectively of . This follows directly from the expressions of  and  in the Otbot. Thus, we can write

,

so that



provides the desired value for .

# 9. Save wokspace and main matrices to file

save("dynamic\_model\_workspace")

save('Mmatrix.mat',"Mmat")

save('Cmatrix.mat',"Cmat")

# 10. Print ellapsed time

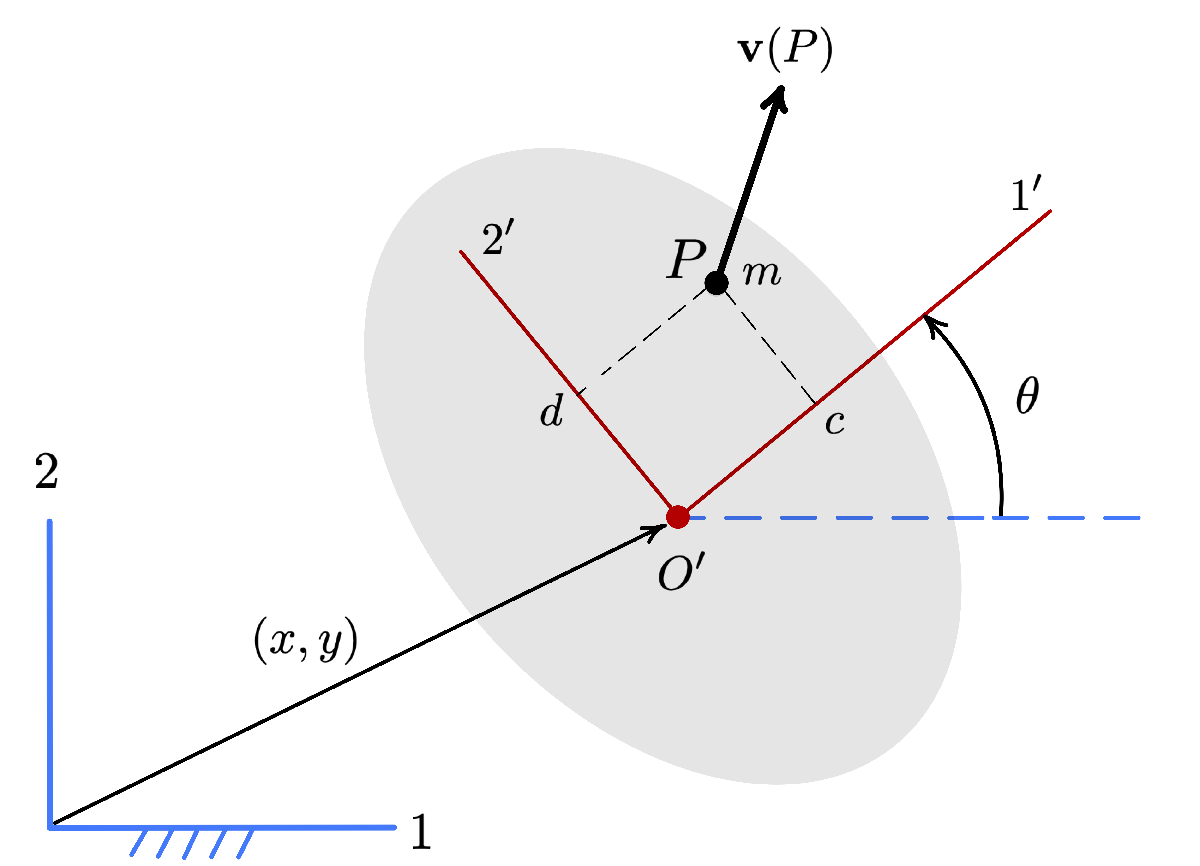
toc

# Appendix: Function kin\_en\_trans

This function computes the translational kinetic energy of a point mass located at P on the grey body, in terms of the velocity coordinates , , and  of the body.

The following notation is used:

*  = Local coordinates of P in the body-fixed frame {1',2'}
* = Absolute coordinates of the origin of the body-fixed frame
*  = absolute angle of the body
*  = mass of P



This function can be used to compute the translational kinetic energy of any body in planar motion. Just let P be the c.o.m. of the body, and m be the total mass of the body.

function T = kin\_en\_trans(c,d,m,xdot,ydot,theta,thetadot)

cth = cos(theta);

sth = sin(theta);

A = [1, 0, -cth \* d - sth \* c;

0, 1, cth \* c - sth \* d;

0, 0, 1 ];

Mp = [m, 0, 0; 0, m, 0; 0, 0, 0];

M = simplify(transpose(A) \* Mp \* A);

w = [xdot;ydot;thetadot];

T = expand(1/2 \* transpose(w) \* M \* w,'ArithmeticOnly',true);

end