1928
K. N. Toosi University of Technology

Robotics Exam

Chalmers Phd examination

Amir Saman Mirjalili

2024

Contents

Abstract

This document contains some basic exercises that evaluate your suitability for working in my research group at Chalmers. Please provide your solutions in the form of PDF (we highly appreciate LateX skills) and also working code (wherever asked) in a language of your choice (preferably C++, Matlab, Ruby, Julia or Python1).

Contents

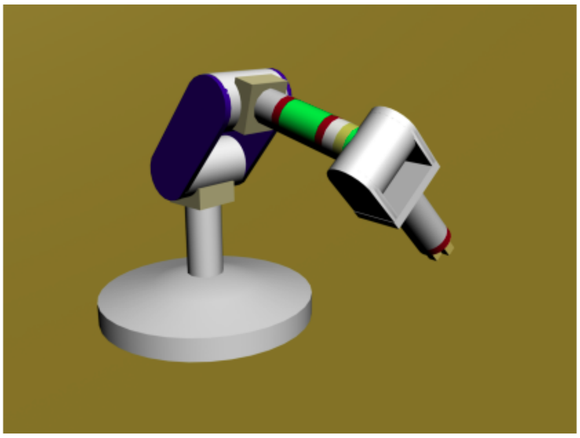
Chapter 2

Mechanics

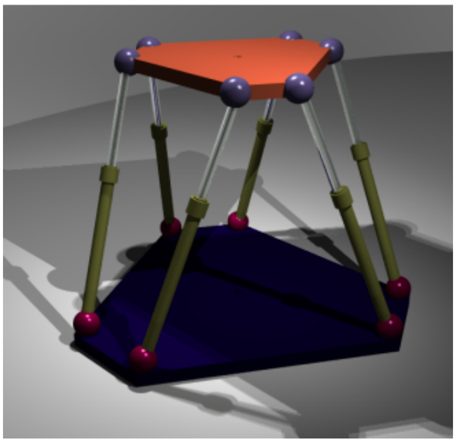
Topology and Mobility Analysis

2.0.1 Exercise 1

In this exercise, we will analyze the topology and mobility of two different robot configurations: a 6-DOF serial robot and a parallel robot (Stewart platform). Figure 2.1 shows a comparison of these two types of robots.



(a) 6-DOF Serial Robot



(b) Parallel Robot

Figure 2.1: Comparison of 6-DOF Serial Robot and Parallel Robot

Let's start by analyzing the 6-DOF serial robot in more detail. Figure 2.2 illustrates the joint axes and links of this robot.

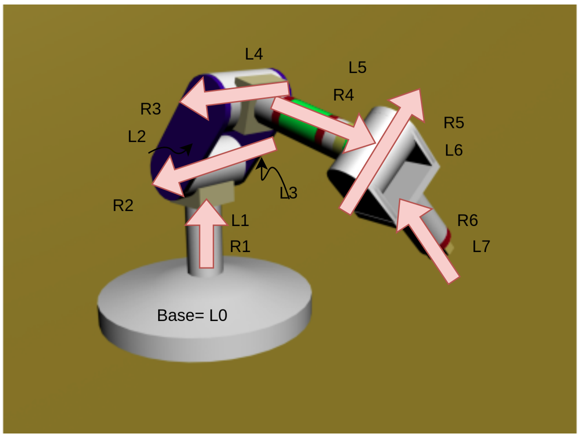


Figure 2.2: Joint axes and links of the 6-DOF serial robot

Solution: 8-Link Serial Robot

The serial robot described in Figure 2.1a and detailed in Figure 2.2 is a typical example of an industrial robotic arm. It consists of the following components:

- **Base (L_0)**:
  - This is the fixed part of the robot, usually mounted on the ground or a stable platform.
  - It serves as the reference frame for the entire robot structure.
- **Seven Moving Links (L_1 to L_7)**:
  - These are the rigid bodies that make up the robot's arm.
  - Each link connects to the next via a joint, forming a chain-like structure.
  - L_7 is typically the end-effector or the link to which a tool or gripper is attached.
- **Revolute Joints (J_0 to J_6)**:

  - These joints connect the links and allow rotational motion between them.
  - Each joint has one degree of freedom, allowing rotation around a single axis.
  - The joints are typically labeled J_0, J_1, J_2, J_3, J_4, J_5, and J_6.
- **Connectivity**:
  - Base (L_0) is connected to L_1 via J_0
  - L_1 is connected to L_2 via J_1 and to L_3 via J_2
  - L_2 is connected to L_4 via J_3
  - L_3 is connected to L_4 via J_4
  - L_4 is connected to L_5 via J_5
  - L_5 is connected to L_6 via J_6
  - L_6 is connected to L_7 via J_7
- **Kinematic Chain**:
  - The robot forms an open kinematic chain, starting from the base and ending at the end-effector.
  - Each joint adds one degree of freedom to the robot.
- **Degrees of Freedom**:
  - With seven revolute joints, this robot has 7 degrees of freedom (7-DOF).
  - This allows the end-effector to achieve any position and orientation within its workspace.

Connectivity Graph

Based on Featherstone's [3] definition, the connectivity graph for this robot represents each link as a node and each joint as an edge connecting the nodes, clearly showing the serial nature of the robot's structure with parallel connections.

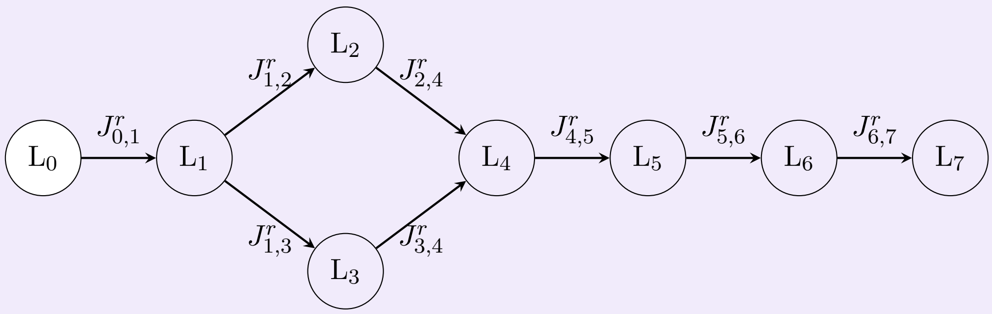


Figure 2.3: Connectivity graph for an 8-link serial robot with parallel connections

In this topological graph:

- Nodes (circles) represent links L_0 to L_7.
- Edges (arrows) represent joints J_0 to J_7.
- The white node (L_0) represents the fixed base.
- The graph clearly shows the serial chain structure of the robot with parallel connections.

Solution: Connectivity graph for the 6-SPS parallel robot (Stewart platform)

Now, let's analyze the connectivity of the parallel robot (Stewart platform) shown in Figure 2.1b. Figure 2.4 illustrates the connectivity graph for this 6-SPS parallel robot.

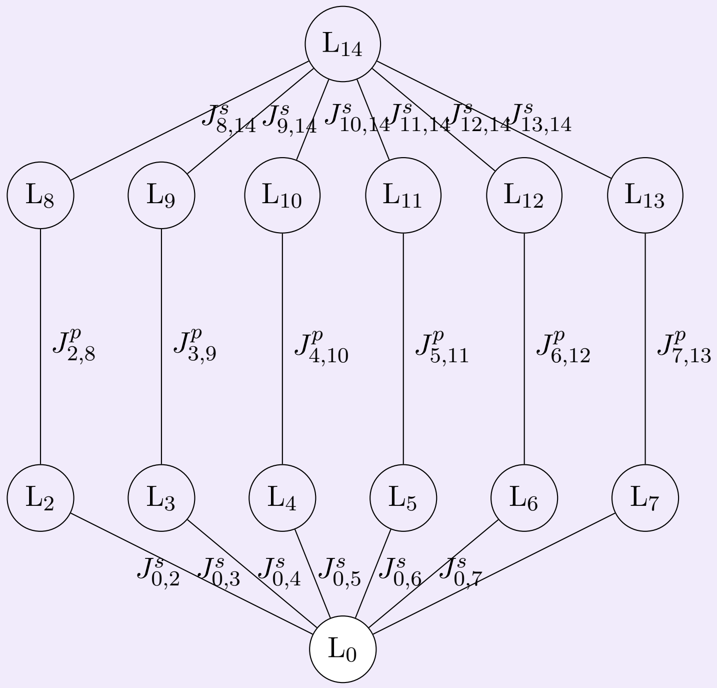


Figure 2.4: Connectivity graph for the 6-SPS parallel robot (Stewart platform)

Connectivity Graph: The graph represents the structure of a 6-SPS (Spherical-Prismatic-Spherical) parallel robot, also known as a Stewart platform.

Components:

- **Base (L_0):** Represented by the white node at the bottom.
- **End-Effector (L_{14}):** Represented by the top node, it's the moving platform.
- **Legs:** Six kinematic chains, each composed of two links:
  - Lower links: L_2 to L_7
  - Upper links: L_8 to L_{13}

Joints:

- **Spherical Joints (S):**
  - Base to lower links: $J^s_{0,i}$ where $i = 2, 3, ..., 7$
  - Upper links to End-Effector: $J^s_{j,14}$ where $j = 8, 9, ..., 13$
- **Prismatic Joints (P):**
  - Between lower and upper links: $J^p_{i,j}$ where $i = 2, 3, ..., 7$ and $j = i + 6$

Structure: Each of the six legs follows an SPS (Spherical-Prismatic-Spherical) configuration, connecting the base to the end-effector. This arrangement provides the robot with 6 degrees of freedom (3 translational and 3 rotational).

2.0.2 Exercise 2

The Chebyshev–Grübler–Kutzbach criterion estimates the degree of freedom (DOF) of a kinematic chain, that is, a coupling of rigid bodies by means of mechanical constraints. The general mobility of a robot can be estimated by the following criteria [9]:

$$F = \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p, \tag{2.1}$$

where

- F – degrees-of-freedom of the mechanism
- λ – degree-of-freedom of the space (= 3 for planar and spherical mechanisms, = 6 for spatial mechanisms)
- n – number of links in the mechanism including the base
- j – number of binary joints of the mechanism
- f_i – degrees of relative motion permitted by joint i
- f_p – total number of passive degrees-of-freedom

Solution:

Let's compute the mobility of the two robots shown in Figure 2.1 using Equation 2.1.

1. 6-DOF Serial Robot

For the serial robot (Figure 2.1a):

- $\lambda = 6$ (spatial mechanism)
- $n = 8$ (7 moving links + 1 base)
- $j = 7$ (7 revolute joints)
- $\sum_{i=1}^{j} f_i = 7$ (1 DOF per revolute joint)
- $f_p = 0$ (no passive degrees-of-freedom)

Applying Equation 2.1:

$$\begin{aligned} F &= \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p \\ &= 6(8 - 7 - 1) + 7 - 0 \\ &= 6(0) + 7 \\ &= 7 \end{aligned}$$

The mobility of the 6-DOF serial robot is 7, which matches our earlier analysis.

2. 6-SPS Parallel Robot (Stewart Platform)

For the parallel robot (Figure 2.1b):

- $\lambda = 6$ (spatial mechanism)
- $n = 14$ (12 leg links + 1 base + 1 platform)
- $j = 18$ (12 spherical joints + 6 prismatic joints)
- $\sum_{i=1}^{j} f_i = 42$ (3 DOF per spherical joint $\times$ 12 + 1 DOF per prismatic joint $\times$ 6)
- $f_p = 6$ (1 passive DOF per leg, rotating around its axis)

Applying Equation 2.1:

$$\begin{aligned} F &= \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p \\ &= 6(14 - 18 - 1) + 42 - 6 \\ &= 6(-5) + 42 - 6 \\ &= -30 + 42 - 6 \\ &= 6 \end{aligned}$$

The calculated mobility of the 6-SPS parallel robot is 6, which is correct.

Simplified Formula for Serial Robots

For serial robots, we can simplify Equation 2.1 because:

- The number of joints is always one less than the number of links ($j = n - 1$)
- There are typically no passive degrees-of-freedom ($f_p = 0$)

Substituting these into Equation 2.1:

$$\begin{aligned} F &= \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p \\ &= \lambda(n - (n - 1) - 1) + \sum_{i=1}^{j} f_i - 0 \\ &= \lambda(0) + \sum_{i=1}^{j} f_i \\ &= \sum_{i=1}^{j} f_i \end{aligned}$$

Therefore, the simplified formula for serial robots is:

$$F = \sum_{i=1}^{j} f_i \tag{2.2}$$

This means that for serial robots, the mobility is simply the sum of the degrees of freedom of all joints.

Application to Parallel Mechanisms

The extended Chebyshev–Grübler–Kutzbach criterion (Equation 2.1) works accurately for parallel mechanisms when we account for passive degrees-of-freedom. In the case of the Stewart platform:

1. Each spherical joint contributes 3 DOF to $\sum_{i=1}^{j} f_i$. 2. Each prismatic joint contributes 1 DOF to $\sum_{i=1}^{j} f_i$. 3. Each leg has 1 passive DOF (rotation around its axis), contributing to f_p.

By including these passive degrees-of-freedom in our calculation, we obtain the correct mobility of 6 for the Stewart platform without needing additional simplifications or corrections.

This demonstrates the importance of considering passive degrees-of-freedom when analyzing the mobility of complex parallel mechanisms.

2.0.3 Exercise 3

It seems almost magical that a simple formula like the Chebychev–Grübler–Kutzbach criterion can be used to estimate the general mobility of a system. Does it always work? Can you find some counter-examples where this formula fails?

Solution:

While the Chebychev–Grübler–Kutzbach criterion is a powerful tool for estimating the mobility of many mechanical systems, it does not always work correctly. This paper [4] addresses most of mechanisms that the mobility criteria failed to predict their DOF. Here we state several situations where the formula can fail to accurately predict the degrees of freedom of a mechanism. Let's explore some of these cases:

1. Overconstrained Mechanisms

Some mechanisms are overconstrained yet still mobile due to special geometric conditions. The Chebychev–Grübler–Kutzbach criterion often predicts zero or negative mobility for these systems, even though they can move.

Example: Bennett's Linkage

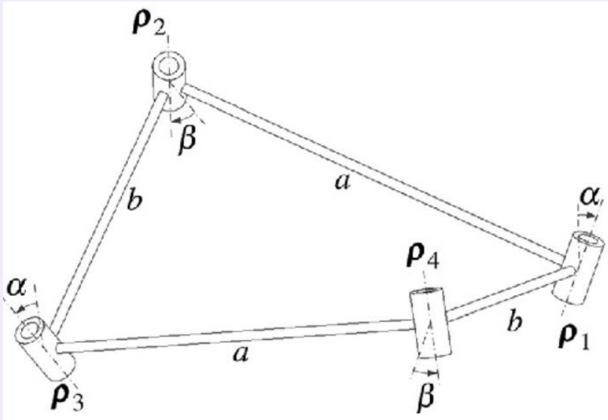


Figure 2.5: Bennett's Linkage [5]

Bennett's linkage is a spatial 4-bar linkage with one degree of freedom. However, applying the Chebychev–Grübler–Kutzbach criterion:

- $\lambda = 6$ (spatial mechanism)
- $n = 4$ (4 links)
- $j = 4$ (4 revolute joints)
- $\sum_{i=1}^{j} f_i = 4$ (1 DOF per revolute joint)
- $f_p = 0$ (no passive DOF)

$$\begin{aligned}
F &= \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p \\
&= 6(4 - 4 - 1) + 4 - 0 \\
&= 6(-1) + 4 \\
&= -2
\end{aligned}$$

The formula predicts -2 DOF, but the mechanism actually has 1 DOF due to its special geometry.

2. Mechanisms with Redundant Constraints

Some mechanisms have redundant constraints that don't affect mobility but are counted in the formula.

Example: Parallel Manipulator with Redundant Actuation Consider a planar parallel manipulator with three legs, each containing an actuated prismatic joint, where only two actuators are needed for full mobility.

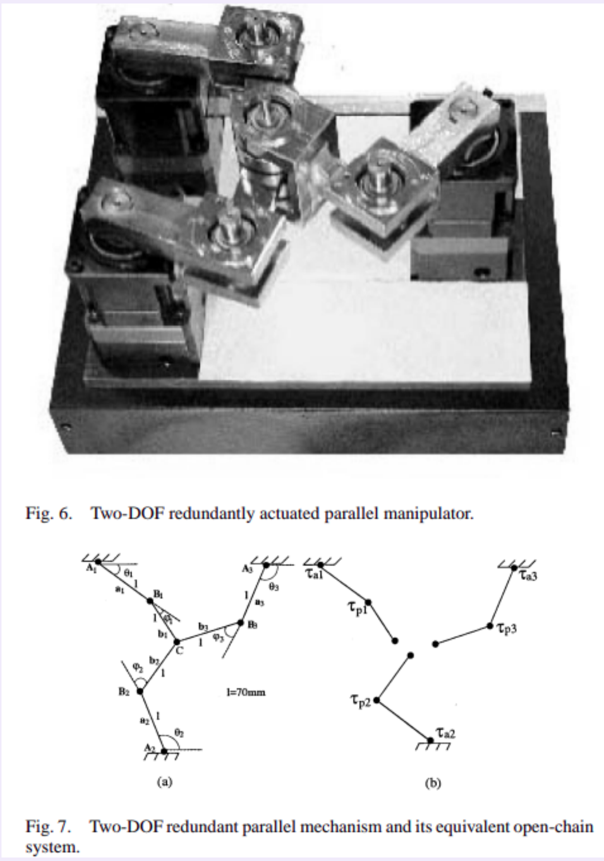


Figure 2.6: Two-DOF redundantly actuated parallel manipulator [2]

Applying the formula:

- $\lambda = 3$ (planar mechanism)
- $n = 8$ (1 base, 1 platform, 6 leg links)
- $j = 9$ (3 prismatic + 6 revolute joints)
- $\sum_{i=1}^{j} f_i = 9$ (1 DOF per joint)
- $f_p = 0$ (no passive DOF)

$$
\begin{aligned}
F &= \lambda(n - j - 1) + \sum_{i=1}^{j} f_i - f_p \\
&= 3(8 - 9 - 1) + 9 - 0 \\
&= 3(-2) + 9 \\
&= 3
\end{aligned}
$$

The formula predicts 3 DOF, which is correct, but it doesn't account for the redundant actuation.

3. Mechanisms with Special Configurations

Some mechanisms can change their mobility in certain configurations, which the formula doesn't capture.

Example: Bricard's Flexible Octahedron [1] : This mechanism can transition between 0 and 1 DOF depending on its configuration, but the Chebychev–Grübler–Kutzbach criterion always predicts the same mobility.

4. Mechanisms with Higher Pair Joints

The formula assumes lower pair joints (e.g., revolute, prismatic). It may not accurately predict mobility for mechanisms with higher pair joints (e.g., cam-follower systems).

Conclusion

While the Chebychev–Grübler–Kutzbach criterion is a valuable tool for initial mobility analysis, it has limitations. It's important to consider:

- Special geometric conditions
- Redundant constraints
- Configuration-dependent mobility
- The nature of the joints involved

In complex or unusual mechanisms, additional analysis methods (e.g., screw theory, instantaneous kinematics) may be necessary to accurately determine mobility.

Geometry and Kinematics

2.0.4 Exercise 4

A serially connected 3 DOF robotic leg with 2 orthogonally intersecting revolute joints and a 1 DOF prismatic joint is shown in Figure 2.7. Given q_1 and q_2 denote the joint angles in the first two revolute joints and q_3 denote the linear displacement in the prismatic joint:

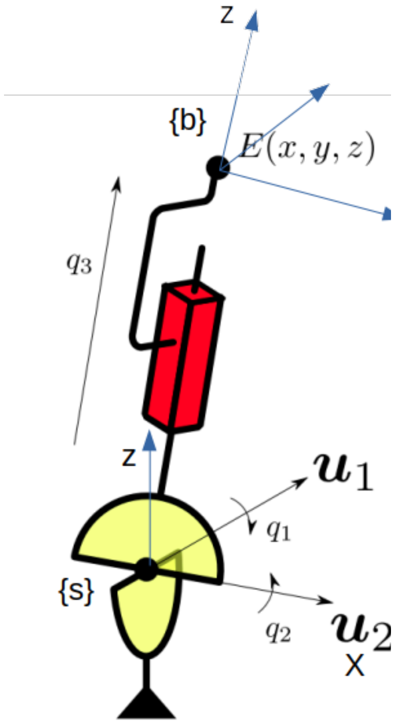


Figure 2.7: A 3 DOF robotic leg with 2 revolute and 1 prismatic joints

Solution:

1. Geometric object where the end-effector point E lives

The end-effector point E lives in a three-dimensional Euclidean space, $\mathbb{R}^3$. More specifically, it traces out a subset of $\mathbb{R}^3$ that forms the workspace of the robot.

2. Forward Kinematics using Product of Exponentials (PoE)

Based on Kevin-Lynch's Modern Robotics book [6] the forward kinematics for our 3 DOF robotic leg can be derived using the Product of Exponentials (PoE) formula. This method offers an elegant and intuitive approach, particularly suitable for open kinematic chains like our robot leg. Let's go through the process step-by-step:

Step 1: Define Frames

- **Space Frame {s}**: Fixed at the base of the robot.
- **Body Frame {b}**: Attached to the end-effector (point E).

Step 2: Zero Configuration

Set all joint variables (q_1, q_2, q_3) to zero. In this configuration, the transformation matrix $M \in SE(3)$ from {s} to {b} is:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Step 3: Determine Screw Axes

For our robot, we need to determine the screw axes for each joint $(\omega_i^T, v_i^T)^T$. The screw axis is represented by two vectors: ω_i (axis of rotation or translation) and v_i (linear velocity). Let's examine how we calculate v_i for each type of joint:

Revolute Joints (Joints 1 and 2): For a revolute joint, v_i is calculated using the formula:

$$v_i = -\omega_i \times q_i \tag{2.4}$$

where q_i is any point on the joint axis. This formula comes from the fact that the linear velocity of any point on a rigid body rotating about an axis is given by the cross product of the angular velocity vector and the position vector from any point on the axis to the point in question.

- **Joint 1:** The axis of rotation is along the y-axis, passing through the origin.

$$\begin{aligned} \omega_1 &= [0, 1, 0]^T \\ q_1 &= [0, 0, 0]^T \text{ (we can choose the origin as our point)} \\ v_1 &= -\omega_1 \times q_1 = -[0, 0, 1]^T \times [0, 0, 0]^T = [0, 0, 0]^T \end{aligned}$$

- **Joint 2:** The axis of rotation is along the x-axis, passing through the origin $[0, 0, 0]$.

$$\begin{aligned} \omega_2 &= [1, 0, 0]^T \\ q_2 &= [0, 0, 0]^T \\ v_2 &= -\omega_2 \times q_2 = -[1, 0, 0]^T \times [0, 0, 0]^T = [0, 0, 0]^T \end{aligned}$$

Prismatic Joint (Joint 3): For a prismatic joint, ω_i is zero (no rotation), and v_i is simply a unit vector in the direction of positive translation.

- **Joint 3:** The prismatic joint moves along the z-axis of the space frame after the second rotation.

$$\begin{aligned} \omega_3 &= [0, 0, 0]^T \\ v_3 &= [0, 0, 1]^T \text{ (unit vector in the direction of translation)} \end{aligned}$$

This calculation of v_i ensures that the screw axis correctly represents the motion of each joint, whether it's a rotation around an axis (revolute) or a translation along an axis (prismatic).

Step 4: Screw Axis Matrices (Extended Explanation)

In the Product of Exponentials (PoE) formula, we represent each joint's motion using a 4x4 matrix $[S_i] \in se(3)$, where $se(3)$ is the Lie algebra of the Special Euclidean group SE(3). This matrix encapsulates both the rotational and translational components of the joint's motion.

The general form of the screw axis matrix $[S_i]$ is:

$$[S_i] = \begin{bmatrix} [\omega_i] & v_i \\ 0 & 0 \end{bmatrix} \tag{2.5}$$

where $[\omega_i]$ is the 3x3 skew-symmetric matrix representation of ω_i, and v_i is the 3x1 linear velocity vector we calculated in Step 3.

For a vector $\omega = [\omega_1, \omega_2, \omega_3]^T$, its skew-symmetric matrix representation is:

$$[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.6}$$

Now, let's calculate $[S_i]$ for each joint:

Joint 1 (Revolute): $\omega_1 = [0, 1, 0]^T$, $v_1 = [0, 0, 0]^T$

$$[S_1] = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.7}$$

Joint 2 (Revolute): $\omega_2 = [1, 0, 0]^T$, $v_2 = [0, 0, 0]^T$

$$[S_2] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.8}$$

Joint 3 (Prismatic): $\omega_3 = [0, 0, 0]^T$, $v_3 = [0, 0, 1]^T$

$$[S_3] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.9}$$

Interpretation: - For revolute joints (1 and 2), the upper-left 3x3 submatrix represents the axis of rotation, while the upper-right 3x1 vector represents the moment of the axis. - For the prismatic joint (3), the upper-left 3x3 submatrix is zero

Step 5: PoE Formula

The forward kinematics is given by:

$$T(q_1, q_2, q_3) = e^{[S_1]q_1} e^{[S_2]q_2} e^{[S_3]q_3} M \tag{2.10}$$

Step 6: Compute Matrix Exponentials

$$e^{[S_1]q_1} = \begin{bmatrix} \cos q_1 & 0 & \sin q_1 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin q_1 & 0 & \cos q_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$e^{[S_2]q_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos q_2 & -\sin q_2 & 0 \\ 0 & \sin q_2 & \cos q_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$e^{[S_3]d} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 7: Final Forward Kinematics

Multiplying these matrices together, we get the final transformation:

$$T(q_1, q_2, q_3) = \begin{bmatrix} \cos q_1 & \sin q_1 \sin q_2 & \sin q_1 \cos q_2 & \cos q_2 \sin q_1 (L_0 + q_3) \\ 0 & \cos q_2 & -\sin q_2 & -\sin q_2 (L_0 + q_3) \\ -\sin q_1 & \cos q_1 \sin q_2 & \cos q_1 \cos q_2 & \cos q_2 \cos q_1 (L_0 + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

The position of the end-effector (x, y, z) can be extracted from the last column of this matrix:

$$\begin{aligned} x &= \cos q_2 \sin q_1 (L_0 + q_3) \\ y &= -\sin q_2 (L_0 + q_3) \\ z &= \cos q_2 \cos q_1 (L_0 + q_3) \end{aligned}$$

This result matches our earlier geometric derivation, validating both approaches.

Advantages of PoE over Denavit-Hartenberg (D-H)

- No need to define individual link frames
- Uniform treatment of revolute and prismatic joints
- More intuitive geometric interpretation via screw axes
- Particularly advantageous for open kinematic chains like our robot leg

The PoE method provides a systematic and elegant approach to deriving forward kinematics, offering both mathematical rigor and geometric intuition.

3. Inverse Kinematics

For this section we utilize Symforce [7] symbolic math package. SymForce is a Python library that blends the flexibility of symbolic mathematics with the speed of optimized code generation for robotics tasks. It excels at creating highly efficient functions for things like computer vision, motion planning, and control, where performance is critical. By allowing you to define a problem symbolically, SymForce automatically generates fast C++ or Python code, eliminating the need for error-prone hand-written derivatives. It shines in situations where you need to perform complex calculations on geometric objects, like rotations and poses, and optimize systems with many variables. SymForce's ability to "flatten" code, exploit sparsity in matrices, and handle singularities without branching makes it significantly faster than traditional automatic differentiation in many robotics applications.

The inverse kinematics for our 3-DOF robotic leg can be solved using symbolic computation. We'll reference the Jupyter notebook code in './Code/E4.ipynb' for this analysis.

Derivation

Using the transformation matrix $T(q_1, q_2, q_3)$, we can extract equations for the end-effector position:

$$
\begin{aligned}
x &= \cos q_2 \sin q_1 (L_0 + q_3) \\
y &= -\sin q_2 (L_0 + q_3) \\
z &= \cos q_2 \cos q_1 (L_0 + q_3)
\end{aligned}
$$

Solving these equations for q_1, q_2, and q_3 yields multiple solution sets due to the nonlinear nature of the equations.

Multiple Solution Sets

The inverse kinematics solution yields 8 distinct solution sets. Each set represents a possible configuration of joint angles (q_1, q_2, q_3) that could position the end-effector at the desired location.

Example Solution Set:

$$
\begin{aligned}
q_1 &\approx 1.81578 \text{ radians} \\
q_2 &\approx 2.37135 \text{ radians} \\
q_3 &\approx 1.87228 \text{ units}
\end{aligned}
$$

Interpretation of Solutions:

- **Real Solutions:** Represent physically achievable robot configurations.
- **Complex Solutions:** Solutions with non-zero imaginary parts are not physically realizable.

Significance of Multiple Solutions

1. **Robot Configuration:** Each solution represents a different arm configuration that achieves the same end-effector position. 2. **Path Planning:** Multiple solutions allow for

choosing optimal configurations based on factors like joint limits, obstacle avoidance, etc. 3. **Singularities:** Some solutions might be near singularities, which should be avoided in practical applications.

Handling Solutions

1. **Filtering:** We discard solutions with significant imaginary parts. 2. **Thresholding:** We set a small threshold (e.g., 10^{-9}) to consider solutions as real if imaginary parts are below this value. 3. **Physical Constraints:** We apply joint limits and workspace constraints to further filter solutions.

In practice, additional criteria such as joint limits, singularity avoidance, and obstacle avoidance would be used to select the most appropriate solution from the multiple possibilities.

4. Verification of Forward and Inverse Kinematics

For the verification of forward and inverse kinematics, we refer to the Jupyter notebook '/Code/E4.ipynb'. In the section titled "Verifying Forward Kinematics", the notebook provides a comprehensive verification process.

The notebook defines a function `verify_forward_kinematics` that performs the following steps:

1. **Input:** Joint angles q_1, q_2, q_3, and expected end-effector position $(x_{exp}, y_{exp}, z_{exp})$
2. **Compute:** Actual end-effector position (x, y, z) using forward kinematics
3. **Display:**
   - Current joint configuration
   - Expected end-effector position
   - Actual end-effector position
4. **Compare:** Expected and actual positions within a small tolerance
5. **Output:** Whether the positions match or not

Four test cases are implemented for different robot configurations:

- Robot arm rotates around q_1 for 90 degrees
- Robot arm straight up $q_1 = 0$ $q_2 = 0$
- Robot arm bent 90 degrees around $q_2 = 90$
- Robot arm straight up with extension $q_1 = 0$ $q_2 = 0$ $q_3 = 0.5$

For each test case, the notebook displays the configuration, expected position, actual position, and whether they match within a small tolerance. This verification process helps ensure that the forward kinematics calculations are correct for various robot poses.

It's important to note that since we have successfully verified the forward kinematics, and the inverse kinematics are derived from the forward kinematics equations, we do not need a separate verification for the inverse kinematics. The correctness of the forward kinematics implies the correctness of the inverse kinematics, as they are mathematically inverse operations of each other.

For the complete verification process and code implementation, please refer to the Jupyter notebook '/Code/E4.ipynb'.

5. Workspace Analysis

The workspace analysis for our 3-DOF robot is conducted through a comprehensive approach, as detailed in the Jupyter notebook `E4.ipynb`. This analysis provides crucial insights into the robot's operational capabilities and limitations. The key components of this analysis, as implemented in `E4.ipynb`, are:

Robot Setup and Forward Kinematics

In `E4.ipynb`, the robot's configuration, including joint angles and screw axes, is defined. The notebook then calculates forward kinematics to determine the end-effector's position for given joint configurations.

Jacobian and Singularity Analysis

The `E4.ipynb` notebook computes the Jacobian matrix, which relates joint velocities to end-effector velocities. This is used to identify singular configurations through Singular Value Decomposition (SVD). The workspace is sampled by iterating through ranges of joint angles, classifying points as either regular or singular.

Inverse Kinematics Validation

Using the inverse kinematics solutions calculated earlier in `E4.ipynb`, the notebook analyzes the robot's workspace. It includes a function that filters out complex solutions, retaining only real, physically meaningful joint configurations. The workspace is sampled in Cartesian space, and each point is checked for valid inverse kinematics solutions.

Workspace Visualization

The `E4.ipynb` notebook creates 3D scatter plots to visualize:

- Regular vs. Singular points
- Valid vs. Invalid points (based on inverse kinematics)

These visualizations provide an intuitive understanding of the robot's operational space and its limitations.

Quantitative Analysis

For both singularity and inverse kinematics analyses, `E4.ipynb` calculates:

- Total number of sampled points
- Number of regular/valid points
- Number of singular/invalid points
- Percentage of singular configurations or valid workspace

The notebook also determines the boundaries of the valid workspace and the center of singular or invalid regions (if applicable).

Interpretation of Results

The comprehensive analysis in `E4.ipynb` offers insights into:

- The robot's reachable workspace
- Regions prone to singularities
- Limitations in the robot's design or configuration
- Potential areas for improvement or optimization

By combining forward kinematics, Jacobian analysis, and inverse kinematics validation, the analysis in `E4.ipynb` provides a holistic view of the robot's workspace characteristics. This is essential for effective task planning, identifying potential control issues, optimizing the robot's design, and ensuring safe and efficient operation within the robot's workspace.

For detailed implementation and results, please refer to the Jupyter notebook `E4.ipynb`.

2.0.5 Exercise 5: Slider-Crank Linkage Analysis

Consider a slider-crank linkage of type 1-RRPR shown in Figure 3, which involves a linear actuator and three revolute joints. Let q_1 denote the output joint angle and q_3 denote the input prismatic joint displacement.

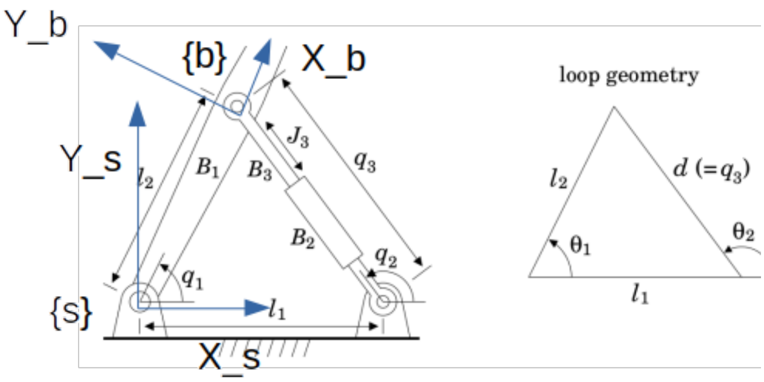


Figure 2.8: Slider-crank linkage (1-RRPR)

Solution:

General Approach to Parallel Mechanisms

Before addressing the specific slider-crank linkage, let's discuss a general approach [6, 8] to analyzing parallel mechanisms. This method can be applied to a wide range of parallel mechanisms, including the one in our problem.

General Parallel Mechanisms

Consider a parallel mechanism where the fixed and moving platforms are connected by multiple open chains, as illustrated in Figure 2.9.

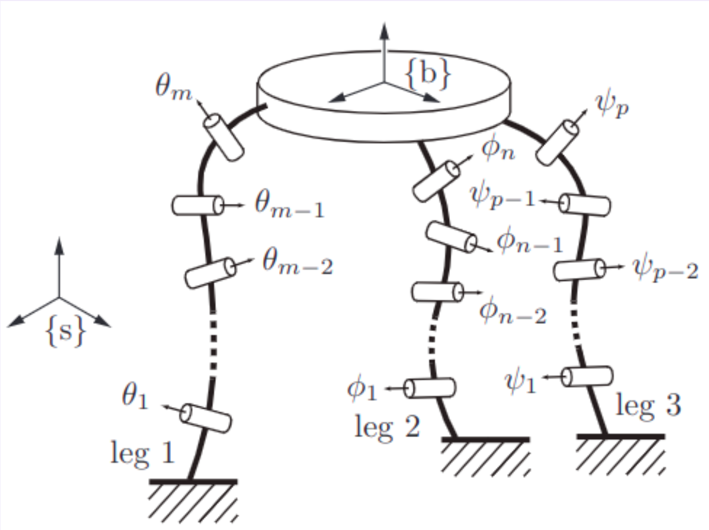


Figure 2.9: General Parallel Mechanism with Multiple Open Chains [6]

Kinematic Formulation

Let the configuration of the moving platform be given by $T_{sb} \in SE(3)$. Denote the forward kinematics of each chain by $T_i(q_i)$, where q_i represents the joint variables for the i-th chain.

Loop-Closure Conditions

The loop-closure conditions can be written as:

$$T_{sb} = T_1(q_1) = T_2(q_2) = \cdots = T_n(q_n) \tag{2.12}$$

Constraint Equations

By equating the transformations, we obtain a set of constraint equations. The number of independent equations determines the degrees of freedom of the mechanism.

Forward and Inverse Kinematics

The forward and inverse kinematics problems involve solving these constraint equations for unknown joint variables or the platform configuration, respectively.

Application to the Slider-Crank Linkage

Now, let's apply this general approach to the specific slider-crank linkage of type 1-RRPR shown in Figure 3.

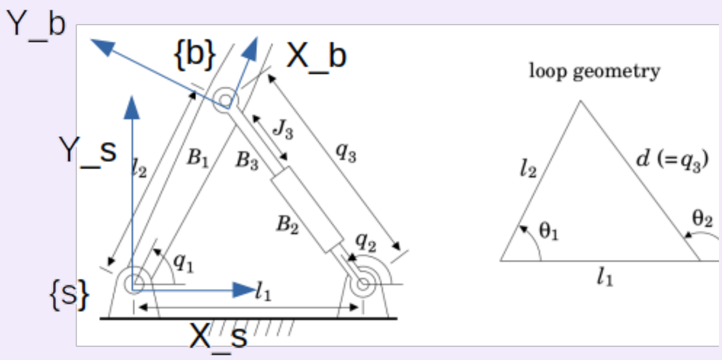


Figure 2.10: Slider-crank linkage (1-RRPR)

Forward Kinematics Analysis

The forward kinematics of the slider-crank linkage (1-RRPR) was analyzed using a symbolic computation approach. The analysis yielded the following results:

Method: The mechanism was modeled using two kinematic chains and a loop-closure condition was applied to ensure that both chains meet at the same point.

Results: The forward kinematics solution provides two possible configurations for the output joint angle q_1 in terms of the input prismatic joint displacement q_3:

$$q_1 = \begin{cases} 2\arctan\left(\sqrt{\frac{l_1^2+2l_1l_2+l_2^2-q_3^2}{-l_1^2+2l_1l_2-l_2^2+q_3^2}}\right) \\ -2\arctan\left(\sqrt{\frac{-l_1^2+2l_1l_2+l_2^2-q_3^2}{l_1^2-2l_1l_2+l_2^2-q_3^2}}\right) \end{cases} \tag{2.13}$$

where:

- q_1 is the output joint angle (revolute)
- q_3 is the input prismatic joint displacement
- l_1 and l_2 are the lengths of the first and second links, respectively

Interpretation:

- The two solutions represent the two possible configurations of the mechanism for a given input q_3.
- The first solution corresponds to the "elbow-up" configuration.
- The second solution corresponds to the "elbow-down" configuration.
- The solutions involve the ratio of differences of squared lengths, reflecting the geometric constraints of the mechanism.

Limitations:

- The expressions under the square roots must be non-negative for real solutions to exist. This condition defines the feasible workspace of the mechanism.
- The solution assumes $l_1^2 - 2l_1l_2 + l_2^2 - q_3^2 \neq 0$ and $-l_1^2 + 2l_1l_2 - l_2^2 + q_3^2 \neq 0$. When these conditions are not met, the mechanism is in a singular configuration.
- The arctan function should be implemented as atan2 in most programming languages to handle all quadrants correctly.

Geometric Interpretation: The solutions can be interpreted geometrically:

- The numerator $l_1^2 + 2l_1l_2 + l_2^2 - q_3^2$ represents the square of the sum of the link lengths minus the square of the input displacement.
- The denominator $-l_1^2 + 2l_1l_2 - l_2^2 + q_3^2$ represents the square of the difference of the link lengths minus the square of the input displacement.
- The ratio of these terms, when square-rooted, gives the tangent of half the output angle, which is then doubled to get the full angle.

This forward kinematics solution provides a comprehensive understanding of the slider-crank mechanism's behavior, allowing for the prediction of the output angle given the input displacement. The form of the solution highlights the nonlinear nature of the mechanism's kinematics and the importance of considering multiple solutions in parallel mechanisms.

Detailed Calculations: The complete derivation and step-by-step calculations for the forward kinematics can be found in the Jupyter notebook `E5.ipynb` located in the Code folder. This notebook provides a comprehensive breakdown of the symbolic computations, including the definition of the kinematic chains, application of the loop-closure condition, and the solution process for obtaining these forward kinematics expressions.

2. Inverse Kinematics

The inverse kinematics of the slider-crank linkage (1-RRPR) was analyzed using python's symbolic computation with Symforce [7]

Detailed Calculations: The complete derivation and step-by-step calculations for the forward kinematics can be found in the Jupyter notebook `E5.ipynb` located in the Code folder. This notebook provides a comprehensive breakdown of the symbolic computations, including the definition of the kinematic chains, application of the loop-closure condition, and the solution process for obtaining the forward kinematics expressions. The analysis yielded the following results:

Method: The inverse kinematics problem involves finding the input prismatic joint displacement q_3 given the output revolute joint angle q_1. The solution was derived using the loop-closure condition and solving the resulting equations.

Results: The inverse kinematics solution provides two possible configurations for the input prismatic joint displacement q_3 in terms of the output joint angle q_1:

$$q_3 = \begin{cases} -\frac{l_2 \sin(q_1)}{\sin(2\alpha_1)} \\ \frac{l_2 \sin(q_1)}{\sin(2\alpha_2)} \end{cases} \tag{2.14}$$

where:

$$\alpha_1 = \arctan\left(-\frac{l_1}{\sin(q_1)l_2} + \frac{1}{\tan(q_1)} + \frac{\sqrt{l_1^2 - 2l_1l_2\cos(q_1) + l_2^2}}{l_2\sin(q_1)}\right)$$

$$\alpha_2 = \arctan\left(-\frac{l_1}{\sin(q_1)l_2} - \frac{1}{\tan(q_1)} + \frac{\sqrt{l_1^2 - 2l_1l_2\cos(q_1) + l_2^2}}{l_2\sin(q_1)}\right)$$

and:

- q_1 is the output joint angle (revolute)
- q_3 is the input prismatic joint displacement
- l_1 and l_2 are the lengths of the first and second links, respectively

Interpretation:

- The two solutions represent the two possible configurations of the mechanism for a given output angle q_1.
- The solutions involve complex trigonometric expressions, reflecting the nonlinear nature of the mechanism's kinematics.
- The α_1 and α_2 terms represent intermediate angles in the mechanism's configuration.
- The solution depends on the ratio of trigonometric functions of these intermediate angles to $l_2 \sin(q_1)$, which relates to the mechanism's geometry.

Limitations and Considerations:

- The solution assumes $\sin(q_1) \neq 0$. When $\sin(q_1) = 0$ (i.e., $q_1 = 0$ or π), the mechanism is in a singular configuration.
- The expressions under the square roots must be non-negative for real solutions to exist. This condition defines the feasible workspace of the mechanism.
- The arctan function should be implemented as atan2 in most programming languages to handle all quadrants correctly.
- Numerical stability should be considered when implementing these solutions, especially near singular configurations.

4. Maximum Output Angular Velocity

To find an expression for the maximum output angular velocity $\dot{q}_{1max}$ given the maximum velocity $\dot{q}_{3max}$ available at the actuator, we used symbolic computation in the Jupyter notebook E5.ipynb. The process and results are as follows:

Method:

- We defined time-dependent variables for q_1, q_2, q_3, and l_2.
- Using the forward kinematics solutions, we calculated the Jacobians $\frac{\partial q_1}{\partial q_3}$ for both solutions.
- The maximum output angular velocity was then calculated as $\dot{q}_{1max} = |\frac{\partial q_1}{\partial q_3}| \cdot \dot{q}_{3max}$.

Results: The symbolic computation yielded the following expressions for $\dot{q}_{1max}$:

For both solutions:

$$\dot{q}_{1max} = 2\dot{q}_{3max} \left| \frac{q_3(t) \cdot \sqrt{\frac{l_1^2 - 2l_1l_2(t) + l_2^2(t) - q_3^2(t)}{l_1^2 + 2l_1l_2(t) + l_2^2(t) - q_3^2(t)}}}{l_1^2 - 2l_1l_2(t) + l_2^2(t) - q_3^2(t)} \right| \tag{2.15}$$

Interpretation:

- The expression for $\dot{q}_{1max}$ is the same for both solutions of the forward kinematics, indicating that the maximum output angular velocity is independent of which configuration the mechanism is in.
- The maximum output angular velocity depends on the current state of the mechanism (represented by $q_3(t)$ and $l_2(t)$) and the geometric parameters (l_1).
- The expression involves ratios of quadratic terms in $q_3(t)$ and $l_2(t)$, reflecting the nonlinear nature of the mechanism's kinematics.
- The absolute value in the expression ensures that $\dot{q}_{1max}$ is always positive, as it represents a magnitude.

Limitations and Considerations:

- The expression becomes undefined when $l_1^2 - 2l_1l_2(t) + l_2^2(t) - q_3^2(t) = 0$, which corresponds to a singular configuration of the mechanism.
- The time dependence of l_2 ($l_2(t)$) in the derived expression suggests that the analysis considered potential variations in link lengths, which might not be applicable in a rigid mechanism. For a fixed-length mechanism, l_2 would be constant.
- The actual maximum velocity achievable may be lower due to physical constraints not captured in this kinematic analysis, such as joint limits or dynamic effects.

Detailed Calculations: The complete derivation, including the symbolic manipulation and intermediate steps, can be found in the Jupyter notebook E5.ipynb in the Code folder. This notebook provides a step-by-step breakdown of the calculation process using the SymForce library for symbolic mathematics.

5. Maximum Output Torque

Given that the maximum force available at the actuator (prismatic joint q3) is f_{max}, we need to find an expression for the maximum output torque τ_{max} at the revolute joint q1.

To derive this relationship, we use the principle of virtual work and the Jacobian of the mechanism.

Method:

- Let J be the Jacobian that relates the velocity of q3 to the angular velocity of q1: $\dot{q}_1 = J\dot{q}_3$
- By the principle of virtual work, we have: $\tau_1 \delta q_1 = f_3 \delta q_3$
- Substituting $\delta q_1 = J\delta q_3$, we get: $\tau_1 J \delta q_3 = f_3 \delta q_3$
- This implies: $\tau_1 = \frac{f_3}{J}$

Expression for Maximum Output Torque: Given the maximum force f_{max} at the actuator, the maximum output torque is:

$$\tau_{max} = \frac{f_{max}}{J} = f_{max} \cdot \frac{1}{J} \tag{2.16}$$

where J is the Jacobian we derived earlier (2.15):

$$J = 2\left|\frac{q_3(t) \cdot \sqrt{\frac{l_1^2 - 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}{l_1^2 + 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}}}{l_1^2 - 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}\right| \tag{2.17}$$

Substituting this into our expression for τ_{max}:

$$\tau_{max} = 2f_{max} \cdot \left|\frac{l_1^2 - 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}{q_3(t) \cdot \sqrt{\frac{l_1^2 - 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}{l_1^2 + 2l_1 l_2(t) + l_2^2(t) - q_3^2(t)}}}\right| \tag{2.18}$$

Interpretation:

- The maximum output torque τ_{max} is directly proportional to the maximum force f_{max} available at the actuator.
- τ_{max} depends on the current state of the mechanism (represented by $q_3(t)$ and $l_2(t)$) and the geometric parameters (l_1).
- The expression involves ratios of quadratic terms in $q_3(t)$ and $l_2(t)$, reflecting the nonlinear nature of the mechanism's kinematics.
- τ_{max} is inversely proportional to the Jacobian, meaning it increases when the mechanism is in configurations that provide mechanical advantage.

Limitations and Considerations:

- This analysis assumes static conditions and doesn't account for dynamic effects.
- The expression becomes undefined when $l_1^2 - 2l_1l_2(t) + l_2^2(t) - q_3^2(t) = 0$, which corresponds to a singular configuration of the mechanism.
- The time dependence of l_2 ($l_2(t)$) in the derived expression suggests that the analysis considered potential variations in link lengths, which might not be applicable in a rigid mechanism. For a fixed-length mechanism, l_2 would be constant.
- In practice, joint limits and other mechanical constraints may prevent reaching the theoretical maximum torque in certain configurations.

The detailed derivation and symbolic manipulation can be found in the Jupyter notebook `E5.ipynb` in the Code folder.

2.0.6 6. singular configurations

Here we will Identify any singular configurations of the slider-crank mechanism.

Solution:

The slider-crank mechanism (1-RRPR) has several singular configurations. These configurations can be identified by analyzing the Jacobian of the mechanism and finding positions where it becomes rank-deficient. The singular configurations are:

1. **Fully Extended Configuration:**
   - When $q_1 = 0$ or π
   - In this configuration, the two links are fully extended or folded
   - The mechanism loses the ability to transmit force/motion in the direction perpendicular to the links
2. **Maximum Extension of Prismatic Joint:**
   - When $q_3 = l_1 + l_2$
   - The prismatic joint is at its maximum extension
   - The mechanism loses the ability to extend further
3. **Minimum Extension of Prismatic Joint:**
   - When $q_3 = |l_1 - l_2|$
   - The prismatic joint is at its minimum extension
   - The mechanism loses the ability to contract further

In these configurations, the mechanism loses one or more degrees of freedom, and the relationship between input and output velocities becomes indeterminate. These singularities are important to consider in the design and control of the mechanism, as they can lead to loss of control or excessive forces in the joints.

Note: The Jacobian analysis leading to the identification of these singular configurations can be found in the Jupyter notebook `E5.ipynb` in the Code folder.

2.0.7 Exercise 6

Donald is from the United States of America and Angela comes from Europe. They have to work together on a robotics project and they don't always agree with each other. While Donald is obsessed with the Imperial system of units, Angela likes to use the modern metric system. They are together working on a 6 DOF robotic manipulator (with 3 translational and 3 rotational DOFs) mounted on a 3 DOF gantry crane which can place the base of the robot in any 3D position $(x_B, y_B, z_B) \in \mathbb{R}^3$. Angela develops the software for the robot manipulator and Donald develops the software for the 3 DOF gantry crane. They are exchanging black-box models and are not allowed to modify each other's code.

Solution:

1. Converting meters to inches for the gantry crane

Yes, Angela can find a linear scaling factor to convert the 3D position from meters to inches.

- The conversion factor from meters to inches is: 1 meter = 39.3701 inches
- This is a constant scalar value that can be applied uniformly to all three dimensions
- The conversion can be represented as a linear transformation:

$$\begin{bmatrix} x_{\text{inches}} \\ y_{\text{inches}} \\ z_{\text{inches}} \end{bmatrix} = 39.3701 \cdot \begin{bmatrix} x_{\text{meters}} \\ y_{\text{meters}} \\ z_{\text{meters}} \end{bmatrix} \tag{2.19}$$

This linear scaling preserves the proportions and relationships in the 3D space, making it suitable for use with Donald's model.

2. Converting metric to imperial for the robot manipulator

Donald cannot use a single linear scaling factor to convert the 6D pose from the metric system to the imperial system.

- For the translational part (x_E, y_E, z_E), a linear scaling factor (39.3701) can be used to convert from meters to inches, similar to part 1.
- However, for the rotational part:
  - Quaternions cannot be directly converted to Euler angles (roll, pitch, yaw) using a linear scaling factor.
  - The conversion from quaternions to Euler angles involves nonlinear trigonometric functions.
  - Even after converting to Euler angles, no scaling is needed as angles are dimensionless (radians to degrees conversion is a simple multiplication by $180/\pi$, but this is not related to the metric-imperial conversion).

Therefore, a single linear scaling factor cannot be applied to the entire 6D pose to achieve the desired conversion.

3. Using each other's models

Yes, it is possible for Donald and Angela to use each other's models with some additional steps:

- For Donald to use Angela's inverse kinematics model:
  - Convert translational inputs from inches to meters: multiply by 1/39.3701
  - Convert rotational inputs from Euler angles (degrees) to quaternions
  - Apply Angela's model
  - No conversion needed for the output (actuator commands)
- For Angela to use Donald's gantry crane model:
  - Convert inputs from meters to inches: multiply by 39.3701
  - Apply Donald's model
  - No conversion needed for the output (actuator commands)

These conversions can be implemented as wrapper functions around the black-box models, allowing Donald and Angela to work in their preferred units while still utilizing each other's software components effectively.

Dynamics

2.0.8 Exercise 7

Consider a pendulum robot with a bob of mass m connected to the ground link by a massless link of length l in Figure 4. Denote with θ the angle made by the bob with the vertical axis in the anti-clockwise sense.

Solution:

1. Inverse Dynamics

The expression for torque acting at the revolute joint (τ) in terms of (θ, $\dot{\theta}$, $\ddot{\theta}$) is:

$$\tau = ml^2\ddot{\theta} + mgl\sin(\theta) \tag{2.20}$$

where:

- m is the mass of the bob
- l is the length of the link
- g is the acceleration due to gravity
- θ, $\dot{\theta}$, and $\ddot{\theta}$ are the angular position, velocity, and acceleration respectively

This equation accounts for both the inertial torque ($ml^2\ddot{\theta}$) and the gravitational torque ($mgl\sin(\theta)$).

2. Forward Dynamics

The expression for acceleration at the revolute joint ($\ddot{\theta}$) in terms of (τ) is:

$$\ddot{\theta} = \frac{\tau - mgl\sin(\theta)}{ml^2} \tag{2.21}$$

This equation is derived by solving the inverse dynamics equation for $\ddot{\theta}$.

3. Simulation Program

To simulate the free-fall of the pendulum (i.e., $\tau = 0.0$), we can use a simple Newton-Euler numerical integration scheme. Here's a Python program that implements this simulation and provides a visualization:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Parameters
m = 1.0  # mass of bob (kg)
l = 1.0  # length of pendulum (m)
g = 9.81  # acceleration due to gravity (m/s^2)
dt = 0.01  # time step (s)
t_max = 10.0  # maximum simulation time (s)

# Initial conditions
theta = np.pi/4  # initial angle (radians)
omega = 0.0  # initial angular velocity (rad/s)

# Lists to store simulation data
t_list = [0]
theta_list = [theta]

# Simulation loop
t = 0
while t < t_max:
    # Calculate acceleration
    alpha = -g/l * np.sin(theta)

    # Update angular velocity and position
    omega += alpha * dt
    theta += omega * dt

    # Store results
    t += dt
```

```
    t_list.append(t)
    theta_list.append(theta)

# Convert lists to numpy arrays
t_array = np.array(t_list)
theta_array = np.array(theta_list)

# Calculate x and y positions of the bob
x = l * np.sin(theta_array)
y = -l * np.cos(theta_array)

# Create animation
fig, ax = plt.subplots()
ax.set_xlim(-l-0.1, l+0.1)
ax.set_ylim(-l-0.1, l+0.1)
line, = ax.plot([], [], 'o-', lw=2)

def init():
    line.set_data([], [])
    return line,

def animate(i):
    line.set_data([0, x[i]], [0, y[i]])
    return line,

anim = FuncAnimation(fig, animate, init_func=init,
                     frames=len(t_array), interval=dt*1000, blit=True)

plt.show()
```

This program uses the forward dynamics equation with $\tau = 0$ to simulate the free-fall of the pendulum. It employs a simple Euler integration method to update the pendulum's state over time. The resulting animation shows the pendulum swinging back and forth under the influence of gravity.

Key components of the simulation:

- The acceleration is calculated using $\ddot{\theta} = -\frac{g}{l}\sin(\theta)$ (derived from setting $\tau = 0$ in the forward dynamics equation).
- The angular velocity and position are updated using Euler integration.
- The position of the bob is calculated in Cartesian coordinates for visualization.
- Matplotlib's FuncAnimation is used to create an animation of the pendulum's motion.

This simulation provides a visual representation of the pendulum's behavior, allowing for intuitive understanding of its dynamics under free-fall conditions.

2.0.9 Exercise 8

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Solution:

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

2.0.10 Exercise 9

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Solution:

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Contents

CHAPTER 3

Control

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Exercise 10

Exercise 11

Exercise 12

Contents

Chapter 4

Questionnaire

Question 1

During your studies, please select if you have taken any formal/related courses in:

- Mechanics or Applied Mechanics
- Mechanism Theory
- Machine Design
- Rigid Body Mechanics or Multi-Body Dynamics
- Modeling and Control of Robot Manipulators
- Humanoid Robotics
- Biomechanics
- Linear Control Theory
- Non-Linear Control Theory
- Introduction to Robotics
- Artificial Intelligence or Machine Learning
- Linear Algebra
- Advanced Calculus

- Probability Theory
- Object Oriented Programming

Additionally, mention your grade along with maximum possible grade in the applicable subjects.

Question 2

How do you find the overall difficulty level of the exercises? Categorize the exercises.

- Easy
- Medium
- Hard

Question 3

What kind of additional help did you seek while solving the exercises?

- Textbooks
- Wikipedia
- Research papers
- Other online sources
- Friends/Colleagues

Question 4

Please categorize your attempt to solve the exercises into:

- Solved without any external reference
- Solved with the help of known textbooks
- Solved with the help of online content (Wikipedia, online blogs etc.)
- Solved after reading research papers

List the exercise numbers in front of each option above.

Question 5

Please select the programming languages where you have a working knowledge.

- C/C++
- Python
- Matlab
- Julia
- Ruby
- Mathematica

Question 6

Please specify if you have worked with any symbolic manipulation packages.

- Matlab Symbolic Toolbox
- Maple
- Singular
- Sympy

Question 7

Have you worked with any industrial robot platforms?

- Universal robots
- KUKA robots
- St¨aubli robots
- PUMA 560
- Others

Question 8

Have you ever built your own robot as a hobby or group project at the University? If yes, describe your robot and your role in the project.

Question 9

Have you worked with a version control system? If yes, which one?

- Git
- SVN

Question 10

Do you have LateX skills?

Bibliography

[1] J Eddie Baker. An analysis of the bricard linkages. *Mechanism and machine Theory*, 15(4):267–286, 1980.

[2] Hui Cheng, Yiu-Kuen Yiu, and Zexiang Li. Dynamics and control of redundantly actuated parallel manipulators. *IEEE/ASME Transactions on mechatronics*, 8(4):483–491, 2003.

[3] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[4] Grigore Gogu. Chebychev–grübler–kutzbach's criterion for mobility calculation of multi-loop mechanisms revisited via theory of linear transformations. *European Journal of Mechanics-A/Solids*, 24(3):427–441, 2005.

[5] Shengnan Lu, Dimiter Zlatanov, and Xilun Ding. Approximation of cylindrical surfaces with deployable bennett networks. *Journal of Mechanisms and Robotics*, 9(2):021001, 2017.

[6] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[7] Hayk Martiros, Aaron Miller, Nathan Bucki, Bradley Solliday, Ryan Kennedy, Jack Zhu, Tung Dang, Dominic Pattison, Harrison Zheng, Teo Tomic, et al. Symforce: Symbolic computation and code generation for robotics. *arXiv preprint arXiv:2204.07889*, 2022.

[8] Tao Sun, Binbin Lian, Shuofei Yang, and Yimin Song. Kinematic calibration of serial and parallel robots based on finite and instantaneous screw theory. *IEEE Transactions on Robotics*, 36(3):816–834, 2020.

[9] Hamid D Taghirad. *Parallel robots: mechanics and control*. CRC press, 2013.