Dynamic model in task space coordinates

This live script obtains a closed formula for the inverse dynamics of the Otbot by using a tricky elimination of the Lagrange multipliers. It then obtains the equation of motion in task-space coordinates, which can be used to design a computed-torque controller for trajectory tracking.

Table of Contents

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);

% 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
syms varphi_l       % Angle of left wheel
syms varphi_p       % Pivot joint angle
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 pivot motor
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 the wheels
syms m_p            % Mass of the platform
syms x_B y_B        % Coords of the c.o.m. of the chassis in chassis frame
syms x_F y_F        % Coords of the c.o.m. of the platform in platform frame
```

```
syms I_c              % Vertical moment of inertia of the chassis at B
syms I_p              % Vertical moment of inertia of the platform at F
syms I_a              % Axial moment of inertia of one wheel

% Load matrices
load('MIIK.mat')
load('MFIK.mat')
load('M.mat')
load('C.mat')
```

2 - Multiplier-free inverse dynamics

Our goal is to obtain an equation of motion that describes the time evolution of the $\mathbf{p}$ coordinates alone, and at the same time does not contain the annoying Lagrange multipliers $\boldsymbol{\lambda}$. In this way we will obtain a one-to-one relationship between platform accelerations and the torques $\mathbf{u}$ applied to the robot. This relationship can later be used to design a computed-torque control law.

Consider the following parametrizations of the feasible $\dot{\mathbf{q}}$

$$\dot{\mathbf{q}} = \boldsymbol{\Lambda} \cdot \dot{\mathbf{p}},$$

$$\dot{\mathbf{q}} = \boldsymbol{\Delta} \cdot \dot{\boldsymbol{\varphi}},$$

where

$$\boldsymbol{\Lambda} = \begin{bmatrix} \mathbf{I}_3 \\ \mathbf{M}_{\text{IIK}} \end{bmatrix}$$

$$\boldsymbol{\Delta} = \begin{bmatrix} \mathbf{M}_{\text{FIK}} \\ \mathbf{I}_3 \end{bmatrix}$$

```
Lambda = [eye(3); MIIK]
Delta  = [MFIK; eye(3)]
```

For later use, also consider the time derivative of the first parameterization

$$\ddot{\mathbf{q}} = \boldsymbol{\Lambda}\ddot{\mathbf{p}} + \dot{\boldsymbol{\Lambda}}\dot{\mathbf{p}}.$$

and let us compute $\dot{\boldsymbol{\Lambda}}$ with Matlab:

```
% Substitute its variables by functions of t
syms f1(t) f2(t)
Lambda2 = subs(Lambda,[alpha varphi_p],[f1(t) f2(t)]);
% Take the time derivative
dLambdadt = diff(Lambda2,t);

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

```
Lambda_dot = subs(dLambdadt,...
    [f1(t) f2(t) diff(f1,t) diff(f2,t)], ...
    [alpha varphi_p alpha_dot varphi_dot_p])

clearvars f1(t) f2(t)
```

Recall that the equation of motion of the Otbot takes the form

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{J}^{\top}\boldsymbol{\lambda} = \mathbf{E}\,\mathbf{u}$$

Let us multiply this equation by $\boldsymbol{\Delta}^{\top}$:

$$\boldsymbol{\Delta}^{\top}\mathbf{M}\ddot{\mathbf{q}} + \boldsymbol{\Delta}^{\top}\mathbf{C}\dot{\mathbf{q}} + \boldsymbol{\Delta}^{\top}\mathbf{J}^{\top}\boldsymbol{\lambda} = \boldsymbol{\Delta}^{\top}\mathbf{E}\mathbf{u}$$

Note that $\boldsymbol{\Delta}^{\top}\mathbf{J}^{\top}\boldsymbol{\lambda} = \mathbf{0}$, as the columns of $\boldsymbol{\Delta}$ form a basis of the kernel of $\mathbf{J}$, and $\mathbf{J}^{\top}\boldsymbol{\lambda}$ is a vector orthogonal to this kernel. Also note that

$$\boldsymbol{\Delta}^{\top}\mathbf{E} = \begin{bmatrix}\mathbf{M}_{\mathrm{FIK}}^{\top} & \mathbf{I}_3\end{bmatrix}\begin{bmatrix}\mathbf{0} \\ \mathbf{I}_3\end{bmatrix} = \mathbf{I}_3,$$

so the equation of motion reduces to

$$\boldsymbol{\Delta}^{\top}\mathbf{M}\ddot{\mathbf{q}} + \boldsymbol{\Delta}^{\top}\mathbf{C}\dot{\mathbf{q}} = \mathbf{u}$$

This formula gives us an explicit solution for the inverse dynamics of the Otbot.

3 - Equation of motion in task-space coordinates

If we now substitute $\dot{\mathbf{q}} = \boldsymbol{\Lambda}\dot{\mathbf{p}}$ and $\ddot{\mathbf{q}} = \boldsymbol{\Lambda}\ddot{\mathbf{p}} + \dot{\boldsymbol{\Lambda}}\dot{\mathbf{p}}$ into the earlier equation we obtain:

$$\boldsymbol{\Delta}^{\top}\mathbf{M}\boldsymbol{\Lambda}\ddot{\mathbf{p}} + \boldsymbol{\Delta}^{\top}\left(\mathbf{M}\dot{\boldsymbol{\Lambda}} + \mathbf{C}\boldsymbol{\Lambda}\right)\dot{\mathbf{p}} = \mathbf{u}$$

Let us compute

$$\bar{\mathbf{M}} = \boldsymbol{\Delta}^{\top}\mathbf{M}\boldsymbol{\Lambda}$$

$$\bar{\mathbf{C}} = \boldsymbol{\Delta}^{\top}\left(\mathbf{M}\dot{\boldsymbol{\Lambda}} + \mathbf{C}\boldsymbol{\Lambda}\right)$$

```
M_bar = simplify(Delta.' * M * Lambda)
C_bar = simplify(Delta.' * (M * Lambda_dot + C * Lambda) )
```

We call these matrices the task-space mass matrix, and the task-space Coriolis matrix respectively. Using them, the equation of motion can be written in the compact form

$$\bar{\mathbf{M}}\,\ddot{\mathbf{p}} + \bar{\mathbf{C}}\,\dot{\mathbf{p}} = \mathbf{u}$$

This is called the equation of motion in task-space coordinates, and can be used to design a computed-torque controller to track arbitrary trajectories in task space.

4 - Save main matrices

```
save('Lambda.mat',"Lambda")
save('Lambda_dot.mat',"Lambda_dot")
save('Delta.mat',"Lambda")
save('M_bar.mat',"M_bar")
save('C_bar.mat',"C_bar")
```

5 - Print ellapsed time

```
toc
```