Robotic Systems Engineering
Coursework 3: Actuators, Mechanisms and Robot Dynamics

Huijuan Ma
zczqhm0@ucl.ac.uk

Jan. 1, 2023

Part I

1. a. **3 degrees of freedom**. To finish picking and sorting tasks, the end-effector needs move a target position then pick the candy to another target position with 3 DOFs for position (translation motion of width and height, as well as rotational motion for precise positioning of candies).

   b. **Series Manipulator**, similar to the robot $SCARA$, which is easier to control than parallel one to have a faster operation speed. Based on 2 prismatic and 1 revolute joints, it can operate faster with 2 translation DOFs (positioning at width and height) and 1 revolution DOF (round z-axis, positioning at the horizontal plane) [1] than that with 3 prismatic joints, i.e. $Cartesian$. Since all candies has quasi-spherical shape, the system directly use a **servo mechanical gripper** with **gear transmissions** at the end-effector of the robotic arm to control the grabbing and releasing action.

   c. **Conveyor belt**: **brushed DC motors** are easy to control with only two wires and no controller required for a constant speed, which meets the demand of a conveyor moving at a constant speed. **Chains** meet the demand of one direction motion, and has efficient power transmission.
   Manipulator: **Brushless DC motors** are able to do continuous localization to pick random appeared candies with higher efficiency, speed, torque-to-weight ratio beacuse of the absence of the brushes, making it more sensitive to input and easier to control to reach accurate position point. **Belt** has lower inertia and less frictional loss from slipping, which is suitable for the gripper to grab candies repetitively.

   d. **Encoders** in actuators (motors) can detect the movement of links in angles or distance, then we can calculate the actual position of the end-effector. Install an **IMU** at the end effector to detect the pose and make the system more accurate.
   Camera: should be mounted at a fixed position near the end-effector to obtain the same front view related to the end-effector. It takes pictures with a high frequency when the end-effector is at a certain height (measured by the encoder for the prismatic joint), and then detect and localize candies in 2D according to the values for pixels.

e. **Joints**: the wear and tear can be minimized by high quality materials, regular maintenance, and uniform distribution of force between joints, and reasonable path planning and trajectory planning to smooth control the changes of velocities, accelerations, forces and torques.

f. **Color**: Modify the code about color information from the camera to focus on pixels of previously specified colors.
Weight: Set a weight sensor at the end-effector to detect the candy's weight when picking up. Only if the weight exceeds threshold, put it in target basket.

Part II

2. Matrix formulation of the equations of motion is

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau \tag{1}$$

a. $get\_jacobian\_centre\_of\_mass$: To compute Jacobian at the center of mass, it need to use forward kinematics to get the transformation matrices in $T0i\_joints\_COM$ for the center of mass of all links first. Then, rearrange all z axis of the rotation part of $T0i\_joints\_COM$ in z_cm and translations of $T0i\_joints\_COM$ in o_cm and pl. The joints here are all revolute. With the equation

$$J_{P_j}^{(l_i)} = \begin{cases} \mathbf{z}_{j-1} & \text{for a prismatic joint} \\ \mathbf{z}_{j-1} \times (\mathbf{p}_{l_i} - \mathbf{o}_{j-1}) & \text{for a revolute joint} \end{cases} \tag{2}$$

$$J_{O_j}^{(l_i)} = \begin{cases} 0 & \text{for a prismatic joint} \\ \mathbf{z}_{j-1} & \text{for a revolute joint} \end{cases} \tag{3}$$

Assume $z_0 = [0, 0, 1]^T$ and $o_0 = [0, 0, 0]^T$, use equations 2 and 3 to calculate $J_{P_j}^{(l_i)}$ and $J_{O_j}^{(l_i)}$, then combine them into a jacobian matrix with 6 by 7 shape.

b. $get\_B, B(q)\ddot{q}$: To compute inertia matrix B, we can use the equation

$$B(q) = \sum_{i=1}^{n} \left(m_{l_i} J_P^{(l_i)^T} J_P^{(l_i)} + J_O^{(l_i)^T} \text{Inertia\_Tensor}_{l_i} J_O^{(l_i)} \right) = [b_{ij}]_{n \times n} \tag{4}$$

We can have a jacobian matrix for each joint. Since there are 7 joints, for inertia matrix B it only need the jacobian and forward kinematics at the centre of mass from second joint to last joint, the array of Jacobian metrics J_cm is 6 by 7 by 6 and the array of transformation metrics T_cm is 4 by 4 by 6. The first joint is usually the base joint of the robot, without containing mass and inertia, and does not involve the center of mass of the robot. Therefore, when calculating the overall dynamics of the robot, starting from the second joint is usually more meaningful. This can better capture the dynamic behavior of the robot in joint space, while ignoring the influence of the base joints. For inertia tensor,

$$\text{Inertia\_Tensor}_i = {}^0 R_{G_i} I_{l_i} R_{G_i}^T \tag{5}$$

For moment on inertia of each link I_{l_i}, it is provided in the code as a principal axes of inertia (all the cross-products of inertia are equal to zero).

c. $get\_C\_times\_qdot, C(q, \dot{q})\dot{q}$: To compute the dynamic component $C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$,

$$c_{ij} = \sum_{j=1}^{n}(\sum_{k=1}^{n} h_{ijk}\dot{q}_k)\dot{q}_j \tag{6}$$

h_{ijk} terms are known as Christoffel symbols.

$$h_{ijk} = \frac{\partial b_{ij}(\boldsymbol{q})}{\partial q_k} - \frac{1}{2}\frac{\partial b_{jk}(\boldsymbol{q})}{\partial q_i} \tag{7}$$

Because partial derivative calculation is too complex with code, we can use an approximation of it:

$$\lim_{h\to 0} \frac{f\Big(x+h\Big) - f\Big(x\Big)}{h} \tag{8}$$

d. $get\_G, g(q)$: To compute the dynamic component $g(q)$, use the equation

$$g_n = \frac{\partial P(\boldsymbol{q})}{\partial \boldsymbol{q}_n} \tag{9}$$

The potential energy $P(\boldsymbol{q})$ can be calculated by

$$\mathcal{P}(\boldsymbol{q}) = -\sum_{i=1} m_{l_i}\boldsymbol{g}_0^T\boldsymbol{p}_{l_i} \tag{10}$$

with $g_0^T = [0, 0, -g]$.
Because partial derivative calculation is too complex with code, we can use an approximation of it:

$$\lim_{h\to 0} \frac{f\Big(x+h\Big) - f\Big(x\Big)}{h} \tag{11}$$

3. **Hypothesis: Huygens-Steiner theorem** says that the moment of inertia of a rigid body around an axis parallel to an axis through the center of mass is equal to the product of the moment of inertia around the axis through the center of mass plus the square of the distance between the two axes and the mass of the rigid body, showing as the equation:

$$J = J_C + md^2 \tag{12}$$

J is the moment of inertia about the parallel axis, J_C is the moment of inertia about an axis through the center of mass, m is the object's mass, and d is the perpendicular distance between the two parallel axis.
Derivation: In the figure 1, I assume that the height of an object is extremely small, the axis through the centre of mass is C, the centre of mass is at point A, another axis that parallels to C is C', the intersection point of C' with object surface is B, the distance between two paralleled axis C and C' is d, a random point on the surface is D, the distance between points B, A and D are $AD = r$ and $BD = r'$, and the angle between AB and AD is α. Based on triangle geometry analysis, we can have

$$r'^2 = (d + r\cos\alpha)^2 + r\sin^2\alpha = d^2 + r^2 + 2dr\cos\alpha \tag{13}$$

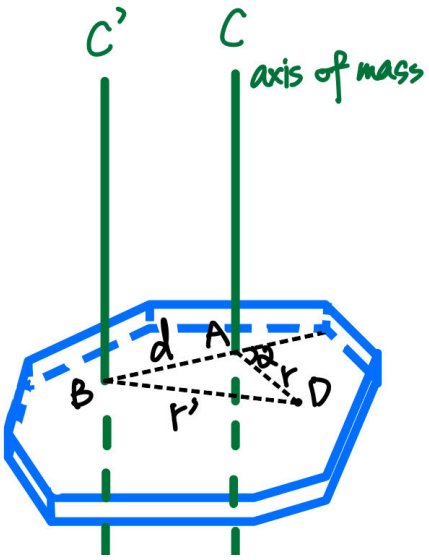

Figure 1: An object with the center of mass

Based on the definition of moment of inertia

$$J = \sum r_i^2 m_i \tag{14}$$

we can have $J_{C'} = \sum d^2 + r_i^2 + 2dr_i \cos\alpha_i m_i = \sum m_i r_i^2 + \sum 2m_i dr_i \cos\alpha_i + md^2$ and $J_C = \sum m_i r_i^2$. So, it can be written as

$$J_{C'} = J_C + \sum 2m_i dr_i \cos\alpha_i + md^2 \tag{15}$$

$\sum 2m_i dr_i \cos\alpha_i$: set $\sum 2m_i dr_i \cos\alpha_i = \sum 2m_i dx_i$ where x_i is the distance to the centre of mass. Because the centre of mass is the weighted average position of all mass elements in the system, for every mass element on one side of the center of mass, there is an equivalent mass element on the opposite side with the same magnitude of m_i but opposite sign for r_i, which means the second term of the equation 15 $\sum 2m_i dr_i \cos\alpha_i = 0$.
Therefore, we can have $J_{C'} = J_C + md^2$
Applications: It provides a easier way to calculate the moment of inertia about arbitrary axis which does not need to do integral or summation over the object again, making the calculation transferable between different axis. Accurate calculation of moment of inertia can improve the controllability of the robot system, effectively simulate dynamics of the robot, and accurately predict and control the robot behaviour.

4. **Forward Dynamics** is calculating the joint accelerations, joint velocities and joint parameters (joint positions) at each moment with a set of initial joint parameters (joint positions: joint angle for revolute joints and link offset for prismatic joints), initial joint velocities and joint torques at each moment to predict the movement trends of robots. Based on the equation 1, the joint accelerations can be calculate by

$$\ddot{\mathbf{q}} = \mathbf{B}^{-1}(\mathbf{q})(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})) \tag{16}$$

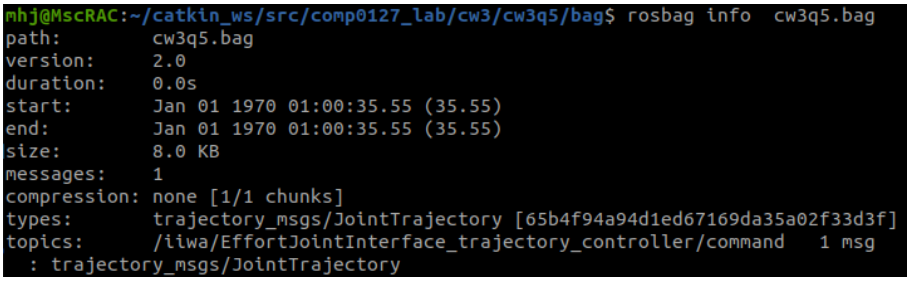

Figure 2: Information of cw3q5.bag

where the inertia matrix **B(q)** is invertible.
Main applications: **Forward Dynamics** is used to simulate a manipulator with different conditions and inputs (torques) or predict further movement of the robot for designing of the controller.
Difficulties: Most robotics dynamics is non-linear, leading to an inaccurate and inefficient model. Besides, if it has many DOFs, the expression of the manipulator and the calculation of **forward Dynamics** become more complex. **Inverse Dynamics** is calculating the joint torques with known joint parameters, including joint positions, joint velocities and joint accelerations, which means finding the torques to achieve certain joint parameters.
Main applications: **Inverse Dynamics** is used to online control the motion and force of a robot in real systems so that the manipulator can move in designated trajectories by calculating the torque required to achieve precise motion.
Difficulties: Most robotics dynamics is non-linear, making it computationally intensive. For robots with degrees of freedom exceeding the required degrees of freedom for specific tasks, **inverse Dynamics** can have multiple solutions, leading to redundancy. Besides, it may be sensitive to inaccuracies in robot models or sensor measurements, resulting in challenges in achieving precise control with uncertainty.

5. a. **Answer**: Use the command $rosbag\ \ info\ \ cw3q5.bag$ to find the type of message in the bagfile, shown in the figure 2. It has only one message and the message type is called $trajectory\_msgs/JointTrajectory$ with a structure shown in the figure **??**, which contains joint positions, joint velocoties, joint accelerations as well as duration of movement. Only joint positions are defined, others are all zeros by default.
Code: Since there is only one message in the bagfile and the robot has 7 joints, I assume the shape of all information are shaped in a 7-by-2 matrix to store the initial position and the only one message about the robot.

   b. **Answer**: **Forward Dynamics**.
This task is a simulation of the manipulator, and asks to compute the joint accelerations throughout the a given trajectory with torques which can be subscribed in the topic $/iiwa/joint\_states$. It is a problem of **Forward Dynamics**.

   c. **Trajectory**: after loading intermediate points in the bag file, I use $forward\_kinematics(joint)$ calculate the transformation matrix of the end-effector in frame 0 with a list describing each joint angle. The current transformation matrix of the end-effector need to

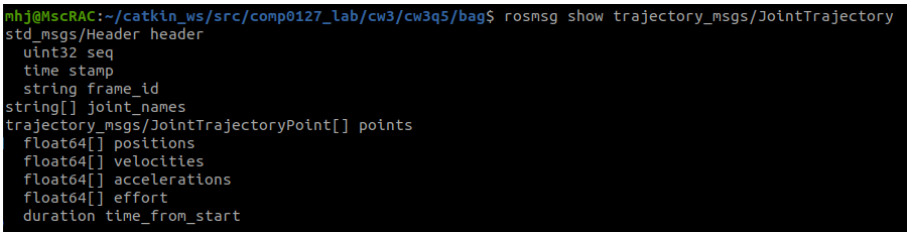

Figure 3: Message Type of cw3q5.bag

be calculated first, which will be used to define trajectory (via points) with the target end-effector positions in Cartesian.

$$
\begin{aligned}
{}^{i-1}\boldsymbol{T}_i &= \boldsymbol{Trans}_{z_{i-1}}(d_i)\boldsymbol{Rot}_{z_{i-1}}(\theta_i)\boldsymbol{Trans}_{x_i}(a_i)\boldsymbol{Rot}_{x_i}(\alpha_i) \\
&= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (17)
\end{aligned}
$$

$$
{}^{0}T_i = {}^{0}\,T_1^1 T_2^2 T_3 ... {}^{i-2}T_{i-1}^{i-1} T_i \qquad (18)
$$

Forward Kinematics is the process of calculating the pose with a given group of joint parameters (joint angle for revolute joints and link offset for prismatic joints). With DH parameters (link Length a, link twist α, link offset d, joint angle θ), given joint parameters (joint positions) and equations 17 and 18, we can calculate transformation martrices describing the relationship between every frame on the joint.
For **publishing the trajectory**, these via points are published in the function run as a $Marker$ with points in Cartesian form to show in $rviz$. These points are achieved from the last column of transformation matrices.
Last, **rearrange all trajectory information**, including joint parameters (joint angles/positions, joint velocities, joint accelerations, and processing time in seconds and nanoseconds) at via points.

d. **Subscribe**: First, I need to find which topic will content the information about torques. The joint parameters can be subscribed in the topic called $/iiwa/joint\_states$ as a list $effort$. I set another subscriber $joint\_torque\_sub$ in the main function use the class $Substriber$ to subscribe the message $JointState$ in the topic $/iiwa/joint\_states$ and callback the function $joint\_torque\_callback$ in the class $iiwa14Accelerations()$, and use $rospy.spin()$ to block until ROS node $iiwa14\_cw3$, defined in the class $iiwa14Accelerations()$ is shutdown.

e. **Calculate the joint accelerations**: The function $iiwa14Accelerations()$ calculates accelerations using the function $cal\_acceleration$ with the equation of $forwarddynamics$ 16 and dynamic components - $\mathbf{B}(\mathbf{q})$ (inertia matrix, equation 4), $C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ (equation 6) and $\mathbf{g}(\mathbf{q})$ (equation 9). To ensure the normal operation of the robotic arm, a delay of 1 second is required for everything to load up. The change of accelerations for each joint as a function of time is shown in the figure 4.

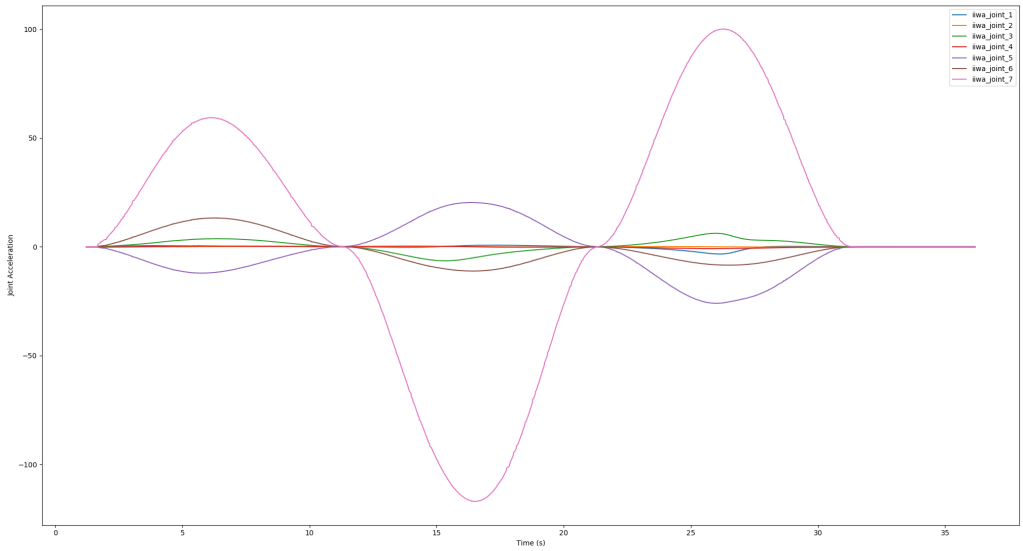

Figure 4: Joints Accelerations

References

[1] Saravana Mohan Mariappan and Anbumalar Veerabathiran. “Modelling and simulation of multi spindle drilling redundant SCARA robot using SolidWorks and MATLAB/SimMechanics/Modelad y simulacion de un robot redundante de perforacion tipo manipulador SCARA utilizando SolidWorks y MATLAB/SimMechanics”. spa ; eng. In: *Revista Facultad de Ingeniería* 81 (2016), pp. 63–. ISSN: 0120-6230.

END OF COURSEWORK