Report Assignment IV - Dynamics

Nils Becker

Fundamentals of Robotics - FS 2021

-

December 9^{th} 2021

1 Euler-Lagrange

$$\tau = M(q)\ddot{q} + c(q,\dot{q})\dot{q} + g(q) \tag{1}$$

1.1 Inertia Matrix M(q)

$$M(q) = \sum_{i=1}^{6} m_i J_{v_i}^T J_{v_i} + (J_{w_i}^T R_i) I_i (R_i^T J_{w_i}) \tag{2}$$

Note that $M(q)$ will be of shape 6x6, because the robot has 6 DoF.

1.1.1 Finding CoM's

We find:

$$cm_1 = R_z(q_1) lc_1 \tag{3}$$

$$cm_2 = R_z(q_1) R_y(q_2) T(pos_2) lc_2 \tag{4}$$

$$cm_3 = R_z(q_1) R_y(q_2) R_y(q_3) T(pos_3) lc_3 \tag{5}$$

$$cm_4 = R_z(q_1) R_y(q_2) R_y(q_3) R_z(q_4) T(pos_4) lc_4 \tag{6}$$

$$cm_5 = R_z(q_1) R_y(q_2) R_y(q_3) R_z(q_4) Rx(q_5) T(pos_6) lc_5 \tag{7}$$

$$cm_6 = R_z(q_1) R_y(q_2) R_y(q_3) R_z(q_4) R_x(q_5) R_z(q_6) T(pos_6) lc_6 \tag{8}$$

pos_i represents the position of joint i. Note that $T(pos_1) = (0,0,0)^T$, thus it can be ignored in (3).
lc_i represents the vector from joint $i-1$ to the CoM of link i
l_i represents the vector from joint $i-1$ to joint i

1.1.2 Finding the Jacobians

The Jacobian for linear velocity is given by:

$$J_{v_i} = \begin{pmatrix} \frac{\partial x_i}{\partial q_1} & \frac{\partial x_i}{\partial q_2} & \frac{\partial x_i}{\partial q_3} & \frac{\partial x_i}{\partial q_4} & \frac{\partial x_i}{\partial q_5} & \frac{\partial x_i}{\partial q_6} \\ \frac{\partial y_i}{\partial q_1} & \frac{\partial y_i}{\partial q_2} & \frac{\partial y_i}{\partial q_3} & \frac{\partial y_i}{\partial q_4} & \frac{\partial y_i}{\partial q_4} & \frac{\partial y_i}{\partial q_6} \\ \frac{\partial z_i}{\partial q_1} & \frac{\partial z_i}{\partial q_2} & \frac{\partial z_i}{\partial q_3} & \frac{\partial z_i}{\partial q_4} & \frac{\partial z_i}{\partial q_5} & \frac{\partial z_i}{\partial q_6} \end{pmatrix} \tag{9}$$

Note that:

$$cm_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{10}$$

The Jacobian for angular velocities is given by:

$$J_{w_i} = \begin{pmatrix} u_0 & u_1 & ... & u_{n-1} \end{pmatrix} \tag{11}$$

Where u_i represents the axis of rotation of the joint. J_{w_i} will be of shape 6x3 for all i.

1.1.3 Matrix I

The Matrix I for each link will be of form:

$$I_i = \begin{pmatrix} I_{xx_i} & & \\ & I_{yy_i} & \\ & & I_{zz_i} \end{pmatrix} \tag{12}$$

Seeing evey link as a cuboid with length a, height b and depth c.

1.2 Centrifugal- and Corelolis Forces

The $c(q, \dot{q})$ will also be of size 6x6. Its components will be calculated like this:

$$c_{ij} = \sum_{k=1}^{6} c_{ijk} * \dot{q} \quad , i, j \in 1, 2, ..., 6 \tag{13}$$

Where:

$$c_{ijk} = \frac{1}{2}(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i}) \tag{14}$$

1.3 Gravity

We get the last vector of the equation as:

$$g(q) = \begin{pmatrix} g_1 \\ g_2 \\ ... \\ g_6 \end{pmatrix} \tag{15}$$

Where we calculate:

$$g_i = -\sum_{k=1}^{6} J_{v_k}^i m_k g_0 \tag{16}$$

Where $J_{v_k}^i$ represents the i^th column of the k^th linear velocity matrix.
And:

$$g_0 = \begin{pmatrix} 0 \\ 0 \\ -9.81 \end{pmatrix} \tag{17}$$

representing the gravity in opposite direction of the z-axis.

1.4 Annotation

As the terms get humongous pretty fast, they are not shown in the report. To see the complete model use $print(s\_euler\_lagrange())$ in the code. To see $M(q)$, $print(c)$ and $g(q)$ use $print(mq[1])$, $cprint(c)$ and $print(g)$ inside $s\_euler\_lagrange()$.
All the commands will be commented out in the submitted code.

2 Code Implementation

2.1 Use of SymPy

The usage of this Python Toolbox allows for symbolical computations of all the matrices and partial deriavtives. The possibility to give variables predefined names proved to be highly advantageous in keeping track of the calculations. In general the used approach can be used for any configuration of $q_1 - q_6$, $l_1 - l_6$ and $lc_1 - lc_6$ as these symbolical values can be swapped out with numerical ones and the code will return the assiciated τ.

2.2 M(q)

2.2.1 Calculating the CoM's

As shown in section 1.1.1 the CoM are calculated through a sequence of matrix multipicaltions and returned in a list. The function, which calculated these CoM should not be called with numerical values, as it's existance is purely for the sake of deriving them later in the Jacobians.

2.2.2 Calculating linear Jacobians

Using the previously calculated CoM's, this function derives the points, by iterating through the CoM's. For each point, it iterates through $q_1 - q_6$ and for each q_i through x, y, z. The results will be put into matrix representing J_{v_i}, maintaining the structure shown in (9). All the J_{v_i} will be put into a list, which will be the output of the function.

2.2.3 Calculating angular Jacobians

Firstly the forward kinematics of the manipulator are calculated symbolically, to get a transformation matrix from the base frame to all local frames. Using these transformations, all the u_i can be calculated like in diffential kinematics.

2.2.4 Inertia I Matrix

Using SymPy, we get I_i. for each link.

2.2.5 Putting it all together

Using the six caluclated J_v, the six J_w and the six I we can use symbolic $3x3$ rotations matrices and -masses $m_1 - m_6$ to complete the equation in (2). as the $M(a)$ has to be symetrically, the programm will check this after calculation, verifying the correctness of the implentation.

2.3 C(q,dq)

2.3.1 Christoffel Symbol

Here the program will apply (14) with M, i, j, k being parameters of the function. Note that the calculation of c_{ijk} might take some time, as the terms to be derived are complex.

2.3.2 Generating C Matrix

Here an 6x6 matrix, filled with zeros will be the starting point. Then all the c_{ij} are calculated according to (13) and put into the correct spot into the resulting matrix.

2.4 G(q)

In the corresponding function, the list of all J_{v_i} is calculated. Iterating over $m_1 - m_6$ the 6x1 Matrix is calculated following (16).