Fundamentels of Robotics

Assignment 5 - Dynamics. Innopolis University, Fall 2020

Name: Hany Hamed
Group Number: BS18-Robotics
GitHub Repository: here (Please read the README, it has a lot of visualization)

Contents

FoR, Innopolis University, Fall 2020
Author: Hany Hamed

Section 1: Euler-Lagrange Solution

In Lagrange solution, I have used the "Moving Frames Algorithm" that was described in the lecture.

As it was described in the assignment, we have 2R manipulator as following:

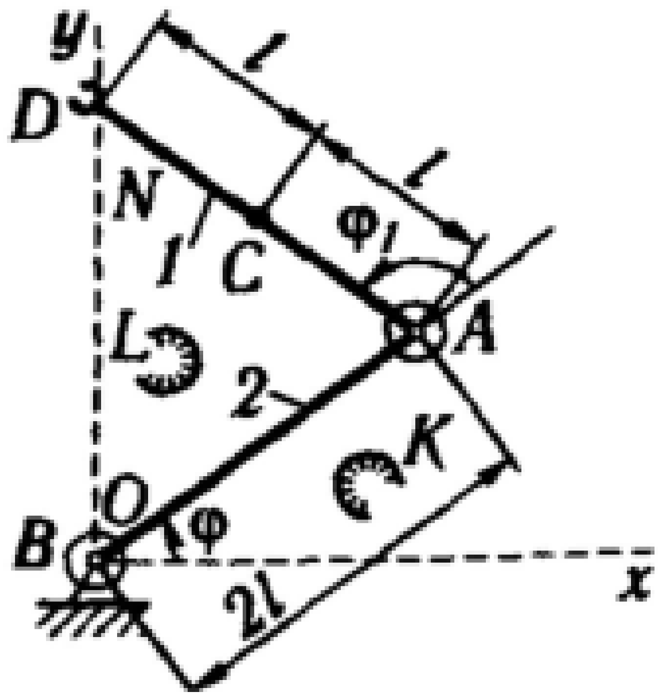

Figure 1. Manipulator from the assignment

I have used the same assumptions as in the lecture slides, with taking the differences: $d_1 = 0, l_1 = l_2 = 0.4, d_2 = 0.4$ and take the direction of the gravity in the negative direction of y as it is described from figure1

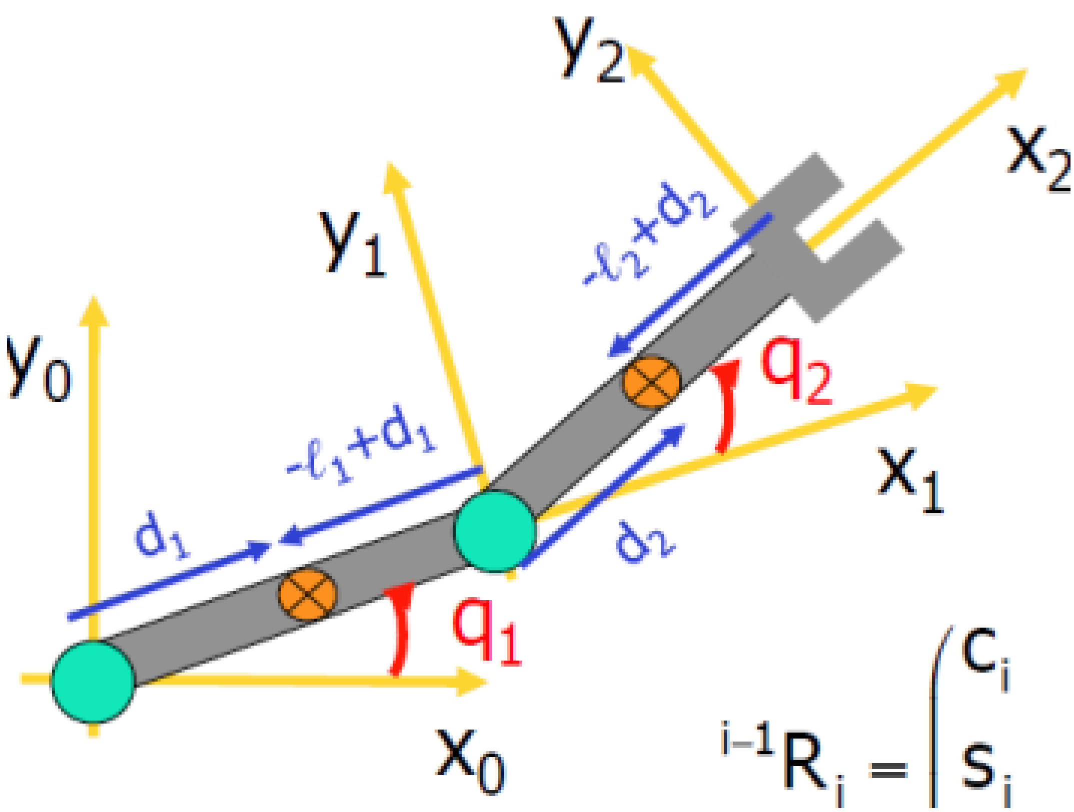

Figure 2. Manipulator from the lecture

The derivation is as following: $R_i^{i-1}(q) = R_z(q) \begin{bmatrix} cos(q) & -sin(q) & 0 \\ sin(q) & cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$g_0 = 9.81m/s^2$, $g = \begin{bmatrix} 0 \\ -g_0 \\ 0 \end{bmatrix}$

$r_{ci}^i = \begin{bmatrix} -l_i + d_i \\ 0 \\ 0 \end{bmatrix}$

And following from DH parameters FK, the link length is always in the x-axis direction, thus:

$r_{0,1}^{1} = \begin{bmatrix} l1 \\ 0 \\ 0 \end{bmatrix}$

$r_{1,2}^{2} = \begin{bmatrix} l2 \\ 0 \\ 0 \end{bmatrix}$

$w_0^0 = \vec{0},\ v_0^0 = \vec{0},\quad z_{i-1}^{i-1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

and with performing the following expressions for i=1,2:

$w_i^i = R_i^{i-1}(q_i)[w_{i-1}^{i-1} + \dot{q}_i z_{i-1}^{i-1}]$

$v_i^i = R_i^{i-1}(q_i)v_{i-1}^{i-1} + w_i^i \times r_{i-1,i}^i]$

$T_i = 0.5 * m_i * ||v_i^i||^2 + 0.5(w_i^i)^T I w_i^i$

Then, we can get:

$T_1 = 0.5(I_{c1,zz} + m_1 d_1^2)\dot{q}_1^2$

$T_2 = 0.5m_2(l_1^2\dot{q}_1^2 + d_1^2(\dot{q}_1 + \dot{q}_2)^2 + 2l_1 d_2 c_2 \dot{q}_1(\dot{q}_1 + \dot{q}_2)) + 0.5I_{c2,zz}(\dot{q}_1 + \dot{q}_2)^2$

$T = T_1 + T_2$ and we can form it in the following form $0.5\dot{q}^T M(q)\dot{q}$ in order to get M(q) matrix:

$M(q) \begin{bmatrix} a_1 + 2a_2c_2 & a_3 + a_2c_2 \\ a_3 + a_2c_2 & a_3 \end{bmatrix}$ such that: (Note: $(m_i = mass_i)$)

$a_1 = I_{c1,zz} + m_1 * d_1^2 + I_{c2,zz} + m_2 * d_2^2 + m_2 * l_1^2$
$a_2 = m_1 l_1 d_2$
$a_3 = I_{c2,zz} + m_2 * d_2^2$

Then, we can get $C(q, \dot{q})$ matrix as from the lecture rules:

$C(q, \dot{q}) \begin{bmatrix} -2 * a_2 s_2 \dot{q}_2 & -a_2 s_2 \dot{q}_2 \\ a_2 * s_2 \dot{q}_1, 0 & \end{bmatrix}$

and we get $G(q)$ as following:

$G(q) \left[a_4 c_1 + a_5 c_{1+2}, a_5 c_{1+2}]\right]$

such that:

$a_4 = g(m_1 d_1 + m_2 l_1)$
$a_5 = g(m_1 d_2)$

And we can construct the dynamics in the following form:

$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau(t) = u(t)$ as we already defined all the matrices above

For the direct problem, we have given the initial q and the initial $\dot{q}$ and the desired input, then doing the following (numerical integration - Semi implicit euler integration) we can obtain the trajectory:

$\ddot{q}_{t+1} = M^{-1}(\tau - G - (C\dot{q}_t))$
$\dot{q}_{t+1} = \dot{q}_t + \ddot{q}_{t+1}\Delta t$
$q_{t+1} = q_t + \dot{q}_{t+1} * \Delta t$

such that $\Delta t = 0.0004$ *seconds* is small time (discretization) for the simulation

Section2: Newton-Euler Solution

In Newton's solution, I have used the recursive algorithm that was described in the lecture (The full description can be found in the textbook starting from page 283). It is described in the code:

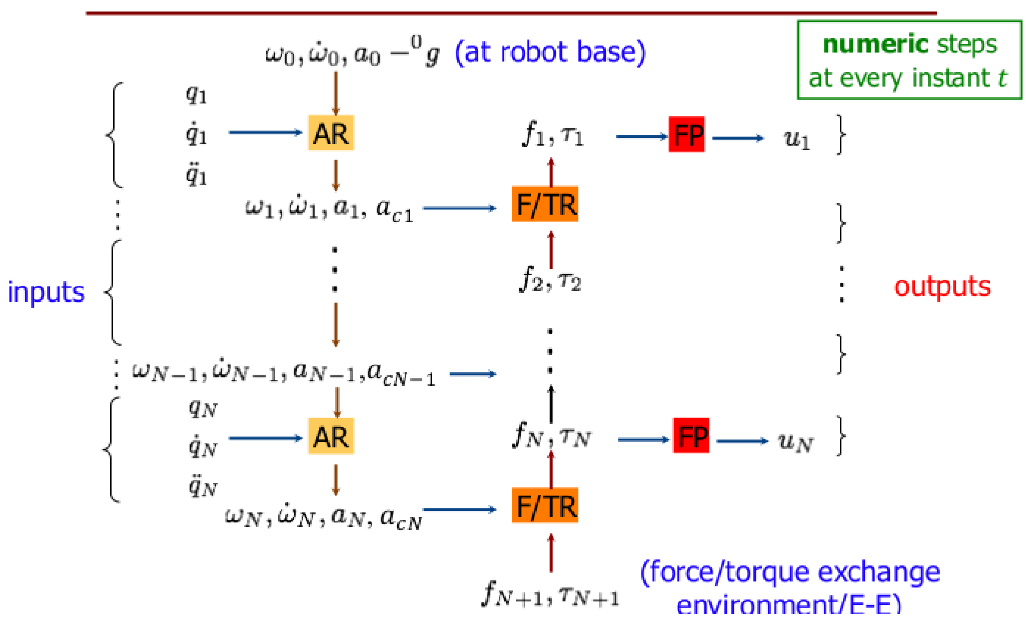

Figure 3. Algorithm from the lecture

Forward feeding:

```python
# AR (Accelerations recursion)
# input: q, dq, ddq
# output: w, dw, a, ac
for i in range(1,len(self.l)+1):
    w_i.append(R_iml_i[i].T @ (w_i[i-1]+ dq[i-1] * z_i[i]))
    # print(w_i[i])
    dw_i.append(  R_iml_i[i].T @ (dw_i[i-1] \
                + ddq[i-1] * z_i[i] + np.cross((dq[i-1]*w_i[i-1]).squeeze(), z_i[i].squeeze()).reshape(3,1)))
    # print(dw_i[i])
    a_i.append(   R_iml_i[i].T @ a_i[i-1] \
                + np.cross(dw_i[i].squeeze(), r_iml_i[i].squeeze()).reshape(3,1) \
                + np.cross(w_i[i].squeeze(),np.cross(w_i[i].squeeze(), r_iml_i[i].squeeze())).reshape(3,1))
    # print(a_i[i])
    ac_i.append(  a_i[i] \
                + np.cross(dw_i[i].squeeze(), r_i_ci[i].squeeze()).reshape(3,1) \
                + np.cross(w_i[i].squeeze(), np.cross(w_i[i].squeeze(), r_i_ci[i].squeeze())).reshape(3,1))
    # print(ac_i[i])
```

Figure 4. AR block in the code

Backward feeding:

```python
# F/TR (Forces/Torques recursion)
# input: w, dw, a, ac
# output: u
for i in range(len(self.l),0, -1):
    f_i[i] = (R_iml_i[i+1]@f_i[i+1] + self.mass[i-1] * (ac_i[i]))
    # print(f_i[i])
    # In general, the inertia should be matrix but here it will not be different result as the matrix is zeros except (3
    tau_i[i] = R_iml_i[i+1]@tau_i[i+1] \
            - np.cross(f_i[i].squeeze(),(r_iml_i[i]+r_i_ci[i]).squeeze()).reshape(3,1) \
            + (np.cross((R_iml_i[i+1]@f_i[i+1]).squeeze(), (r_i_ci[i]).squeeze()).reshape(3,1)) \
            + self.inertia[i-1]*dw_i[i]+ np.cross(w_i[i].squeeze(),(self.inertia[i-1]*w_i[i]).squeeze()).reshape(3,1)
```

Figure 5. F/TR block in the code

Getting torques:

```python
# FP
for i in range(2):
    # u.append(f_i[i+1].T*z_i[i+1])
    u.append(tau_i[i+1].T@z_i[i+1])
```

Figure 5. FP block in the code

Section3: Testing

For the full testing with gifs and images, please check the repository page.