ROBOT MODELING AND CONTROL

ECE470S

LAB2 : Velocity Kinematics for the PUMA 560 Manipulator

1 Purpose

The purpose of this lab is to derive the velocity kinematics for the PUMA 560 robot. This includes the development of the geometric and analytic Jacobians. These equations will be implemented as Matlab functions and used to perform basic computer simulations.

2 Introduction

The exercises in this lab will build upon the forward and inverse kinematics developed in Lab 1. The forward and inverse kinematics define the relationships between Cartesian positions of the end-effector and the joint positions. The geometric and analytic Jacobians define the velocity relationships.

Geometric Jacobian

Define the *twist*, ξ, as

$$\xi = \begin{bmatrix} \dot{o}_n^0 \\ \omega_n^0 \end{bmatrix}$$

where $\dot{o}_n^0$ is the linear velocity of the end effector in the base frame, and ω_n^0 is the angular velocity of the end effector in the base frame. The *Jacobian* $J(q)$, also referred to as the *geometric Jacobian* or the *manipulator Jacobian,* is a $6 \times n$ matrix relating the joint velocities to the twist as follows

$$\xi = J(q)\dot{q}$$

The Jacobian can be written as

$$J(q) = \begin{bmatrix} J_v(q) \\ J_\omega(q) \end{bmatrix},$$

where the $3 \times n$ matrix $J_v(q)$ relates the joint velocities to the linear velocity of the end effector, and the $3 \times n$ matrix J_ω relates the joint velocities to the angular velocity of the end effector. For a robot with n joints, $J_v(q)$ is expressed as

$$J_v(q) = \begin{bmatrix} J_{v_1}(q) & \dots & J_{v_n}(q) \end{bmatrix},$$

where

$$J_{v_i} = \begin{cases} z_{i-1}^0 \times (o_n^0 - o_{i-1}^0) & \text{if joint } i \text{ is revolute} \\ z_{i-1}^0 & \text{if joint } i \text{ is prismatic.} \end{cases}$$

Similarly, $J_\omega(q)$ is expressed as

$$J_\omega(q) = \begin{bmatrix} J_{\omega_1}(q) & \dots & J_{\omega_n}(q) \end{bmatrix},$$

where

$$J_{\omega_i}(q) = \begin{cases} z_{i-1}^0 & \text{joint } i \text{ is revolute} \\ 0 & \text{joint } i \text{ is prismatic.} \end{cases}$$

Therefore, $J(q)$ depends only on the joint axes $z_0^0, \ldots, z_5^0$ and the origins $o_0^0, \ldots, o_6^0$, which can be evaluated from the forward kinematics derived in Lab 1. More specifically, recall that

$$H_i^0 = \begin{bmatrix} R_i^0 & o_i^0 \\ 0 & 1 \end{bmatrix}.$$

Since the vector z_i^0 is the third column of R_i^0, we see that from H_i^0 we can extract z_i^0 and o_i^0. To calculate H_i^0, we simply multiply $H_i^0 = A_1 \cdot \ldots \cdot A_i$, where A_i is evaluated from the values in the DH table as

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Analytic Jacobian

The *analytic Jacobian*, $J_a(q)$ is similar to the geometric Jacobian, but it relates the joint speeds to the end effector linear velocity and to the time derivatives of the Euler angles representing the orientation of the end effector. Define

$$\chi = \begin{bmatrix} o_n^0 \\ \alpha \end{bmatrix},$$

where, as usual, o_n^0 is the origin of the end effector frame expressed in base frame coordinates, and $\alpha = \begin{bmatrix} \phi, \theta, \psi \end{bmatrix}^T$ is the vector of Euler angles expressing the end effector rotation R_n^0. Then, $J_a(q)$ is defined by the expression

$$\dot{\chi} = \begin{bmatrix} \dot{o}_n^0 \\ \dot{\alpha} \end{bmatrix} = J_a(q)\dot{q}.$$

We can express the end effector angular velocity as

$$\omega_n^0 = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\theta \\ 0 & c_\phi & s_\phi s_\theta \\ 1 & 0 & c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = B(\alpha)\dot{\alpha},$$

and so we can express the analytic Jacobian in terms of the geometric Jacobian as

$$J_a(q) = \begin{bmatrix} I & 0 \\ 0 & B^{-1}(\alpha) \end{bmatrix} J(q),$$

provided that $\sin\theta \neq 0$, so that B is invertible.

3 Preparation

1. Write by hand pseudocode of the algorithm needed to compute the geometric Jacobian of the Puma560.

   Algorithm inputs: The robot structure and the vector $q = [\theta_1, \ldots, \theta_6]^\top$.

   Algorithm output: The 6×6 Jacobian matrix $J(q)$.

2. Write by hand pseudocode of the algorithm needed to compute the analytic Jacobian of the Puma560. Your algorithm may use the previous geometric Jacobian algorithm as a subroutine.

   Algorithm inputs: The robot structure and the vector $q = [\theta_1, \ldots, \theta_6]^\top$.

   Algorithm output: The 6×6 analytic Jacobian matrix $J_a(q)$.

3. Write a function `jacobian.m` that computes the geometric Jacobian of the PUMA 560. Your function must have these arguments

   ```
   J = jacobian(q,myrobot)
   ```

   where `q` is a column vector of joint angles, `myrobot` is the robot structure from Lab 1, and `J` is the geometric Jacobian matrix.

   Test your function as follows. You should get

   ```
   >> jacobian([pi/4 pi/3 -pi/2 pi/4 pi/6 -pi/6],myrobot)

   ans =

   -0.0995 0.1308 0.3955 0.0067 0.1970 -0.0000
   -0.1350 0.1308 0.3955 -0.0933 0.0238 -0.0000
   0 -0.0251 -0.2413 0.0354 0.0254 -0.0000
   0 0.7071 0.7071 -0.3536 0.0670 0.1603
   0 -0.7071 -0.7071 -0.3536 -0.9330 -0.3397
   1.0000 0.0000 0.0000 -0.8660 0.3536 -0.9268
   ```

4. Write a function `ajacobian.m` that computes the analytic Jacobian of the PUMA 560. Your function must have these arguments

   ```
   Ja = ajacobian(q,myrobot)
   ```

   where `q` is a column vector of joint angles, `myrobot` is the robot structure from Lab 1, and `Ja` is the analytic Jacobian matrix.

   Test your function as follows. You should get

   ```
   >> ajacobian([pi/4 pi/3 -pi/2 pi/4 pi/6 -pi/6],myrobot)

   ans =

   -0.0995 0.1308 0.3955 0.0067 0.1970 -0.0000
   -0.1350 0.1308 0.3955 -0.0933 0.0238 -0.0000
   0 -0.0251 -0.2413 0.0354 0.0254 -0.0000
   1.0000 2.3225 2.3225 -0.4495 2.5060 0.0000
   0 0.3377 0.3377 -0.4706 -0.3377 0.0000
   0 2.5060 2.5060 0.4495 2.3225 1.0000
   ```

5. Read Sections 6.3 and 6.4 and draw the block diagram of a controller making the joint angles q of the robot track a desired reference signal $q_{\text{ref}}(t)$. Highlight the signals $q_{\text{ref}}(t)$, $\dot{q}_{\text{ref}}(t)$, and $\ddot{q}_{\text{ref}}(t)$ in your block diagram.

6. Using motor inertia and viscous friction coefficients in the table below,

Joint	J_m	B
1	$2 \cdot 10^{-4}$	$1.48 \cdot 10^{-3}$
2	$2 \cdot 10^{-4}$	$8.17 \cdot 10^{-4}$
3	$2 \cdot 10^{-4}$	$1.38 \cdot 10^{-3}$
4	$3.3 \cdot 10^{-5}$	$7.12 \cdot 10^{-5}$
5	$3.3 \cdot 10^{-5}$	$8.26 \cdot 10^{-5}$
6	$3.3 \cdot 10^{-5}$	$3.67 \cdot 10^{-5}$

compute gain matrices

$$K_p = \begin{bmatrix} K_{p1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & K_{p6} \end{bmatrix}, \; K_d = \begin{bmatrix} K_{d1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & K_{d6} \end{bmatrix},$$

such that the poles of the closed-loop system are all at $s = -2$ (the closed-loop system has twelve poles, two poles per joint).

4 Experiment

In this lab, you will use your Jacobian functions to test a basic control algorithm making the end effector position and orientation track a desired trajectory. You will be given the desired end effector position and orientation in the form of a sequence of homogeneous transformations at various sampling instants, $\{H_{\text{ref}}(t_i)\}$. You will also be given a sequence expressing the time derivative of the homogeneous transformation matrix at the sampling instants, $\{\dot{H}_{\text{ref}}(t_i)\}$. Using this information, you will compute a sequence of desired joint angles $q_{\text{ref}}(t_i)$. Using the identity $\dot{q} = [J(q)]^{-1}\xi$, you will compute the sequence of desired joint velocities $\dot{q}_{\text{ref}}(t_i)$. Using cubic spline interpolation, from the samples you will create functions $q_{\text{ref}}(t)$ and $\dot{q}_{\text{ref}}(t)$. You will also approximate $\ddot{q}_{\text{ref}}(t)$. The reference signals $q_{\text{ref}}(t)$, $\dot{q}_{\text{ref}}(t)$, and $\ddot{q}_{\text{ref}}(t)$ so generated will be used to simulate the tracking controller in Sections 6.3 and 6.4 of the textbook. You will verify that, starting from an initial condition far away from the desired trajectory, the controller will make the robot end effector converge to the reference signal and follow it. Note that the simulation you will perform in this lab completely ignores the robot dynamics. It only models the dynamics of the motors and, as such, it represents a significant simplification.

The significance of this experiement is that you'll learn how to systematically translate reference positions and velocities of the end effector into suitable reference joint angles and reference joint velocities.

4.1 Computation of joint reference signals

Download the file `Href.mat` from the course webpage and load it in the workspace. The file loads three variables:

- `t` - a sequence of sample times from 0 to 6π.
- `Href` - a matrix $4 \times 4 \times 100$ containing a sequence of end effector homogeneous transformations. For each `i` between 1 and 100, the 4×4 matrix `Href(:,:,i)` is the homogeneous transformation matrix of the end effector at time `t(i)`:

$$H_{\text{ref}}(t(i)) = \begin{bmatrix} R_n^0(t(i)) & o_n^0(t(i)) \\ 0 & 1 \end{bmatrix}.$$

- `Hrefdot` - a matrix $4 \times 4 \times 100$ containing a sequence time derivatives of Href. For each `i` between 1 and 100, the 4×4 matrix `Hrefdot(:,:,i)` is the time derivative

$$\dot{H}_{\text{ref}}(t(i)) = \begin{bmatrix} \dot{R}_n^0(t(i)) & \dot{o}_n^0(t(i)) \\ 0 & 0 \end{bmatrix}.$$

Perform the following steps in a Matlab file `lab2.m`

1. Using `Href` and the inverse kinematics subroutine you developed in Lab 1, find `qref`, the link trajectory corresponding to `Href`. `qref` should be a matrix 6×100.
2. For each `i` between 1 and 100
   (a) Use `Href(:,:,i)` and `Hrefdot(:,:,i)` to extract the end effector linear and angular velocities, `odot` and `omega`, respectively. To compute `omega`, you need to use the fact that $S(\texttt{omega}) = \dot{R}_n^0 (R_n^0)^\top$.
   (b) Using `odot` and `omega` and your geometric Jacobian function, find the corresponding joint velocities `qdref(:,i)` according to the formula

$$\dot{q}_{\text{ref}} = [J(q_{\text{ref}})]^{-1} \begin{bmatrix} \texttt{odot} \\ \texttt{omega} \end{bmatrix}.$$

   (c) Using `Href(:,:,i)` and the Robotics Toolbox function `tr2eul`, get the Euler angles `phi, theta, psi` of the end effector.
   (d) Using the fact that

$$\dot{\alpha} := \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = B(\alpha)^{-1} \omega_n^0, \quad \text{where } B(\alpha) = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\theta \\ 0 & c_\phi & s_\phi s_\theta \\ 1 & 0 & c_\theta \end{bmatrix}$$

   compute `alphadot`.
   (e) Using `alphadot`, `odot`, and your analytic Jacobian function, find the corresponding joint velocities `qdref1(:,i)` according to the formula

$$\dot{q}_{\text{ref}} = [J_a(q_{\text{ref}})]^{-1} \begin{bmatrix} \texttt{odot} \\ \texttt{alphadot} \end{bmatrix}.$$

   If $\sin\theta = 0$, then $B(\alpha)$ is singular. Put a check in place that skips this step and the next one when $\sin\theta < 10^{-5}$.
   (f) Verify that `qdref1(:,i)` is equal to `qdref(:,i)` (check that the norm of the difference of these two vectors is less than 10^{-5}). If not, there's a mistake somewhere in your code.
3. You now have matrices `qref` and `qdref` of dimension 6×100 containing samples of the signals $q_{\text{ref}}(t(i))$, $\dot{q}_{\text{ref}}(t(i))$, $i = 1, \ldots, 100$. To generate reference signals, $q_{\text{ref}}(t)$, $\dot{q}_{\text{ref}}(t)$, we will use cubic spline interpolation. In your `lab2.m` file, issue the commands

```
splineqref=spline(t,qref);
splineqdref=spline(t,qdref);
```

   The above creates two cubic splines interpolating the sequences `qref` and `qdref`.
4. To implement the joint controller, we need $q_{\text{ref}}, \dot{q}_{\text{ref}}, \ddot{q}_{\text{ref}}$, so we need one more derivative of q_{ref}. You can compute this derivative by creating a new spline `splineqddref` as follows:

```
d=length(splineqdref.coefs);
splineqddref=splineqdref;
splineqddref.coefs=splineqdref.coefs(:,1:3).*(ones(d,1)*[3 2 1]);
splineqddref.order=3;
```

The above code creates a quadratic spline `splineqddref` whose polynomials are the derivatives of the polynomials in `splineqdref`.

Explanation: A cubic spline is a collection of "pieces." Each piece is a cubic polynomial, valid on a given time interval. In Matlab, the coefficients of all polynomial pieces in a spline such as `splineqdref` are contained in the matrix `splineqdref.coefs`. Each row of this matrix contains the coefficients of a corresponding polynomial piece. For instance, the coefficients of the cubic polynomial $5t^3-2t^2+t+4$ are represented by the row vector `C= [5 -2 1 4]`. The derivative of the polynomial above is $15t^2-4t^2+1$. The coefficients of this polynomial are `[15 -4 1]`, and can be directly obtained from `C` as `C(1:3).*[3 2 1]`. The code above performs precisely this operation on each polynomial piece.

4.2 Test Independent Joint Controller

You'll now use the three splines you've generated so far, `splineqref, splineqdref,splineqddref`, to test an independent joint controller making $q(t)$ track $q_{\text{ref}}(t)$. This controller ignores the dynamics of the robot. It is solely based on the dynamics of the motors.

1. Download the file `motors.m`, edit it, and enter the gain matrices `Kp` and `Kd` you designed in your preparation.
2. To simulate the motors with the jont controllers, in your `lab2.m` file, issue the commands

   ```
   sys=@(t,x)motors(t,x,myrobot,splineqref,splineqdref,splineqddref);
   Ts=0.02;
   q0=[3*pi/2;zeros(11,1)];
   [t,q]=ode45(sys,[0:Ts:6*pi]',q0);
   ```

3. Generate six figures. Each figure should display the plot of a component of the solution `q(:,i)` against time, and the corresponding component of the reference signal `qref(:,i)`, where the matrix `qref` is generated form the spline `splineqref` using the command `qref=ppval(splineqref,t)'`.
4. In a separate figure, plot the desired end effector trajectory:

   ```
   hold on;
   oref=squeeze(Href(1:3,4,:));
   plot3(oref(1,:),oref(2,:),oref(3,:),'r')
   view(-125,40);
   ```

   Now using the Robotics Toolbox command `plot`, plot the robot trajectory in three-space

   ```
   plot(myrobot,q(:,1:6))
   hold off
   ```

   You should notice that the robot end effector converges rapidly to the desired trajectory, and then follows it.

5 Submission

Each student will submit the lab preparation and the following material on Quercus.

1. A zipped folder containing your Matlab functions. Thoroughly comment your code. Your folder may contain intermediate Matlab functions, but calls to the main functions `jacobian` and `ajacobian` must work without any modification required.
2. A Matlab file `lab2.m` working out the various steps in Section 4.