EE 521:
Kinematics and Dynamics of Machines

Term Project:
Kinematic and Dynamic Analysis of the 3RRP Mechanism

Yunus Emre Danabaş
(yunusdanabas@sabanciuniv.edu - 29359)

January 12, 2025

Sabancı Üniversitesi

Abstract

This report presents a comprehensive study of the 3RRP mechanism's kinematics and dynamics. The primary objectives include deriving closed-form forward and inverse kinematics, computing the largest symmetric workspace with full rotational capabilities, evaluating the kinematic Jacobian, and establishing a Global Isotropy Index (GII) to quantify isotropy within the mechanism's workspace.

To formulate the dynamic equations of motion, both Kane's and Lagrange's methods are employed. Symbolic derivations leverage *Autolev* for partial velocity computations and constraint management, while *MATLAB/Simulink* implementations verify correctness through numerical simulations. Comparative analyses highlight that Kane's approach often yields compact symbolic expressions and inherently manages loop-closure constraints, whereas the Lagrangian framework offers an energy-based perspective but requires explicit constraint enforcement and stabilization (Baumgarte).

Simulation results demonstrate stable end-effector trajectories under small external loads, corroborating the validity of the kinematic and dynamic models. The study concludes with discussions on the mechanism's suitability for planar tasks demanding precise positioning, recommendations for refining models to incorporate non-ideal effects such as friction, and potential directions for advanced control strategies and real-world implementation.

Contents

1 Introduction **3**

2 Problem Definition **3**
2.1 System Overview . . . 3
2.2 Mechanism Description . . . 4
2.3 Scope of Analysis . . . 5

3 Analytical Derivations and Results **5**
3.1 Kinematic Analysis (3RRP Focus) . . . 5
3.1.1 Symbolic Derivations . . . 5
3.1.2 Workspace Calculation . . . 6
3.1.3 Kinematic Jacobian . . . 7
3.1.4 Global Isotropy Index (GII) . . . 8
3.2 Dynamic Modeling . . . 9
3.2.1 System Setup in Autolev . . . 9
3.2.2 Kane's Method . . . 11
3.2.3 Lagrangian Formulation in Autolev with Baumgarte Stabilization . . . 12
3.2.4 Method Comparison . . . 14
3.2.5 Simulation and Validation Using Kane's Dynamic Code . . . 16
3.2.6 Kinematic Simulation in Simulink . . . 19

4 Discussion **21**
4.1 Overview of Key Findings . . . 21
4.2 Analysis of Kinematic Results . . . 21
4.3 Interpretation of Dynamic Modeling Outcomes . . . 22
4.4 Comparisons and Correlations . . . 22
4.5 Practical Implications for 3RRP Mechanism . . . 22
4.6 Reflections and Future Directions . . . 22
4.7 Summary of the Discussion . . . 22

5 Conclusion **23**

1 Introduction

The study of kinematics and dynamics in robotic mechanisms is crucial for designing reliable and high-performance machines. Complex mechanisms such as the 3RRP (three revolute-revolute-prismatic) mechanism, the Linear Delta mechanism, and their combined systems find extensive applications in robotics and automation. Analyzing these systems demands robust mathematical modeling, rigorous derivations, and practical implementation strategies to ensure accurate control and simulation.

This project aims to perform a comprehensive analysis of these mechanisms, focusing primarily on the 3RRP mechanism in Part A, with initial exploratory work on the Linear Delta mechanism in Part B, and considerations for combining both systems. The objectives include deriving the kinematic equations, computing the workspace, evaluating the Jacobian and Global Isotropy Index (GII), formulating dynamic equations using Kane's and Lagrange's methods, and implementing simulation models in Simulink for verification and controller design.

Project Goals: The main goals of this project are:

- To derive closed-form symbolic equations for both forward and inverse kinematics of the 3RRP mechanism.
- To compute the largest square workspace that allows for all possible end-effector orientations.
- To derive the kinematic Jacobian and compute the Global Isotropy Index (GII).
- To develop equations of motion using Kane's and Lagrange's methods and compare their results.
- To implement Simulink models for kinematics and dynamic simulations.

Report Structure: The remainder of this report is organized as follows:

- **Section 3: Problem Definition** — Describes the mechanisms, system overview, and the scope of analysis.
- **Section 4: Analytical Derivations and Results** — Presents the symbolic derivations for kinematics, Jacobians, dynamics, workspace calculations, and simulation results focused on the 3RRP mechanism.
- **Section 5: Discussion** — Analyzes the findings, compares methodologies, and discusses challenges and insights.
- **Section 6: Conclusion** — Summarizes key outcomes and outlines potential future work, including further development on Part B and combined system analysis.

2 Problem Definition

2.1 System Overview

The focus of this study is the 3RRP mechanism, a robotic system composed of three serial links connected by revolute-revolute-prismatic joints. This section provides an overview of the mechanism, its reference frames, basis vectors, relevant points, and any external forces or torques acting on it. Additionally, we zoom in on the end-effector configuration for a closer look at its reference frames and orientation.

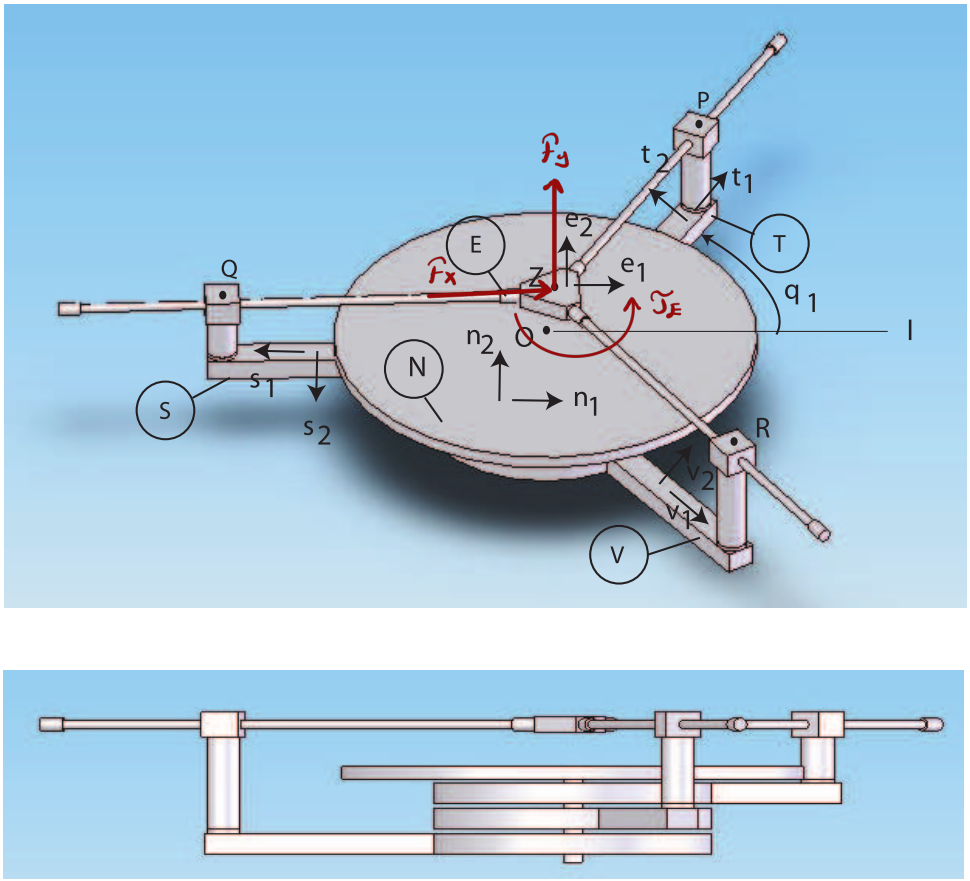


Figure 1: Labeled schematic of the 3RRP mechanism showing reference frames, basis vectors, key points, and external forces/torques.

The above figure (Figure 1) illustrates the overall structure of the 3RRP mechanism, including all necessary labels to understand the system's configuration and the forces applied.

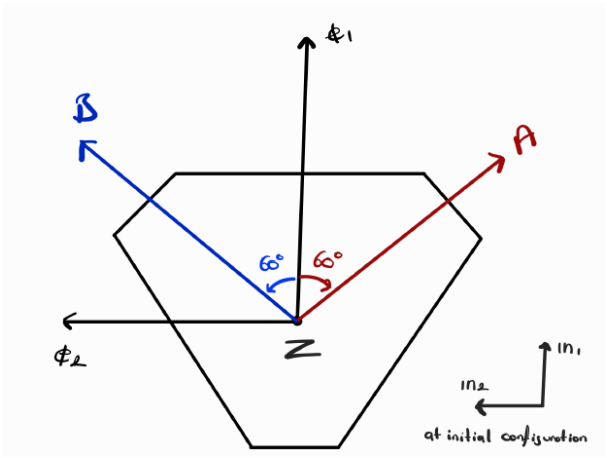


Figure 2: Zoomed-in diagram of the end-effector showing additional frames and detailed orientation.

2.2 Mechanism Description

The 3RRP mechanism features three joints in sequence—two revolute joints followed by a prismatic joint:

- **Degrees of Freedom (DoF):** The mechanism provides three degrees of freedom, allowing complex planar motion of the end-effector.

- **Joint Types and Characteristics:**
  - Revolute joints: Allow rotation between links.
  - Prismatic joint: Provides linear extension or retraction.
- **Inputs and Outputs:**
  - Inputs: Joint angles and extension lengths (q_1, q_2, q_3).
  - Output: Position (x, y) and orientation θ of the end-effector.
- **Motion Capabilities:** The mechanism can achieve a variety of configurations, and the end-effector can reach any position within its workspace with full rotational freedom.

2.3 Scope of Analysis

The primary focus of this analysis is on:

- Derivation of forward and inverse kinematics.
- Computation of the workspace and the kinematic Jacobian.
- Evaluation of the Global Isotropy Index (GII).
- Derivation of dynamic equations using Kane's and Lagrange's methods.
- Development and verification of Simulink models for simulation purposes.

Although initial work was attempted on the Linear Delta mechanism (Part B), this report emphasizes the detailed analysis and simulation of the 3RRP mechanism. The combined system will be addressed in future work.

Assumptions and Simplifications: - The links are assumed to be rigid with symmetric configurations. - Ideal joints with no friction or backlash are considered for simplicity. - External forces/torques depicted in Figure 1 are included where relevant for dynamic analysis.

3 Analytical Derivations and Results

3.1 Kinematic Analysis (3RRP Focus)

3.1.1 Symbolic Derivations

In this subsection, we present the final closed-form equations for forward and inverse kinematics of the 3RRP mechanism. These equations relate the joint variables q_1, q_2, q_3 to the end-effector's position (x, y) and orientation θ. The detailed derivations leading to these equations are provided in Appendix A.

Forward Kinematics: The forward kinematics problem involves computing the end-effector's position and orientation given the joint angles. For the 3RRP mechanism, the closed-form equations are:

$$x = -\frac{M}{\sqrt{3}(K^2 + L^2)}, \tag{1}$$

$$y = c_{22} - \frac{K}{L}c_{21} - \frac{KM}{\sqrt{3}L(K^2 + L^2)}, \tag{2}$$

$$\theta = \arctan 2(K, L), \tag{3}$$

where:

$$K = c_{12} + c_{32} + \sqrt{3}c_{31} - 2c_{22} - \sqrt{3}c_{11}, \tag{4}$$

$$L = c_{11} + c_{31} + \sqrt{3}c_{12} - 2c_{21} - \sqrt{3}c_{32}, \tag{5}$$

$$M = L(L - \sqrt{3}K)c_{12} - L(K + \sqrt{3}L)c_{11} - (L - \sqrt{3}K)(Lc_{22} - Kc_{21}). \tag{6}$$

The coefficients c_{ij} are defined as:

$$c_{11} = r\cos(q_1), \qquad c_{12} = r\sin(q_1), \tag{7}$$
$$c_{21} = r\cos(q_2), \qquad c_{22} = r\sin(q_2), \tag{8}$$
$$c_{31} = r\cos(q_3), \qquad c_{32} = r\sin(q_3). \tag{9}$$

These equations provide a direct method to compute the end-effector's pose from given joint angles.

Inverse Kinematics: The inverse kinematics problem is the reverse: determining the joint angles required to achieve a desired end-effector pose (x, y, θ). The closed-form solutions for the 3RRP mechanism are:

$$q_1 = \arctan 2(M_1, L_1), \tag{10}$$
$$q_2 = \arctan 2(M_2, L_2), \tag{11}$$
$$q_3 = \arctan 2(M_3, L_3), \tag{12}$$

where:

$$M_1 = K_1\cos\left(\theta + \frac{\pi}{3}\right) - \sqrt{r^2 - K_1^2}\sin\left(\theta + \frac{\pi}{3}\right), \tag{13}$$
$$L_1 = -K_1\sin\left(\theta + \frac{\pi}{3}\right) - \sqrt{r^2 - K_1^2}\cos\left(\theta + \frac{\pi}{3}\right), \tag{14}$$
$$M_2 = K_2\cos\left(\theta + \pi\right) - \sqrt{r^2 - K_2^2}\sin\left(\theta + \pi\right), \tag{15}$$
$$L_2 = -K_2\sin\left(\theta + \pi\right) - \sqrt{r^2 - K_2^2}\cos\left(\theta + \pi\right), \tag{16}$$
$$M_3 = K_3\cos\left(\theta - \frac{\pi}{3}\right) - \sqrt{r^2 - K_3^2}\sin\left(\theta - \frac{\pi}{3}\right), \tag{17}$$
$$L_3 = -K_3\sin\left(\theta - \frac{\pi}{3}\right) - \sqrt{r^2 - K_3^2}\cos\left(\theta - \frac{\pi}{3}\right). \tag{18}$$

The intermediate variables K_1, K_2, K_3 are given by:

$$K_1 = x\sin\left(\theta + \frac{\pi}{3}\right) - y\cos\left(\theta + \frac{\pi}{3}\right), \tag{19}$$
$$K_2 = x\sin\left(\theta + \pi\right) - y\cos\left(\theta + \pi\right), \tag{20}$$
$$K_3 = x\sin\left(\theta - \frac{\pi}{3}\right) - y\cos\left(\theta - \frac{\pi}{3}\right). \tag{21}$$

These formulas provide the necessary joint angles to attain a desired end-effector pose, completing the inverse kinematics analysis.

For the step-by-step derivation of these results, please refer to Appendix A.

3.1.2 Workspace Calculation

To determine the largest symmetric workspace of the 3RRP mechanism, we employed a numerical simulation using MATLAB. The goal was to compute all possible end-effector positions (x, y) reachable under the assumption of symmetric link lengths $l_1 = l_2 = l_3 = L$, with $L = 200$ mm, and verify the achievable orientations.

Logic Behind the Code: The provided MATLAB code (see Appendix B for full implementation) performs the following steps:

1. **Parameter Initialization:** The code sets the link length L and initializes a symmetric link length parameter $r = L$. It defines a dense grid of possible joint angles q_1, q_2, and q_3 over the range 0 to 2π radians with a specified resolution.

2. **Parallel Computation for Efficiency:** Using nested `parfor` loops, the code iterates over all combinations of q_1, q_2, and q_3. This parallel approach accelerates the computation by distributing tasks across multiple cores.

3. **Forward Kinematics Calculation:** For each combination of joint angles, the function `calculateEndEffectorPos` computes the end-effector's position (x, y) using the forward kinematics equations. These equations involve calculating intermediate variables and subsequently determining x, y, and θ.

4. **Workspace Data Collection:** The code collects all computed (x, y) points corresponding to the various joint configurations and stores them in arrays.

5. **Visualization:** Once all points are collected, the code plots them to visualize the workspace. The resulting plot displays the boundary of the reachable area by the end-effector.

Results: The simulation revealed that the set of all reachable (x, y) positions forms a circular region. The radius of this circle was found to be approximately 230 mm, which defines the largest symmetric workspace of the 3RRP mechanism under the given assumptions.

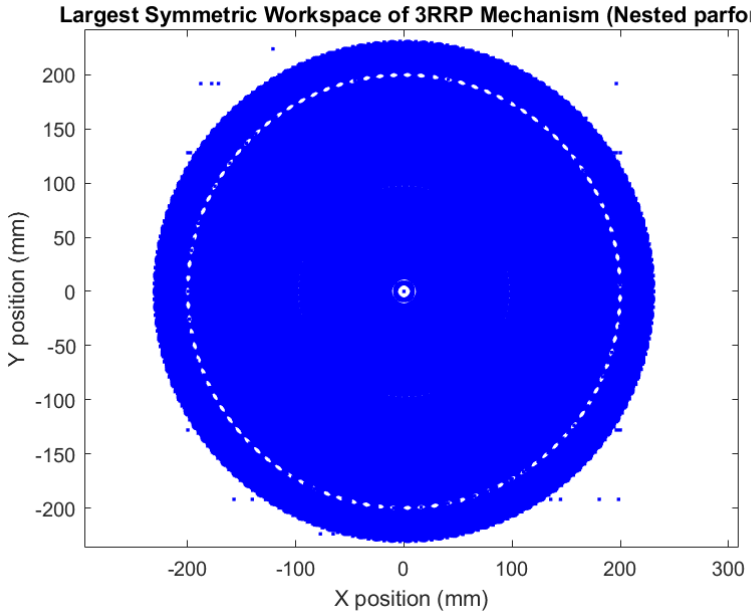


Figure 3: Calculated Largest Symmetric Workspace of the 3RRP Mechanism

3.1.3 Kinematic Jacobian

The kinematic Jacobian matrix $J(q_1, q_2, q_3)$ provides a relationship between the joint velocities $\dot{q}_1$, $\dot{q}_2$, $\dot{q}_3$ and the end-effector's linear and angular velocities $(\dot{x}, \dot{y}, \dot{\theta})$. It plays a critical role in velocity analysis, singularity identification, and dynamic performance evaluation.

Computation Using Autolev: To derive the Jacobian matrix for the 3RRP mechanism, we utilized Autolev, a computer algebra system designed for mechanics. In Autolev, the Jacobian was calculated with the following command:

```
JACOBIAN =[D(u7,u1), D(u7,u2), D(u7,u3);

          D(u8,u1), D(u8,u2), D(u8,u3);

          D(u9,u1), D(u9,u2), D(u9,u3)]
```

where the variables are defined as:

$$\begin{aligned} \dot{q}_1 &= u_1, \quad \dot{q}_2 = u_2, \quad \dot{q}_3 = u_3, \\ \dot{s}_1 &= u_4, \quad \dot{s}_2 = u_5, \quad \dot{s}_3 = u_6, \\ \dot{x} &= u_7, \quad \dot{y} = u_8, \quad \dot{\theta} = u_9. \end{aligned}$$

Here, $D(\cdot, \cdot)$ denotes the partial derivative operation in Autolev, and this command computes the partial derivatives of the end-effector velocities with respect to the joint velocities, constructing the 3×3 Jacobian matrix.

Symbolic Representation: Symbolically, this computation corresponds to evaluating:

$$\text{JACOBIAN} = \begin{bmatrix} \frac{\partial \dot{x}}{\partial \dot{q}_1} & \frac{\partial \dot{x}}{\partial \dot{q}_2} & \frac{\partial \dot{x}}{\partial \dot{q}_3} \\ \frac{\partial \dot{y}}{\partial \dot{q}_1} & \frac{\partial \dot{y}}{\partial \dot{q}_2} & \frac{\partial \dot{y}}{\partial \dot{q}_3} \\ \frac{\partial \dot{\theta}}{\partial \dot{q}_1} & \frac{\partial \dot{\theta}}{\partial \dot{q}_2} & \frac{\partial \dot{\theta}}{\partial \dot{q}_3} \end{bmatrix},$$

which reflects the partial derivatives of the end-effector's velocity components with respect to each joint velocity.

Resulting Expression: Due to the complexity of the 3RRP mechanism, the explicit symbolic form of the Jacobian matrix is extensive. The full expression is provided in Appendix C for reference. This matrix encapsulates how changes in the joint variables affect the end-effector's motion.

Interpretation and Significance: The Jacobian matrix is used to:

- Compute the end-effector velocities given a set of joint velocities.
- Analyze singular configurations where the mechanism loses degrees of freedom or gains uncontrolled movements.
- Inform the design of controllers that manipulate the mechanism in real-time.

The computation of the Jacobian in Autolev streamlines the derivation process and ensures accuracy, while its lengthy form is documented in Appendix C.

Further details on the Autolev code implementation and derivation of the Jacobian will be discussed in subsequent sections.

3.1.4 Global Isotropy Index (GII)

The Global Isotropy Index (GII) quantifies the uniformity of a manipulator's performance across its workspace. A higher GII indicates more isotropic (uniform) behavior, which is desirable for consistent performance in all directions.

GII Computation and Visualization: To compute the GII for the 3RRP mechanism, we executed a MATLAB script (see Appendix D for full code) that:

1. **Initialization:** Sets up the mechanism parameters (e.g., link length r), default values for system parameters s_1, s_2, s_3, and defines the range and resolution for each joint variable q_1, q_2, and q_3.
2. **Workspace Sampling:** Generates a grid of joint angle combinations using specified sampling resolution. The total number of combinations is determined by the sampling density for each joint.
3. **Parallel Computation:** Utilizes a parallel loop (`parfor`) to efficiently compute forward kinematics and the Jacobian matrix for each joint configuration. For every sampled configuration, the script:
   - Computes intermediate values necessary for forward kinematics.
   - Calculates the end-effector position and orientation (x, y, θ).
   - Assembles the Jacobian matrix for the current configuration.
   - Performs Singular Value Decomposition (SVD) on the Jacobian to extract the smallest ($\sigma_{\min}$) and largest ($\sigma_{\max}$) singular values.
4. **Singular Value Analysis:** After processing all valid configurations, the script identifies the minimum of all computed $\sigma_{\min}$ values and the maximum of all $\sigma_{\max}$ values across the workspace.

5. **GII Calculation:** Calculates the Global Isotropy Index using the formula:

$$\text{GII} = \frac{\min_{\gamma_0 \in W} \sigma_{\min}(J)}{\max_{\gamma_1 \in W} \sigma_{\max}(J)}$$

   and outputs the computed GII value.

6. **Workspace Visualization:** Recomputes the reachable (x, y) positions for valid configurations and visualizes the workspace using a scatter plot. Each point in the workspace is colored according to its corresponding minimum singular value, providing insight into the isotropy distribution across the workspace.

The MATLAB code that performs these computations and visualizations is provided in detail in Appendix D.

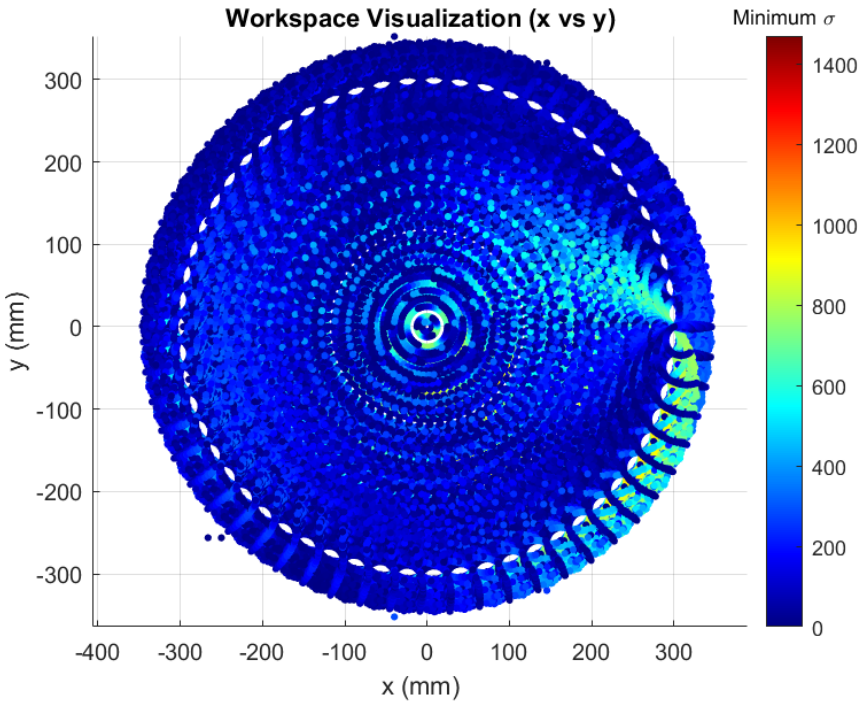


Figure 4: Workspace visualization colored by the minimum singular value, illustrating the Global Isotropy Index for the 3RRP mechanism.

3.2 Dynamic Modeling

Dynamic modeling is crucial for predicting and controlling the behavior of robotic mechanisms under various operating conditions. By formulating the equations of motion, we can accurately simulate and analyze how the 3RRP mechanism responds to external forces, torques, and motion inputs.

In this section, we derive the dynamic equations of the 3RRP mechanism using two well-known approaches: Kane's method and Lagrange's method. Each approach has its own advantages in terms of computational efficiency, conceptual clarity, and ease of extension. After presenting both methods, we compare the resulting equations to highlight differences and similarities in their formulation.

3.2.1 System Setup in Autolev

Before applying Kane's method to derive the equations of motion, we first establish the 3RRP system in Autolev with appropriate frames, constraints, and forces. The following sections summarize these key elements.

Rotations of Frames w.r.t. the Newtonian Frame Autolev uses `SIMPROT` commands to define the rotation of each moving frame relative to the inertial (Newtonian) frame **N**. In this model:

- The frames **T**, **S**, **V** rotate about the z-axis of **N** (labeled as **N3**) by angles q_1, q_2, and q_3, respectively.
- The end-effector frame **E** also rotates about **N3** by the angle θ.
- Two additional frames **A** and **B** are defined relative to the end-effector frame **E** by fixed angles of $\pm 60°$.

In Autolev, these rotations are specified with:

```
SIMPROT(N,T,3,q1)
SIMPROT(N,S,3,q2)
SIMPROT(N,V,3,q3)
SIMPROT(N,E,3,theta)

SIMPROT(E,A,3,60)
SIMPROT(E,B,3,-60)
```

These commands establish the proper orientation relationships among the frames for subsequent kinematic and dynamic analyses.

Configuration Constraints To ensure the correct geometric relationships between points on the mechanism, Autolev uses `LOOP` vectors and `ZeroConfig` equations. For the 3RRP mechanism:

```
LOOP1> = P_Z_Q> + P_Q_O> + P_O_Z>
LOOP2> = P_Z_R> + P_R_O> + P_O_Z>
LOOP3> = P_Z_P> + P_P_O> + P_O_Z>

ZeroConfig[1] = DOT(LOOP1>,N1>)
ZeroConfig[2] = DOT(LOOP1>,N2>)
ZeroConfig[3] = DOT(LOOP2>,N1>)
ZeroConfig[4] = DOT(LOOP2>,N2>)
ZeroConfig[5] = DOT(LOOP3>,N1>)
ZeroConfig[6] = DOT(LOOP3>,N2>)
```

Here:

- $\mathbf{P_{ZQ}} >, \mathbf{P_{QO}} >, \mathbf{P_{OZ}} >$ define the closed-loop geometry for one loop of the mechanism, and similarly for the other loops.
- `ZeroConfig[1 ...6]` impose the conditions that the dot products of these loop vectors with **N1** and **N2** (the x- and y-directions in the inertial frame) must be zero, effectively closing each loop in the mechanism at the desired configuration.

Motion Constraints Beyond the static (configurational) constraints, Autolev also forms time derivatives of these loops to establish velocity-level constraints. Each `dLOOP` vector is computed with respect to the inertial frame, and then used to define dependent motion equations:

```
dLOOP1> = dt(LOOP1>,N)
dLOOP2> = dt(LOOP2>,N)
dLOOP3> = dt(LOOP3>,N)

Dependent[1] = dot(dLOOP1>,N1>)
Dependent[2] = dot(dLOOP1>,N2>)
Dependent[3] = dot(dLOOP2>,N1>)
Dependent[4] = dot(dLOOP2>,N2>)
Dependent[5] = dot(dLOOP3>,N1>)
Dependent[6] = dot(dLOOP3>,N2>)
```

These velocity-level constraints ensure that the loops remain closed as the mechanism moves, dictating relationships among the joint velocities (e.g., prismatic and revolute joints).

Forces and Torques Autolev allows the definition of external forces (such as gravity and end-effector loads) and internal actuator torques. In the 3RRP system, we include:

```
Gravity( -g*N3> )
Force_Z>  = FE1*N1> + FE2*N2>
Torque_E> = TZ*N3>
```

```
Torque_S> = TS1*S3>
Torque_T> = TT1*T3>
Torque_V> = TV1*V3>
```

This setup applies:

- A uniform gravitational force $-g\,\mathbf{N3}>$ on all bodies.
- End-effector forces $\mathbf{FE1}, \mathbf{FE2}$ in the $\mathbf{N1}$ and $\mathbf{N2}$ directions.
- Actuator torques $\mathbf{TS1}, \mathbf{TT1}, \mathbf{TV1}$ about the joints $\mathbf{S3}, \mathbf{T3}, \mathbf{V3}$, respectively.
- An external torque $\mathbf{TZ}\,\mathbf{N3}>$ on the end-effector frame if needed.

Summary and References By defining the rotations, constraints, and applied forces/torques in this manner, we establish a comprehensive Autolev model of the 3RRP mechanism. The subsequent derivation of the equations of motion (using Kane's method) leverages these definitions to automatically generate the system's dynamic equations. In the following sections, we refer back to these frames, constraints, and force definitions as we derive and analyze the 3RRP mechanism's motion.

3.2.2 Kane's Method

Kane's method is a systematic approach to formulating the equations of motion for mechanical systems. Unlike the more traditional Lagrangian approach, Kane's method leverages generalized speeds and partial velocities to produce a compact set of governing equations. In this subsection, we outline how this method is applied to the 3RRP mechanism using Autolev.

Overview of Kane's Method Kane's method aims to simplify the derivation of equations of motion by focusing on generalized speeds rather than generalized coordinates alone. It constructs the equations via the principle

$$\sum(\mathbf{F}_{\text{active}} \cdot \mathbf{v}_i) + \sum(\mathbf{F}_{\text{inertial}} \cdot \mathbf{v}_i) = 0,$$

where $\mathbf{v}_i$ are the partial velocities corresponding to each generalized speed. This approach handles constraints in a straightforward manner and often yields fewer algebraic steps compared to other formulations.

Kinematical Differential Equations To implement Kane's method for the 3RRP mechanism, we define the following generalized speeds $\{u_i\}$ within Autolev. The relevant code snippet is shown below:

```
% Kinematical differential equations

q1' = u1          s1' = u4          x' = u7
q2' = u2          s2' = u5          y' = u8
q3' = u3          s3' = u6          theta' = u9
```

Here, q_1, q_2, q_3 are the revolute joint angles; s_1, s_2, s_3 are prismatic extensions if relevant for the mechanism; and x, y, θ describe the end-effector's planar pose. Their time derivatives are mapped to $u_1, u_2, \ldots, u_9$. By doing so, Autolev treats these generalized speeds directly when forming the system's dynamic equations.

Handling Dependent Variables Because the 3RRP mechanism includes closed-loop constraints, some variables become dependent. Autolev's `Constrain` command eliminates dependent generalized speeds automatically:

```
Constrain(Dependent[u4,u5,u6, u7,u8,u9])
```

This enforces the loop closure conditions (both configuration- and velocity-level) described previously, ensuring that motion constraints are upheld. As a result, Autolev solves for the dependent variables internally, leaving a minimal set of independent equations for $\dot{q}_1, \dot{q}_2, \dot{q}_3$.

Formation of the Equations of Motion Once the generalized speeds and constraints are defined, Autolev uses the following commands to assemble the system's dynamic equations via Kane's method:

```
%       Equations of motion
Zero = Fr() + FrStar()
Kane( )
```

In this snippet:

- `Fr()` represents the generalized active forces (e.g., actuator torques, external forces).
- `FrStar()` represents the generalized inertial forces (mass/inertia effects).
- Adding them produces a set of algebraic equations (`Zero`) that must equal zero.
- The `Kane()` command instructs Autolev to finalize the equations of motion, solving for the accelerations of the independent generalized coordinates.

The outcome is a set of ordinary differential equations (ODEs) governing the time evolution of the 3RRP mechanism. If these ODEs are lengthy, they can be relegated to an appendix (e.g., Appendix E) for clarity.

General Implementation Details Autolev's computer algebra capabilities automatically compute partial velocities and assemble all the terms associated with inertial, gravitational, and applied forces. Key points include:

- **Initial Conditions:** Joint angles (q_1, q_2, q_3) and prismatic extensions (s_1, s_2, s_3) can be specified, along with end-effector position (x, y) and orientation θ.
- **Constraint Enforcement:** Autolev respects the specified constraints, so any motion violating these constraints is automatically excluded.
- **Exporting to Simulation:** Commands like

  ```
  code dynamics() dinamik_yunus.m
  ```

  generate a MATLAB-compatible file that numerically integrates the resulting ODEs, enabling rapid simulation and analysis.

Interpretation of the Kane Formulation Kane's method is particularly well-suited to robotic applications with constraints, as it streamlines the algebra involved. By designating generalized speeds, we reduce the system of equations to a more manageable form. Some notable advantages include:

- **Direct Constraint Handling:** No need to introduce Lagrange multipliers explicitly.
- **Compact Equations:** Often, fewer symbolic manipulations are required compared to alternative methods.
- **Straightforward Extensions:** Nonholonomic or additional constraints can be incorporated consistently.

Summary Through these Autolev commands, we apply Kane's method to derive the 3RRP mechanism's equations of motion, respecting the geometric and velocity constraints of the system. The resulting ODEs form the backbone of our dynamic model, which we will use for simulation, control design, and further analysis. In the next subsection, we present the Lagrange formulation of the same mechanism to compare the two approaches and highlight any notable differences.

3.2.3 Lagrangian Formulation in Autolev with Baumgarte Stabilization

In this section, we describe how the Lagrangian method is implemented for the 3RRP mechanism in Autolev, emphasizing the Euler–Lagrange formulation, the inclusion of constraint forces via Lagrange multipliers, and the addition of Baumgarte stabilization terms.

Forming the Lagrangian The Lagrangian is classically defined as

$$\text{Lag} = T - V,$$

where T is the total kinetic energy, and V is the potential energy. In our Autolev code:

```
KE = KE()      % Kinetic energy automatically computed by Autolev
PE = 0         % No explicit potential energy term (e.g., gravity modeled as a
    force)
Lag = KE - PE % Form the Lagrangian
```

Autolev's `KE()` function computes the total kinetic energy from masses, inertia tensors, and velocities of all bodies (links and end-effector). Here, `PE = 0` means we are not adding extra potential energy terms (e.g., gravitational potential might be accounted for as a force, or no springs are present).

Euler–Lagrange Terms The Euler–Lagrange equations for each generalized coordinate q_i follow the well-known form:

$$\frac{d}{dt}\left(\frac{\partial \text{Lag}}{\partial \dot{q}_i}\right) - \frac{\partial \text{Lag}}{\partial q_i} = Q_i,$$

where Q_i is the generalized (non-conservative) force corresponding to q_i. In Autolev, we compute these partial derivatives as:

```
ddLag = [dt(d(Lag,q1')); dt(d(Lag,q2')); dt(d(Lag,q3'));
         dt(d(Lag,s1')); dt(d(Lag,s2')); dt(d(Lag,s3'));
         dt(d(Lag,X')); dt(d(Lag,Y')); dt(d(Lag,theta'))]

dLag = [d(Lag,q1); d(Lag,q2); d(Lag,q3);
        d(Lag,s1); d(Lag,s2); d(Lag,s3);
        d(Lag,X); d(Lag,Y); d(Lag,theta)]
```

Here:

- `ddLag` represents $\frac{d}{dt}\left(\frac{\partial \text{Lag}}{\partial \dot{q}_i}\right)$.
- `dLag` corresponds to $\frac{\partial \text{Lag}}{\partial q_i}$.

Each index in these arrays matches one of the nine generalized coordinates $\left(q_1, q_2, q_3, s_1, s_2, s_3, X, Y, \theta\right)$ used to describe the mechanism.

Generalized Forces Next, we compute the generalized forces Q_i by extracting the virtual work contribution of non-conservative forces and torques:

```
Work = dot(W_S_N>,Torque_S>) + dot(W_T_N>,Torque_T>) +
       dot(W_V_N>,Torque_V>) + dot(V_Z_N>,Force_Z>)
Q = [coef(Work,q1'); coef(Work,q2'); coef(Work,q3');
     coef(Work,s1'); coef(Work,s2'); coef(Work,s3');
     coef(Work,X'); coef(Work,Y'); coef(Work,theta')]
```

- `Torque_S`, `Torque_T`, `Torque_V` are applied torques about the respective frames $\mathbf{S3}, \mathbf{T3}, \mathbf{V3}$. - `Force_Z` is an external force on the end-effector, and `V_Z_N>` is its velocity in the inertial frame. - The `coef()` function associates each generalized velocity (like q_1') with its corresponding force coefficient, forming the generalized force vector $\{Q_i\}$.

Equations of Motion: Unconstrained vs. Constrained Combining these terms yields the unconstrained equations of motion:

$$\text{Zero\_EoM} = \underbrace{\texttt{ddLag}}_{\frac{d}{dt}\left(\frac{\partial \text{Lag}}{\partial \dot{q}_i}\right)} - \underbrace{\texttt{dLag}}_{\frac{\partial \text{Lag}}{\partial q_i}} - \underbrace{\texttt{Q}}_{Q_i}.$$

This appears in code as:

```
Zero_EoM = ddLag - dLag - Q
```

However, the 3RRP mechanism has loop-closure constraints. We introduce Lagrange multipliers $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_6)$ to incorporate these constraints into the dynamic equations:

$$\text{Zero\_Constrained\_EoM} \;=\; \text{Zero\_EoM} \;+\; \big(\texttt{transpose(dZeroConfig)}\big)\,\Lambda.$$

In Autolev:

```
Zero_Constrained_EoM = Zero_EoM + transpose(dZeroConfig)*Lambda
```

Here, `dZeroConfig` is the Jacobian matrix of the constraint equations (i.e., partial derivatives of the loop-closure functions), and multiplying by the Lagrange multipliers Λ enforces those constraints in the motion-level equations.

Baumgarte Stabilization In many multibody simulations, purely enforcing constraints through multipliers can lead to numerical drift: small errors in positions or velocities grow over time. **Baumgarte stabilization** adds proportional–derivative feedback on the constraint errors to reduce this drift. The main idea is to replace the strict acceleration-level constraint

$$\ddot{C}(q, \dot{q}, \ddot{q}) \;=\; 0$$

with

$$\ddot{C}(q, \dot{q}, \ddot{q}) \;+\; \alpha\,\dot{C}(q, \dot{q}) \;+\; \beta\,C(q) \;=\; 0,$$

where α, β are user-defined gains. The extra terms $\alpha\,\dot{C}$ and $\beta\,C$ act like a PD-controller on the constraint error, stabilizing it at zero.

In the Autolev code, the lines:

```
first_term  = -dtemp*dt(q_vec)
second_term = -2*dt(dZeroConfig)*dt(q_vec)
third_term  = -dt(dt(ZeroConfig))
fourth_term = -alpha*(dZeroConfig*dt(q_vec)-third_term)
fifth_term  = -beta*ZeroConfig
gamma       = first_term + second_term + third_term + fourth_term + fifth_term
extra_term  = transpose(dZeroConfig)*Lambda
```

correspond to computing $\dot{C}$ and $\ddot{C}$ for the loop-closure constraints $C(q) = 0$ (and their time derivatives), then adding the Baumgarte correction terms. Specifically,

- α multiplies the velocity-level constraint error ($\dot{C}$).
- β multiplies the position-level constraint error (C).
- The code modifies the final equations of motion to include these corrections, preventing numerical drift of the constraints.

Solving for Accelerations and Multipliers Finally, Autolev solves the combined set of differential-algebraic equations (DAEs) for the second derivatives of the generalized coordinates and for the Lagrange multipliers:

```
solve(eqn, [dt(dt(q_vec)); Lambda])
```

This procedure yields the accelerations $(q_1'', q_2'', \ldots, \theta'')$ that respect both dynamics and constraints (including Baumgarte stabilization), as well as the multipliers λ_i that represent constraint forces.

Summary Through these steps, we apply the Euler–Lagrange formulation to the 3RRP mechanism, capture non-conservative torques/forces as generalized forces, and explicitly handle loop-closure constraints with Lagrange multipliers. Baumgarte stabilization reduces constraint violation over time by adding PD-like terms on the constraint error. The final outcome is a set of numerically robust equations of motion suitable for simulation and analysis in downstream environments (e.g., MATLAB/Simulink).

3.2.4 Method Comparison

Having derived the equations of motion for the 3RRP mechanism using both Kane's and Lagrange's methods, we now compare the two approaches in terms of derivation complexity, computational efficiency, and interpretability. For completeness, the Autolev codes used for each method are provided in Appendix E.

1) Derivation Complexity

- **Equation Length and Effort:** In our experience, Kane's method yielded more compact intermediate expressions, especially when handling velocity-level constraints. In contrast, the Lagrangian approach required explicit introduction of Lagrange multipliers for loop-closure constraints, increasing the symbolic complexity.
- **Handling of Constraints:** Kane's method allows direct enforcement of constraints via partial velocities, whereas Lagrange's method requires additional multipliers. For a closed-loop mechanism like 3RRP, the extra step of formulating and stabilizing constraint equations via Baumgarte was somewhat more involved in the Lagrangian approach.

2) Computational Efficiency

- **Symbolic Computation:** Both methods were implemented in Autolev, which automatically performs partial derivatives and matrix assembly. Kane's method tended to produce slightly shorter symbolic expressions, and it processed faster in some tests, though the difference was not prohibitive.
- **Numerical Integration:** Simulations of both formulations, after exporting to MATLAB, showed comparable run times. However, the Lagrange-based code required careful tuning of Baumgarte gains to prevent numerical drift. The simpler constraint enforcement in Kane's method reduced the need for extensive tuning.

3) Interpretability and Physical Insight

- **Energy-Based vs. Force-Based Views:** Lagrange's method directly relates kinetic and potential energies, offering clear insights when energy terms are a focal point (e.g., adding springs or analyzing energy conservation). Kane's method, by contrast, is often more direct for force/torque-driven analyses and can simplify constraint modeling.
- **Equation Structure:** The final Lagrangian equations required explicit constraint terms and multipliers. Kane's formulation consolidated these elements through partial velocities and effectively embedded the constraint relations in the generalized force components.

4) Practical Considerations

- **Ease of Implementation:** Kane's method and Lagrange's method each have dedicated Autolev functions. In practice, using Kane's method felt more streamlined for the 3RRP's multiple closed loops, since the tool automatically resolved dependent speeds. On the other hand, the Lagrangian approach is straightforward to interpret physically, but more tedious to manage constraints.
- **Scalability:** Both methods can scale to higher DoFs or 3D systems, but the added constraints could become cumbersome in Lagrange's method. Kane's method remains appealing for larger systems with many loop closures, given its partial velocity framework.
- **Controller Design:** For basic PD controllers, either formulation suffices. However, force/torque-based control laws might be slightly more direct with Kane's equations, whereas potential/energy-based controllers (e.g., passivity-based) may be more transparent in the Lagrangian formulation.

5) Summary of Key Findings

- **Advantages of Kane's Method:**
  - Naturally incorporates constraints using partial velocities.
  - Often leads to more compact symbolic expressions.
  - Potentially simpler for force/torque-focused analysis.
- **Advantages of Lagrange's Method:**
  - Directly tied to energy principles and potential functions.

- Well-known standard procedure with straightforward interpretation.
- Useful for systems requiring explicit energy-based analyses.

Overall, **Kane's method** can be more efficient for mechanisms with multiple loops, while **Lagrange's method** offers a clear energy-based interpretation. In practice, the choice may depend on the specific application and whether energy or force analyses are the primary concern. For this 3RRP mechanism, both methods yield valid results; the preference for one over the other might hinge on the user's familiarity with energy-based vs. partial-velocity formulations.

3.2.5 Simulation and Validation Using Kane's Dynamic Code

This section presents the simulation results obtained from the 3RRP mechanism's dynamic equations, which were derived using Kane's method. We focus on verifying the model's response under small force and torque disturbances applied to the end-effector.

Simulation Setup

- **Environment and Code:** The dynamic code generated by Autolev (Kane's method) was implemented in MATLAB. For details on the Autolev script and code structure, refer to Appendix E.
- **Solver and Parameters:** We used MATLAB's `ode45` solver with a simulation duration of 5 seconds and a time step size of 0.001 seconds.
- **Initial Conditions:** Unless otherwise stated, the 3RRP joints and end-effector start in a nominal configuration with zero velocities.
- **Disturbances:** We applied a small 0.001 N or 0.001 N·m torque to the end-effector in specific directions to evaluate the system's dynamic response.

Implementation of Kane's Dynamics Kane's method handles loop-closure constraints through partial velocities, reducing the complexity of enforcing constraints explicitly. The Autolev-generated code automatically computes the system accelerations based on the generalized speeds, masses, inertias, and any applied forces or torques.

Simulation Results and Discussion We carried out four distinct simulations, each focusing on a different end-effector loading scenario. The plots below illustrate how the position and orientation of the end-effector evolve over time in response to the applied disturbances.

1. **Small Force Along $-X$ Direction** A force of 0.001 N is applied along the negative x-axis of the mechanism's base frame. Figure 5 shows the end-effector position (x, y) and orientation θ over time.

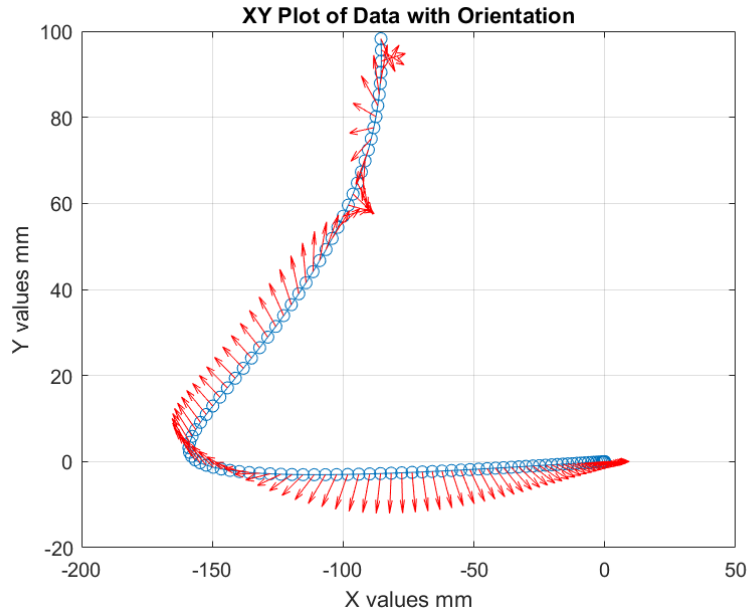


Figure 5: End-effector response under a 0.001 N force along $-X$.

The end-effector moves slightly in the negative x-direction before stabilizing, indicating the mechanism's relatively stiff response.

2. **Small Force Along $-Y$ Direction** A force of 0.001 N is applied along the negative y-axis. Figure 6 depicts the resulting trajectory.

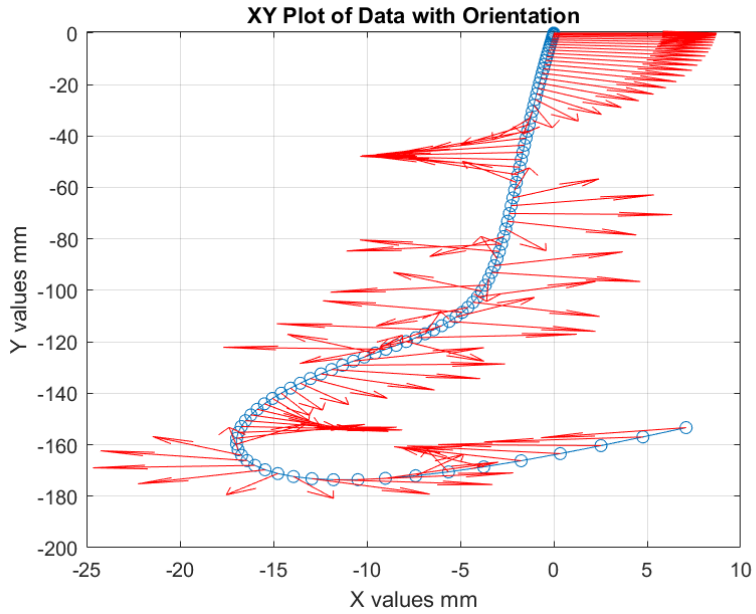


Figure 6: End-effector response under a 0.001 N force along $-Y$.

Similar to the $-X$ case, the end-effector shifts slightly in the negative y-direction and settles to a nearby equilibrium, demonstrating consistent dynamic behavior.

3. **Small Force Along Both $-X$ and $-Y$ Directions** Next, we applied a combined force of 0.001 N simultaneously along the $-X$ and $-Y$ axes. Figure 7 shows the end-effector's motion.

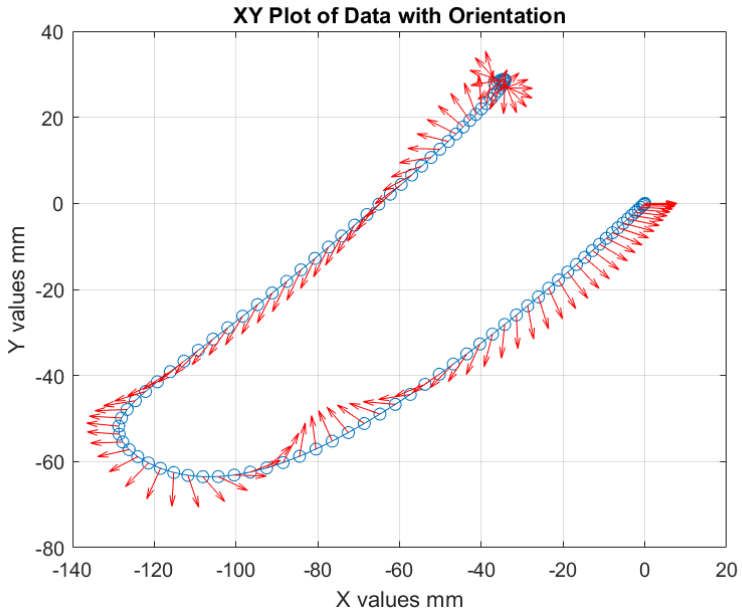


Figure 7: End-effector response under a 0.001 N force along $-X$ and $-Y$.

The mechanism responds with a diagonal shift in the $-X, -Y$ quadrant. The orientation θ shows minor deviations, reflecting small coupled effects on the end-effector's rotation.

4. **Small Torque Along $N3$ Direction** Finally, we applied a torque of 0.001 N·m around the $N3$ axis (i.e., negative z-axis of the inertial frame). Figure 8 captures the end-effector orientation changes.

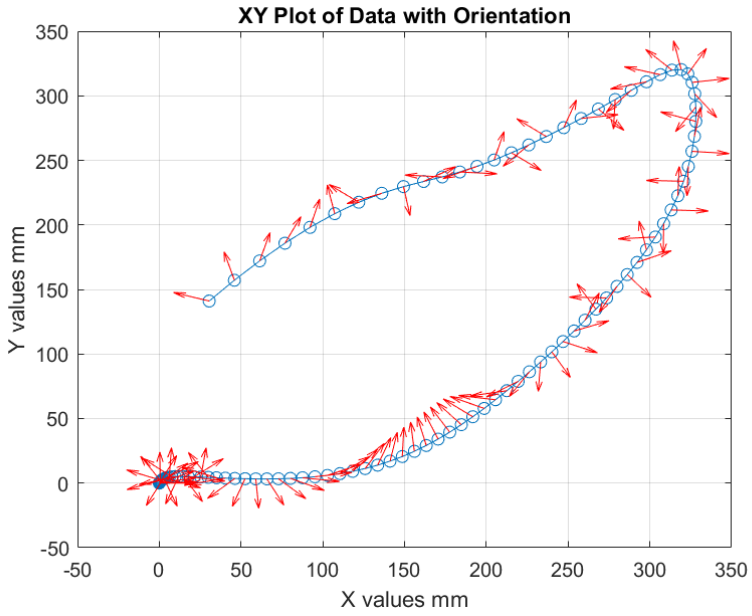


Figure 8: End-effector response under a 0.001 N·m torque around $N3$.

In this scenario, the end-effector rotates slightly while maintaining its planar constraints. The small magnitude of the torque produces only a gentle angular displacement over the simulation duration.

Validation and Observations

- **Constraint Satisfaction:** Throughout all four simulations, the closed-loop constraints of the 3RRP mechanism remain satisfied, confirming the effectiveness of Kane's method in automatically handling dependent velocities.
- **Small Perturbation Behavior:** The low forces and torques produce modest displacements, demonstrating linear-like responses around the nominal configuration. This small-signal response is a valuable baseline for future control design.
- **Numerical Stability:** No numerical instabilities or constraint drift were observed, indicating that the code correctly integrates the equations of motion and respects loop closures.
- **Comparison with Physical Expectation:** The end-effector displacements and orientations align well with intuitive expectations of how a planar mechanism should react to small external loads.

Summary Using the Kane-derived dynamic code, we simulated the 3RRP mechanism's response to small forces and torques. The results confirm that the code accurately captures the mechanism's behavior, maintaining constraints and producing realistic motions. These validation efforts provide confidence in the correctness of the derived equations and pave the way for further analysis, such as controller design or path planning, where the mechanism's dynamic behavior under external perturbations is critical.

3.2.6 Kinematic Simulation in Simulink

Beyond the dynamic simulations, we also implemented and tested the 3RRP mechanism's forward and inverse kinematics in Simulink at both the configuration and motion levels. These simulations validate the correctness of our symbolic kinematic equations and the Jacobian-based velocity mapping.

Configuration-Level Kinematics Figure 9 showcases two Simulink blocks: the *configuration-level forward kinematics* block and the *configuration-level inverse kinematics* block.

- **Forward Kinematics Block (Configuration Level):** Takes the joint angles (q_1, q_2, q_3) as inputs and outputs the end-effector's configuration (x, y, θ) based on the derived closed-form equations.
- **Inverse Kinematics Block (Configuration Level):** Accepts the end-effector's desired configuration (x, y, θ) and computes the required joint angles to achieve that pose.

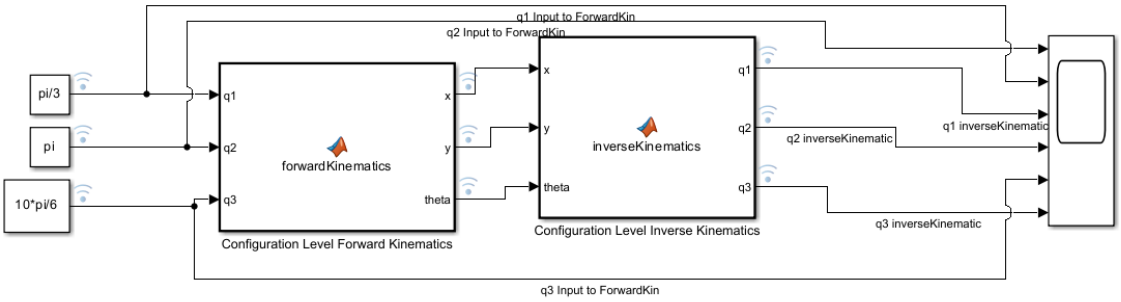


Figure 9: Configuration-level forward and inverse kinematics blocks in Simulink.

Motion-Level Kinematics Similarly, Figure 10 shows the *motion-level forward kinematics* block and the *motion-level inverse kinematics* block.

- **Forward Kinematics Block (Motion Level):** Accepts joint *trajectories* or time-varying joint angles and outputs the corresponding end-effector trajectory over time.
- **Inverse Kinematics Block (Motion Level):** Computes the joint trajectories necessary to follow a specified end-effector trajectory $(x(t), y(t), \theta(t))$.

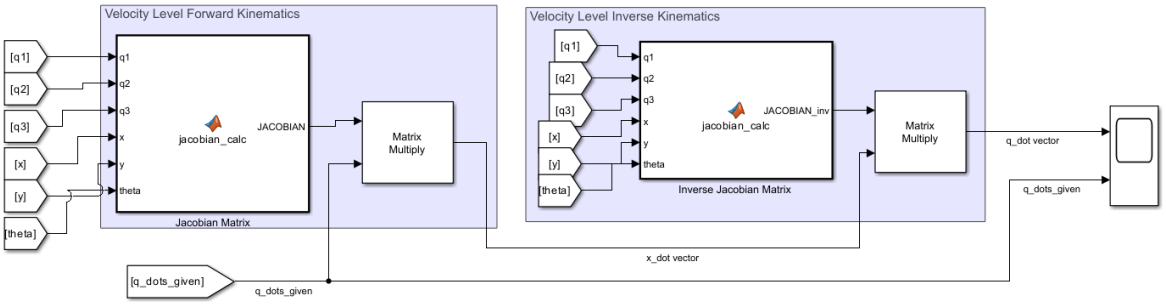


Figure 10: Motion-level forward and inverse kinematics blocks in Simulink.

Jacobian and Jacobian Transpose Blocks To facilitate velocity-level analyses and manipulator control strategies, we also built dedicated blocks for the *Jacobian* and its transpose, as displayed in Figure 11. These blocks:

- Compute the Jacobian matrix $J(q)$ given the current joint angles q.
- Output $J^\top(q)$ for tasks such as Jacobian transpose-based control or force mapping.

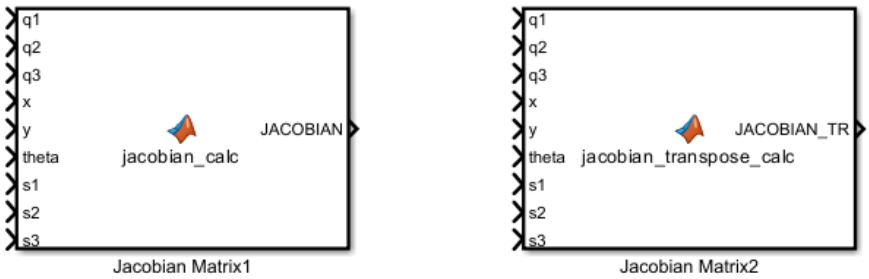


Figure 11: Jacobian and Jacobian transpose blocks in Simulink.

Verification of Kinematic Accuracy As a final check, we connected the forward and inverse kinematics blocks back-to-back and monitored the resulting joint angles. Figure 12 shows a sample output plot comparing the *input* joint angles to the *angles* recovered after passing through the forward $\rightarrow$ inverse pipeline.

- **Result:** The input angles and the output angles match closely, validating the correctness of the kinematic equations and their Simulink implementation.
- **Implication:** This ensures that for any desired (x, y, θ) within the reachable workspace, the inverse kinematics block yields the correct joint configuration, which the forward kinematics block can then accurately map back to the same (x, y, θ).

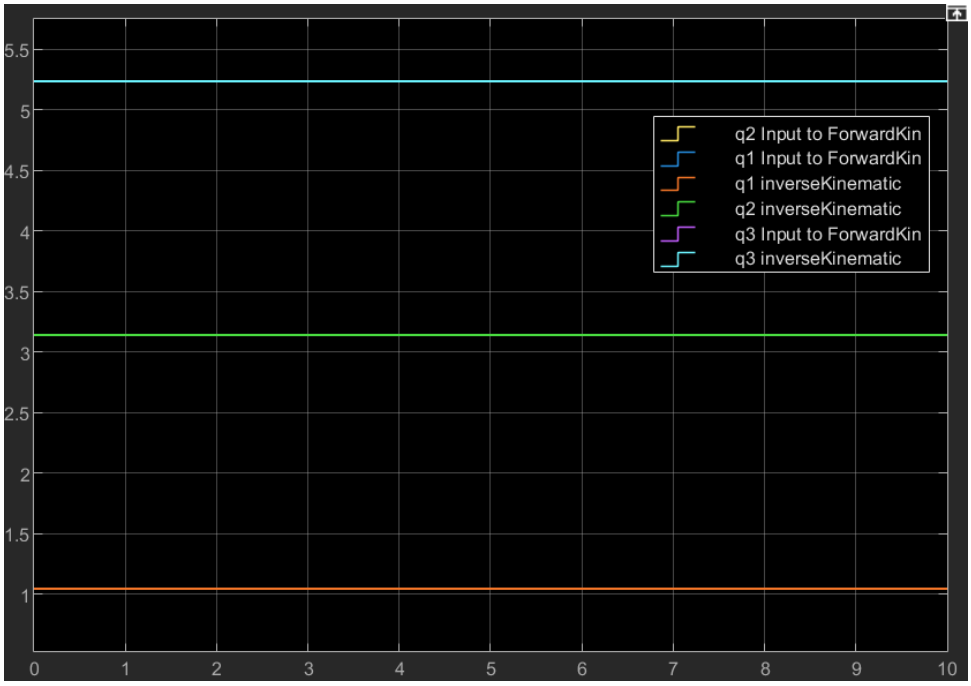


Figure 12: Input vs. output joint angles when forward and inverse kinematics blocks are connected.

Conclusions on Kinematic Simulation The Simulink models confirm that the symbolic kinematic equations for both configuration and motion levels are accurate. By verifying that the input and output angles coincide in a forward-inverse loop, we eliminate potential algebraic or sign errors. Moreover, the separate Jacobian blocks enable velocity-level and control-related analyses, paving the way for advanced manipulator control schemes in future work.

4 Discussion

4.1 Overview of Key Findings

In the preceding sections, we thoroughly examined the 3RRP mechanism by deriving and validating its kinematic and dynamic models. The kinematic study included closed-form forward and inverse solutions, workspace determination, and evaluation of the Jacobian matrix along with the Global Isotropy Index (GII). On the dynamic side, both Kane's and Lagrange's methods were employed, and their respective formulations were tested via simulation under small external forces and torques. This section consolidates these results to highlight their broader significance and practical implications.

4.2 Analysis of Kinematic Results

Forward and Inverse Kinematics Our forward and inverse kinematics analyses established a robust mapping between the joint variables (q_1, q_2, q_3) and the end-effector pose (x, y, θ). Notably, the verification in Simulink—where forward and inverse kinematic blocks were connected—confirmed the mathematical consistency of these solutions: input joint angles re-emerged intact, reinforcing the correctness of the symbolic derivations. Although certain configurations could theoretically yield multiple inverse solutions, the numerical checks indicated reliable uniqueness for typical workspace configurations.

Workspace and Jacobian Insights Workspace visualization revealed a circular boundary under symmetrical link assumptions, offering clear insight into feasible end-effector positions and orientations. By scrutinizing the kinematic Jacobian across this workspace, we identified how joint velocities map onto end-effector velocities and localized potential singularities. Incorporating the Global Isotropy Index (GII) highlighted where the mechanism operates most uniformly, cautioning against working in regions with poor manipulability. These findings directly inform strategies for path planning and operation near lower-singularity areas.

4.3 Interpretation of Dynamic Modeling Outcomes

Kane's vs. Lagrange's Equations Both Kane's and Lagrange's formulations successfully captured the 3RRP mechanism's dynamic behavior. However, Kane's method offered more compact expressions, leveraging partial velocities to incorporate loop-closure constraints seamlessly. In contrast, the Lagrangian framework provided an intuitive energy-based interpretation but demanded explicit constraint handling via Lagrange multipliers and Baumgarte stabilization. These trade-offs emphasize that the choice of method may hinge on factors such as system complexity, desired symbolic simplicity, and the relevance of energy concepts to subsequent control design.

Simulation Observations Simulations under small forces and torques validated the system's stable and constraint-respecting motion. The prismatic and revolute joints collaborated smoothly, indicating that neither approach to dynamic modeling introduced numerical instabilities or constraint drift under moderate loading. The minor deviations observed can be traced to typical integration tolerances or subtle parameter assumptions. Overall, the 3RRP mechanism demonstrated predictable, controllable responses that underscore its aptitude for planar tasks requiring moderate precision.

4.4 Comparisons and Correlations

Kinematic–Dynamic Consistency A key outcome was the alignment of dynamic simulation results with the kinematic predictions. End-effector trajectories stayed within the computed workspace bounds, and velocities agreed with expected joint-space mappings. Inconsistencies, where present, were nominal and stemmed from solver thresholds rather than from conceptual flaws in the models.

Potential Experimental Benchmarks Although direct experimental validation lies beyond the current scope, the trends discovered align with established literature on planar parallel manipulators. Minor numerical artifacts, such as marginal drift in near-singular regions, mirror the behavior reported when friction, damping, or measurement noise are minimal. These parallels suggest a realistic pathway for future hardware-based experiments to confirm and refine the theoretical and simulated results.

4.5 Practical Implications for 3RRP Mechanism

Design and Control Considerations The synergy between large isotropy regions and straightforward dynamic responses indicates that the 3RRP mechanism is prime for applications such as assembly tasks, pick-and-place operations, and any planar motion tasks needing precision and moderate payload handling. Controller tuning may focus on Jacobian-based strategies, adjusting control gains in areas where the GII drops to ensure robust tracking near singularities.

Real-World Applications Industrial operations—including packaging, inspection, and PCB assembly—benefit from planar parallel manipulators with high dexterity and stiffness. The 3RRP mechanism's combination of revolute and prismatic joints delivers adaptable work envelopes with relatively simple forward/inverse calculations. This simplicity, along with the validated dynamic models, facilitates streamlined hardware integration, making it feasible to deploy basic trajectory-following controllers and potentially incorporate advanced real-time strategies like Jacobian transpose or hybrid force-motion control.

4.6 Reflections and Future Directions

The collective results—covering comprehensive kinematic derivations, isotropy assessments, and validated dynamic models—form a robust analytical and simulation-based platform for the 3RRP mechanism. As technology advances and novel robotic applications arise, the methods presented here can be extended by incorporating realistic joint friction, elastic elements, or more sophisticated control approaches, enabling the 3RRP mechanism to meet higher-precision or higher-speed demands.

4.7 Summary of the Discussion

Overall, the investigation shows that a well-calibrated 3RRP mechanism can achieve accurate planar motion and sustain moderate external disturbances without undermining its constraint structure or workspace reach. By bridging rigorous symbolic derivations (for both kinematics and dynamics) with

simulation verifications, this work establishes a strong technical foundation for future refinements in design, control, and performance optimization.

5 Conclusion

This project provided an in-depth exploration of the 3RRP mechanism's kinematic and dynamic performance. The following key achievements highlight its contributions and potential impact:

- **Robust Kinematic Framework:** Closed-form forward and inverse kinematics were verified through Simulink simulations, ensuring a precise mapping between joint variables and end-effector pose. The Jacobian and Global Isotropy Index (GII) further quantified the mechanism's local dexterity and guided the identification of near-singular regions.
- **Workspace Characterization:** Numerical methods established the largest symmetric workspace for the 3RRP, demonstrating the system's ability to access a wide planar area with full rotational freedom. These findings assist in planning maneuvers that exploit optimal regions of manipulability.
- **Dynamic Modeling via Kane's and Lagrange's Methods:** Equations of motion were derived using both approaches, offering insights into the trade-offs between compactness (Kane's) and energy-focused formulations (Lagrange's). The inclusion of Baumgarte stabilization in the Lagrangian framework emphasized best practices for constraint-enforced simulations.
- **Simulation and Verification:** MATLAB/Simulink implementations validated both the kinematic and dynamic models, revealing stable end-effector responses under small perturbations. Joint velocities remained consistent with theoretical predictions, supporting real-time feasibility for moderate tasks.

Significance and Limitations These results affirm that the 3RRP mechanism is well-suited for planar robotic tasks—ranging from precise assembly to general pick-and-place—thanks to its closed-form kinematics, relatively high isotropy, and stable dynamic responses. While frictionless and rigid-link assumptions simplify the analysis, they limit real-world applicability. Introducing friction, compliance, or high-speed regimes would require more advanced models and possibly higher-end actuators.

Recommendations for Future Work Building on this foundation, several avenues can extend the scope and deepen the realism of the 3RRP study:

- **Controller Design and Optimization:** Implement advanced control laws (e.g., adaptive, robust, or model-predictive) leveraging the validated equations of motion and Jacobian-based velocity mappings.
- **Parametric Sensitivity and Optimization:** Explore variations in link lengths, masses, or prismatic stroke limits to optimize the workspace–dexterity trade-off.
- **Experimental Validation:** Construct a physical prototype or testbed to compare measured data against simulations, informing friction compensation or real-time control tuning.
- **Complex System Integration:** Combine the 3RRP with other planar or spatial mechanisms for multi-axis tasks, applying the same systematic derivation and simulation approach developed here.

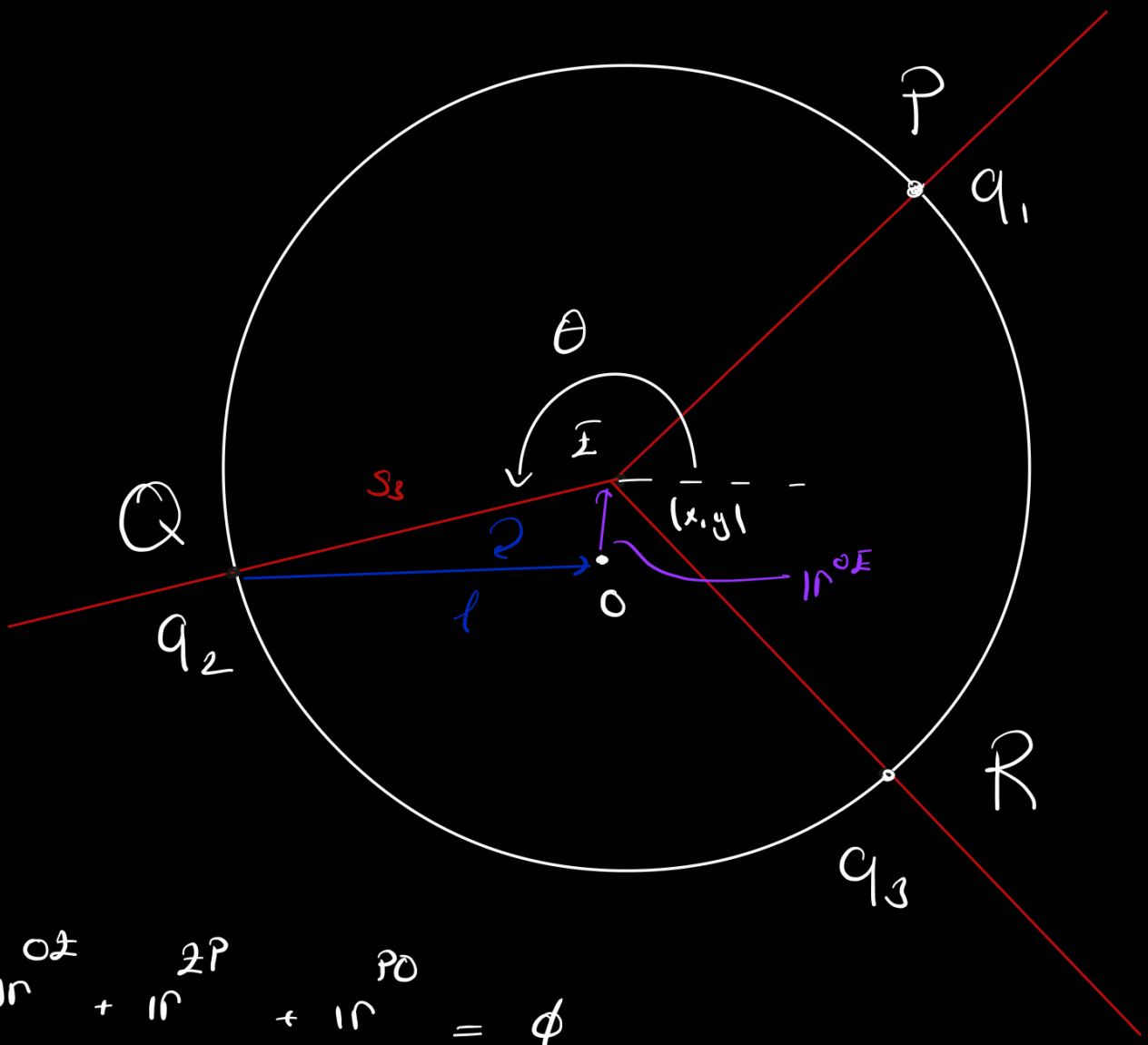
Final Remarks By presenting unified kinematic and dynamic analyses, supported by rigorous symbolic derivations and numerical validations, this report underscores the 3RRP mechanism's potential in achieving precise, robust planar motion. The demonstrated synergy among theoretical modeling, software-based verification, and practical design considerations offers a solid springboard for future research and implementation, bridging academic rigor with industrial relevance.

Bibliography

References

Appendix A

RRP Inverse Kinematic



$$r^{O\Sigma} + r^{\Sigma P} + r^{PO} = \phi$$

known s_1 unknown ℓ known

θ known q_1 unknown

Case (5)

$$r^{O\Sigma} + r^{\Sigma Q} + r^{QO} = \phi$$

known s_2 unknown ℓ known

θ known q_2 unknown

$$r^{O\Sigma} + r^{\Sigma R} + r^{RO} = \phi$$

known s_3 unknown ℓ known

θ known q_3 unknown

$$\frac{\partial \Gamma}{\partial \phi}^{OZ} + \frac{\partial \Gamma}{\partial a_1}^{ZP} + \frac{\partial \Gamma}{\partial b}^{PO} = \phi$$

Loop 1:

case ⑤

$a_1 = a \cdot \hat{a}_1 = s_1 \cdot \hat{1k}_1 \Rightarrow \hat{1k}_1$: arbitrary vector

↳ unknown (a_1)

$1b = b \cdot \hat{1b} = f_1 \cdot \hat{1f}_1 \longrightarrow$ unknown ($\hat{1b}$)

$\phi = c \cdot \hat{\phi} = x\,1n_1 + y\,1n_2$

$c = (x^2 + y^2)^{1/2} \qquad \hat{\phi} = \dfrac{x\,1n_1 + y\,1n_2}{(x^2+y^2)^{1/2}}$

$$s_1 = \left[-(x\,1n_1 + y\,1n_2)\,\hat{1k}_1 - \sqrt{f_1^2 - \left[(x\,1n_1 + y\,1n_2)\cdot(\hat{1k}_1 \times 1n_3)\right]^2} \right]$$

$$f_1 \cdot \hat{f}_1 = -\left[(x\,1n_1 + y\,1n_2)\cdot(\hat{1k}_1 \times 1n_3)\right](\hat{1k}_1 \times 1n_3)$$

$$+ \sqrt{f_1^2 - \left[(x\,1n_1 + y\,1n_2)\cdot(\hat{1k}_1 \times 1n_3)\right]^2}\;\hat{1k}_1$$

$$\underset{\phi}{\overset{OZ}{1\Gamma}} + \underset{a1}{\overset{2Q}{1\Gamma}} + \underset{1b}{\overset{Q0}{1\Gamma}} = \phi$$

Loop 2:

case ⑤

$a1 = a \cdot \hat{a1} = s_2 \cdot 1\hat{f}_1 \Rightarrow 1\hat{f}_1$: anbitrony vector

↳ unknown (a1)

$1b = b \cdot \hat{1b} = f_2 \cdot \$_1 \longrightarrow$ unknown ($\hat{1b}$)

$\phi = c \cdot \hat{\phi} = x\,in_1 + y\,in_2$

$c = (x^2 + y^2)^{1/2} \qquad \hat{\phi} = \dfrac{x\,in_1 + y\,in_2}{(x^2+y^2)^{1/2}}$

$$s_2 = \left[-(x\,in_1 + y\,in_2)\,1\hat{f}_1 - \sqrt{f_2^2 - \left[(x\,in_1 + y\,in_2)\cdot(1\hat{f}_1 \times in_3)\right]^2}\right]$$

$$\underbrace{f_2\,\hat{\$}_1}_{\$_1} = -\left[(x\,in_1 + y\,in_2)\cdot(1\hat{f}_1 \times in_3)\right](1\hat{f}_1 \times in_3)$$

$$+ \sqrt{f_2^2 - \left[(x\,in_1 + y\,in_2)\cdot(1\hat{f}_1 \times in_3)\right]^2}\;\hat{f}1_1$$

$$\frac{\overrightarrow{OZ}}{\phi} + \frac{\overrightarrow{ZR}}{a_1} + \frac{\overrightarrow{RO}}{l_b} = \phi$$

Loop 3:

case (5)

$a_1 = a \cdot \hat{a_1} = s_3 \cdot \hat{m_1} \Rightarrow \hat{m_1}$: arbitrary vector

s_3 → unknown (a_1)

$l_b = b \cdot \hat{l_b} = f_3 \cdot V_1 \longrightarrow$ unknown ($\hat{l_b}$)

$\phi = c \cdot \hat{\phi} = x\hat{n}_1 + y\hat{n}_2$

$$c = (x^2 + y^2)^{1/2} \qquad \hat{\phi} = \frac{x\hat{n}_1 + y\hat{n}_2}{(x^2+y^2)^{1/2}}$$

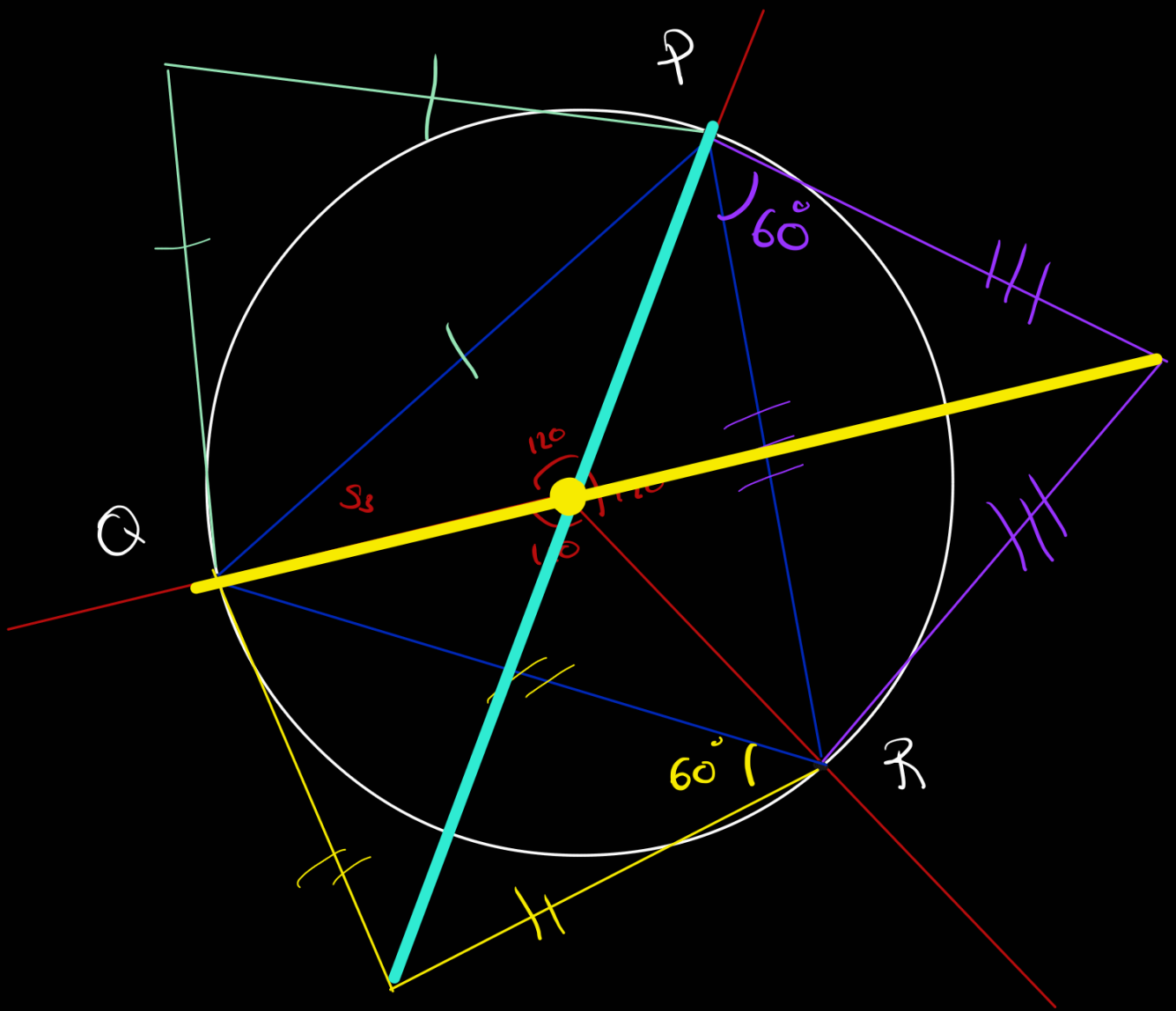
$$s_3 = \left[-(x\hat{n}_1 + y\hat{n}_2)\hat{m}_1 - \sqrt{f_3^2 - \left[(x\hat{n}_1 + y\hat{n}_2)\cdot(\hat{m}_1 \times \hat{n}_3)\right]^2} \right]$$

$$f_3 V_1 = -\left[(x\hat{n}_1 + y\hat{n}_2)\cdot(\hat{m}_1 \times \hat{n}_3)\right](\hat{m}_1 \times \hat{n}_3)$$

$$+ \sqrt{f_3^2 - \left[(x\hat{n}_1 + y\hat{n}_2)\cdot(\hat{m}_1 \times \hat{n}_3)\right]^2}\ \hat{m}_1$$

After organizing these values we can find all intermediate variables S_1, S_2, S_3 and using these variables we can find q_1, q_2, q_3

Forword Kinematic

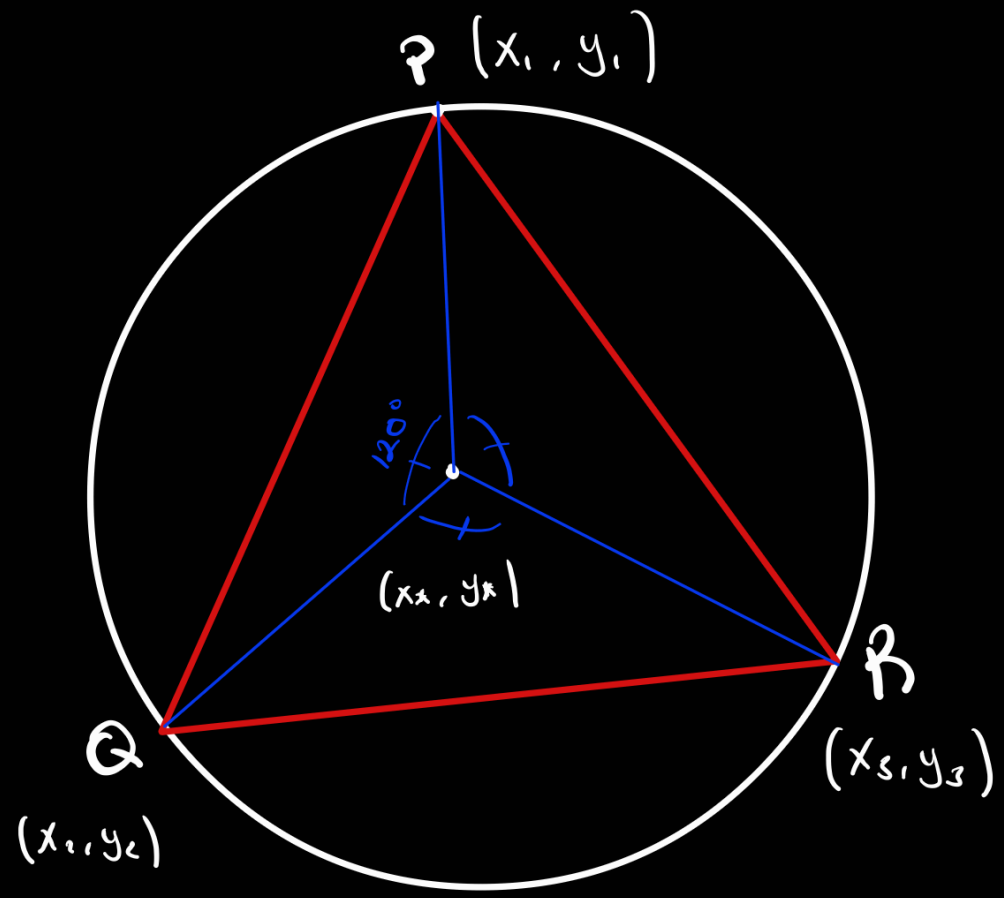


we know q_1, q_2, q_3

that means we know the location of Q, P, R

We can use Fermat - Toricelli for this

$m_1 = m_2 = m_3 = 1$



$$x_* = \frac{k_1 k_2 k_3}{2\sqrt{3}\, S\, d}\left(\frac{x_1}{k_1} + \frac{x_2}{k_2} + \frac{x_3}{k_3}\right)$$

$$y_* = \frac{k_1 k_2 k_3}{2\sqrt{3}\, S\, d}\left(\frac{y_1}{k_1} + \frac{y_2}{k_2} + \frac{y_3}{k_3}\right)$$

$$x_1 = l\cos(q_1) \qquad y_1 = l\sin(q_1)$$

$$x_2 = l\cos(q_2) \qquad y_2 = l\sin(q_2)$$

$$x_3 = l\cos(q_3) \qquad y_3 = l\sin(q_3)$$

$$k_1 = \frac{\sqrt{3}}{2}\left(r_{12}^2 + r_{13}^2 - r_{23}^2\right) + S$$

$$k_2 = \frac{\sqrt{3}}{2}\left(r_{23}^2 + r_{32}^2 - r_{13}^2\right) + S$$

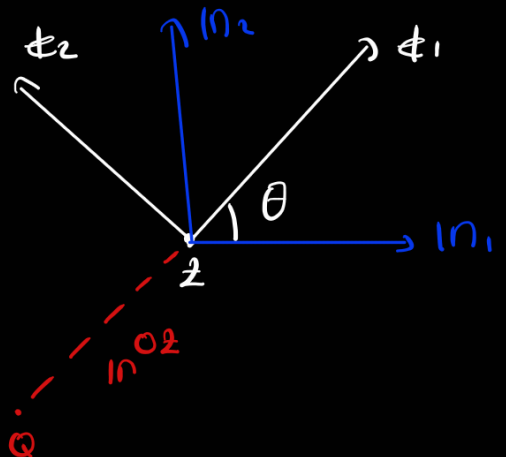
$$k_3 = \frac{\sqrt{3}}{2}\left(r_{13}^2 + r_{23}^2 - r_{12}^2\right) + S$$

$$d = \frac{1}{\sqrt{3}}\left(k_1 + k_2 + k_3\right)$$

$$S = \left| x_1 y_2 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_3 y_2 - x_2 y_1 \right|$$

---

for theta:



$\hat{t}_1 \longrightarrow$ some direction with r^{Q2}

↳ some slope: m

↳ passes through 2 (x, y known)

$\hat{t}_2 \longrightarrow \perp$ to r^{Q2}

↳ slope: $-\frac{1}{m}$

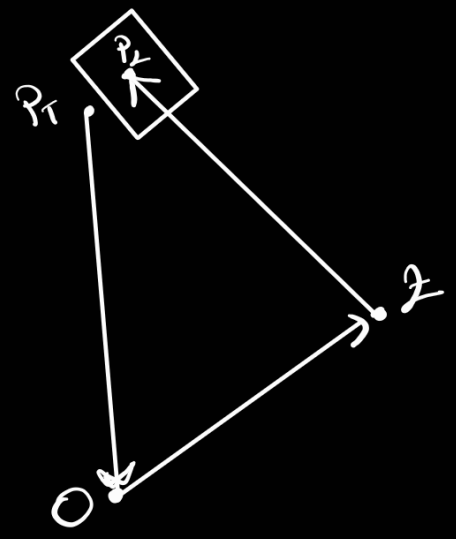
↳ passes through 2 (x, y known)

$$\theta = \operatorname{atan2}\left(\frac{\hat{t}_2 \cdot \hat{n}_2}{\hat{t}_1 \cdot \hat{n}_1}\right)$$

$\Rightarrow$ we can form 2 equation for each to solve 2 unknown for each:

$$y = mx + b$$

Motion Level Forward Kinematics



P_T fixed in T

P_k fixed in K

$$\frac{{}^{N}d}{dt}\left({}^{O2}r + {}^{2P_k}r + {}^{P_kP_T}r + {}^{P_TO}r = 0 \right)$$

$$\frac{{}^{N}d}{dt}{}^{O2}r + \frac{{}^{N}d}{dt}{}^{2P_k}r + \frac{{}^{N}d}{dt}{}^{P_kP_T}r + \frac{{}^{N}d}{dt}{}^{P_TO}r = \phi$$

$$\frac{{}^{N}d}{dt}{}^{O2}r = {}^{N}V^{2} = \dot{x}\,{}^{N}n_1 + \dot{y}\,{}^{N}n_2$$

$$\frac{{}^{N}d}{dt}{}^{2P_k}r = \frac{{}^{K}d}{dt}{}^{2P_k}r + {}^{N}\omega^{K} \times {}^{2P_k}r$$

$\frac{{}^{K}d}{dt}{}^{2P_k}r \to 0$

$$\frac{{}^{N}d}{dt}{}^{P_kP_T}r = \frac{{}^{K}d}{dt}{}^{P_kP_T}r + {}^{N}\omega^{K} \times {}^{P_kP_T}r = {}^{K}V^{P_T} + {}^{N}\omega^{K} \times {}^{P_kP_T}r$$

${}^{K}V^{P_T} \to \dot{s}_1 \cdot \hat{k}_1$

$\frac{{}^{K}d}{dt}{}^{P_kP_T}r \to {}^{K}V^{P_T}$

$$\frac{{}^{N}d}{dt}{}^{P_TO}r = \frac{{}^{T}d}{dt}{}^{P_TO}r + {}^{N}\omega^{T} \times {}^{P_TO}r$$

$\frac{{}^{T}d}{dt}{}^{P_TO}r \to 0$

$$\tilde{V}^{2}_{/} + \overset{\sim}{\omega}^{k} \times \overset{2P_k}{r} + \overset{k}{V}^{P_T}_{/} + \overset{\sim}{\omega}^{k} \times \overset{P_k P_T}{r} + \overset{\sim}{\omega}^{T} \times \overset{P_T O}{r} = \phi$$

$$\overset{\sim}{\omega}^{k} = \dot{\theta}\, \hat{n}_3 \qquad \overset{2P_k}{r} = s_1\, \hat{l}_1$$

$$\overset{\sim}{\omega}^{T} = \dot{q}_1\, \hat{n}_3 \qquad \overset{P_k P_T}{r} \doteq 0$$

$$\overset{P_T O}{r} = \rho_1\, \hat{t}_1 = \rho_1 \left(\cos(q_1)\, \hat{n}_1 + \sin(q_1)\, \hat{n}_2 \right)$$

$$\overset{\sim}{\frac{d}{dt}} \left(\overset{O2}{r} + \overset{2Q_2}{r} + \overset{Q_2 Q_3}{r} + \overset{Q_3 O}{r} = \phi \right)$$

$$\overset{\sim}{\frac{d}{dt}} \overset{O2}{r} = \tilde{V}^{2}_{/} = \dot{x}\, \hat{n}_1 + \dot{y}\, \hat{n}_2$$

$$\overset{\sim}{\frac{d}{dt}} \overset{2Q_2}{r} = \overset{2}{\frac{d}{dt}} \overset{2Q_2}{r} + \overset{\sim}{\omega}^{2} \times \overset{2Q_2}{r}$$

$\overset{2}{\frac{d}{dt}} \overset{2Q_2}{r} \rightarrow 0$

$$\overset{\sim}{\frac{d}{dt}} \overset{Q_2 Q_3}{r} = \overset{2}{\frac{d}{dt}} \overset{Q_2 Q_3}{r} + \overset{\sim}{\omega}^{2} \times \overset{Q_2 Q_3}{r} = \overset{2}{V}^{Q_3}_{/} + \overset{\sim}{\omega}^{2} \times \overset{Q_2 Q_3}{r}$$

$\overset{2}{V}^{Q_3}_{/} \rightarrow \dot{s}_2\, \hat{l}_1$

$$\overset{\sim}{\frac{d}{dt}} \overset{Q_3 O}{r} = \overset{3}{\frac{d}{dt}} \overset{Q_3 O}{r} + \overset{\sim}{\omega}^{3} \times \overset{Q_3 O}{r}$$

$\overset{3}{\frac{d}{dt}} \overset{Q_3 O}{r} \rightarrow 0$

$$\tilde{V}^{2} + \tilde{\omega} \times r^{2Q_2} + \tilde{V}^{2} + \tilde{\omega} \times r^{Q_2Q_3} + \tilde{V}^{3} + \tilde{\omega} \times r^{Q_3O} = 0$$

$$\tilde{\omega}^{2} = \dot{\theta}\, \hat{n}_3 \qquad r^{2Q_2} = s_2 \cdot \hat{f}_1$$

$$\tilde{\omega}^{3} = \dot{q}_2\, \hat{n}_3 \qquad r^{Q_2Q_3} \doteq 0$$

$$r^{Q_3O} = f_2\, \hat{s}_1 = f_2 \left(\cos(q_2)\, \hat{n}_1 + \sin(q_2)\, \hat{n}_2 \right)$$

$$\tilde{\frac{d}{dt}} \left(r^{O2} + r^{2B_m} + r^{B_mB_v} + r^{B_vO} = 0 \right)$$

$$\tilde{\frac{d}{dt}}\, r^{O2} = \tilde{V}^{2} = \dot{x}\, \hat{n}_1 + \dot{y}\, \hat{n}_2$$

$$\tilde{\frac{d}{dt}}\, r^{2B_m} \overset{m}{=} \frac{d}{dt}\, r^{2B_m} + \tilde{\omega}^{m} \times r^{2B_m}$$

$$\tilde{\frac{d}{dt}}\, r^{B_mB_v} \overset{m}{=} \frac{d}{dt}\, r^{B_mB_v} + \tilde{\omega}^{m} \times r^{B_mB_v} = \tilde{V}^{B_v} + \tilde{\omega}^{m} \times r^{B_mB_v}$$

$$\tilde{\frac{d}{dt}}\, r^{B_mO} \overset{V}{=} \frac{d}{dt}\, r^{B_vO} + \tilde{\omega}^{V} \times r^{B_vO}$$

$$\tilde{V}^{2}_{\mathcal{A}} + \tilde{\omega}^{\mathcal{A}}_{\mathcal{M}} \times {}^{2R_M}r + \tilde{V}^{R_M}_{\mathcal{A}} + \tilde{\omega}^{\mathcal{A}}_{R_M} \times {}^{R_M R_V}r + \tilde{\omega}^{R_M}_{R_V} \times {}^{R_V O}r = 0$$

$$\tilde{\omega}^{\mathcal{M}} = \dot{\theta}\, {}^{\mathcal{M}}n_3 \qquad {}^{2R_M}r = s_3 \cdot {}^{\mathcal{M}}\hat{n}_1$$

$$\tilde{\omega}^{R_V} = \dot{q}_3\, {}^{\mathcal{M}}n_3 \qquad {}^{R_M R_V}r \doteq 0$$

$${}^{R_V O}r = \rho_3 V_1 = \rho \left(\cos(q_3)\, {}^{\mathcal{M}}n_1 + \sin(q_3)\, {}^{\mathcal{M}}n_2 \right)$$

Acceleration Level Forward Kin.

$$\frac{d}{dt}\left({}^{\sim}V^{2} + {}^{\sim}\omega^{k} \times r^{2P_k} + {}^{k}V^{P_T} + {}^{\sim}\omega^{k} \times r^{P_k P_T} + {}^{\sim}\omega^{T} \times r^{P_T 0} = 0 \right)$$

$$\frac{{}^{\sim}d}{dt}\, {}^{\sim}V^{2} = {}^{\sim}a^{2}$$

$$\frac{{}^{\sim}d}{dt}\, {}^{\sim}\omega^{k} \times r^{2P_k} = {}^{\sim}\alpha^{k} \times r^{2P_k}$$

$${}^{\sim}\omega^{k} \times \frac{{}^{\sim}d}{dt}\, r^{2P_k} = {}^{\sim}\omega^{k} \times \left(\frac{{}^{k}d}{dt}\, r^{2P_k} + {}^{\sim}\omega^{k} \times r^{2P_k} \right)$$

$$\frac{{}^{\sim}d}{dt}\, {}^{k}V^{P_T} = {}^{k}a^{P_T} + {}^{\sim}\omega^{k} \times {}^{k}V^{P_T}$$

$$\frac{{}^{\sim}d}{dt}\, {}^{\sim}\omega^{k} \times r^{P_k P_T} = {}^{\sim}\alpha^{k} \times r^{P_k P_T}$$

$${}^{\sim}\omega^{k} \times \frac{{}^{\sim}d}{dt}\, r^{P_k P_T} = {}^{\sim}\omega^{k} \times \left({}^{k}V^{P_T} \times r^{P_k P_T} \right)$$

$$\frac{{}^{\sim}d}{dt}\, {}^{\sim}\omega^{T} \times r^{P_T 0} = {}^{\sim}\alpha^{T} \times r^{P_T 0}$$

$${}^{\sim}\omega^{T} \times \frac{{}^{\sim}d}{dt}\, r^{P_T 0} = {}^{\sim}\omega^{T} \times \left({}^{T}V^{P_T 0} + {}^{\sim}\omega^{T} \times r^{P_T 0} \right)$$

$${}^{\sim}a^{2} + {}^{\sim}\alpha^{k} \times r^{2P_k} + {}^{\sim}\omega^{k} \times {}^{\sim}\omega^{k} \times r^{2P_k} + {}^{k}a^{P_T} + 2\, {}^{\sim}\omega^{k} \times {}^{k}V^{P_T}$$

$$+ {}^{\sim}\alpha^{k} \times r^{P_T 0} + {}^{\sim}\omega^{T} \times {}^{\sim}\omega^{T} \times r^{P_T 0}$$

$$\frac{d}{dt}\left(\vec{V}_{2} + \vec{\omega}_{2} \times \vec{r}_{2Q_2} + \vec{V}_{Q_2Q_3} + \vec{\omega}_{S} \times \vec{r}_{Q_3O} = \phi \right)$$

$$\frac{d}{dt}\vec{V}_{2} = \vec{a}_{2}$$

$$\frac{d}{dt}\vec{\omega}_{2} \times \vec{r}_{2Q_2} = \vec{\alpha}_{2} \times \vec{r}_{2Q_2}$$

$$\vec{\omega}_{2} \times \frac{d}{dt}\vec{r}_{2Q_2} = \vec{\omega}_{2} \times \left(\frac{d}{dt}\vec{r}_{2Q_2} + \vec{\omega}_{2} \times \vec{r}_{2Q_2} \right)$$

$$\frac{d}{dt}\vec{V}_{Q_2Q_3} = \vec{a}_{Q_2Q_3} + \vec{\omega}_{2} \times \vec{V}_{Q_2Q_3}$$

$$\frac{d}{dt}\vec{\omega}_{2} \times \vec{r}_{Q_2Q_3} = \vec{\alpha}_{2} \times \vec{r}_{Q_2Q_3}$$

$$\vec{\omega}_{2} \times \frac{d}{dt}\vec{r}_{Q_2Q_3} = \vec{\omega}_{2} \times \left(\vec{V}_{Q_2Q_3} \times \vec{r}_{Q_2Q_3} \right)$$

$$\frac{d}{dt}\vec{\omega}_{S} \times \vec{r}_{Q_3O} = \vec{\alpha}_{S} \times \vec{r}_{Q_3O}$$

$$\vec{\omega}_{S} \times \frac{d}{dt}\vec{r}_{Q_3O} = \vec{\omega}_{S} \times \left(\frac{d}{dt}\vec{r}_{Q_3O} + \vec{\omega}_{S} \times \vec{r}_{Q_3O} \right)$$

$$\vec{a}_{2} + \vec{\alpha}_{2} \times \vec{r}_{2Q_2} + \vec{\omega}_{2} \times \vec{\omega}_{2} \times \vec{r}_{2Q_2} + \vec{a}_{Q_2} + \vec{\omega}_{2} \times \vec{V}_{Q_2Q_3} + \vec{\alpha}_{2} \times \vec{r}_{Q_2Q_3} + \vec{\omega}_{2} \times \left(\vec{\omega}_{2} \times \vec{r}_{Q_2Q_3} \right) + \vec{a}_{Q_3} + \vec{\omega}_{S} \times \vec{V}_{Q_3O} + \vec{\alpha}_{S} \times \vec{r}_{Q_3O} + \vec{\omega}_{S} \times \left(\vec{\omega}_{S} \times \vec{r}_{Q_3O} \right) = 0$$

$$\frac{d}{dt}\left(\tilde{V}^{z} + \tilde{\omega}^{m} \times r^{2R_m} + V^{v} + \omega^{R_m R_v} \times r^{v} + \tilde{\omega}^{v} \times r^{R_v O} = \phi \right)$$

$$\frac{\tilde{d}}{dt} \tilde{V}^{z} = \tilde{a}^{z}$$

$$\frac{\tilde{d}}{dt} \tilde{\omega}^{m} \times r^{2R_m} = \tilde{\alpha}^{m} \times r^{2R_m}$$

$$\tilde{\omega}^{m} \times \frac{\tilde{d}}{dt} r^{2R_m} = \tilde{\omega}^{m} \times \left(\frac{d}{dt} r^{2R_m} + \tilde{\omega}^{m} \times r^{2R_m} \right)$$

$$\frac{\tilde{d}}{dt} V^{R_v} = a^{R_v} + \tilde{\omega}^{m} \times V^{R_v}$$

$$\frac{\tilde{d}}{dt} \tilde{\omega}^{R_m R_v} \times r^{R_m R_v} = \tilde{\alpha}^{m} \times r^{R_m R_v}$$

$$\tilde{\omega}^{m} \times \frac{\tilde{d}}{dt} r^{R_m R_v} = \tilde{\omega}^{m} \times \left(V^{R_v} \times r^{R_m R_v} \right)$$

$$\frac{\tilde{d}}{dt} \tilde{\omega}^{v} \times r^{R_v O} = \tilde{\alpha}^{v} \times r^{R_v O}$$

$$\tilde{\omega}^{v} \times \frac{\tilde{d}}{dt} r^{R_v O} = \tilde{\omega}^{v} \times \left(\frac{d}{dt} r^{R_v O} + \tilde{\omega}^{v} \times r^{R_v O} \right)$$

$$\tilde{a}^{z} + \tilde{\alpha}^{m} \times r^{2R_m} + \tilde{\omega}^{m} \times \tilde{\omega}^{m} \times r^{2R_m} + a^{k R_v} +$$

$$2\,\tilde{\omega}^{m} \times V^{R_v} + \tilde{\alpha}^{m} \times r^{R_v O} + \tilde{\omega}^{v} \times \tilde{\omega}^{v} \times r^{R_v O}$$

we can solve these equations together to find $\ddot{x}$, $\ddot{y}$, $\ddot{\theta}$ and also $\ddot{s}_1$, $\ddot{s}_2$, $\ddot{s}_3$

for velocity and acc level inverse kinematic we can use these values and Jacobian that we got from Autolev

Appendix B: MATLAB Code for Workspace Calculation

The following MATLAB code was implemented to compute the largest symmetric workspace of the 3RRP mechanism. The function utilizes parallel loops for efficient computation of all reachable end-effector positions and visualizes the resulting workspace.

```
function visualize3RRPWorkspaceParallel()
    % Parameters
    L = 200; % Length of each link in mm
    r = L;   % Symmetrical link length assumption
    num_points = 100; % Number of points for q1, q2, q3

    % Range of joint angles (in radians)
    q1_values = linspace(0, 2*pi, num_points);
    q2_values = linspace(0, 2*pi, num_points);
    q3_values = linspace(0, 2*pi, num_points);

    % Initialize storage for workspace positions
    all_x = [];
    all_y = [];

    % Parallel computation using nested parfor loops
    parfor i = 1:length(q1_values)
        local_x = []; % Local storage for this worker
        local_y = [];
        for j = 1:length(q2_values)
            for k = 1:length(q3_values)
                q1 = q1_values(i);
                q2 = q2_values(j);
                q3 = q3_values(k);

                % Compute forward kinematics
                [x, y, ~] = calculateEndEffectorPosition3RRP(q1, q2, q3, r);

                % Store results locally
                local_x = [local_x, x];
                local_y = [local_y, y];
            end
        end
        % Append local results to global arrays
        all_x = [all_x, local_x];
        all_y = [all_y, local_y];
    end

    % Plot the workspace
    figure;
    plot(all_x, all_y, 'b.', 'MarkerSize', 5);
    xlabel('X position (mm)');
    ylabel('Y position (mm)');
    title('Largest Symmetric Workspace of 3RRP Mechanism (Nested parfor)');
    axis equal;
end

function [x, y, theta] = calculateEndEffectorPosition3RRP(q1, q2, q3, r)
    % Forward kinematics for the 3RRP mechanism
    % Input: q1, q2, q3 (joint angles in radians), r (link length)
    % Output: x, y (end effector position), theta (end effector orientation)

    % Calculate intermediate variables
```

```
    c11 = r * cos(q1); c12 = r * sin(q1);
    c21 = r * cos(q2); c22 = r * sin(q2);
    c31 = r * cos(q3); c32 = r * sin(q3);

    K = c12 + c32 + sqrt(3)*c31 - 2*c22 - sqrt(3)*c11;
    L = c11 + c31 + sqrt(3)*c12 - 2*c21 - sqrt(3)*c32;
    M = L * (L - sqrt(3)*K) * c12 - L * (K + sqrt(3)*L) * c11 - ...
        (L - sqrt(3)*K) * (L*c22 - K*c21);

    % Forward kinematics equations
    x = -M  / ((K^2 + L^2) * sqrt(3));
    y = c22 - (K/L) * c21 - (K * M) / (L * sqrt(3) * (K^2 + L^2));
    theta = atan2(K, L);
end
```

Appendix C: Detailed Jacobian Matrix

This appendix provides the detailed symbolic expression of the kinematic Jacobian matrix for the 3RRP mechanism. The Jacobian matrix relates the joint velocities $(\dot{q}_1, \dot{q}_2, \dot{q}_3)$ to the end-effector's velocities $(\dot{x}, \dot{y}, \dot{\theta})$.

Listing 1: Detailed Symbolic Jacobian Matrix for 3RRP Mechanism

```
JACOBIAN[1,1] = -0.5773503*L*(SIN(q1)*(SIN(theta)+1.732051*COS(theta))
    - COS(q1)*(1.732051*SIN(theta)-COS(theta)))
    *(2*s2*(COS(theta)+1.732051*SIN(theta))
    + s3*COS(theta)*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))

JACOBIAN[1,2] = -0.5773503*L*COS(q2-theta)
    *(s3*(SIN(theta)-1.732051*COS(theta))^2
    *(1.732051*SIN(theta)-COS(theta))
    + (COS(theta)+1.732051*SIN(theta))
    *(s1*(SIN(theta)+1.732051*COS(theta))^2
    + s1*(1.732051*SIN(theta)-COS(theta))^2
    + s3*(COS(theta)+1.732051*SIN(theta))
    *(1.732051*SIN(theta)-COS(theta))))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))

JACOBIAN[1,3] = -0.5773503*L*(SIN(q3)*(SIN(theta)-1.732051*COS(theta))
    + COS(q3)*(COS(theta)+1.732051*SIN(theta)))
    *(2*s2*(1.732051*SIN(theta)-COS(theta))
    - s1*COS(theta)*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))

JACOBIAN[2,1] = -0.5773503*L*(SIN(q1)*(SIN(theta)+1.732051*COS(theta))
    - COS(q1)*(1.732051*SIN(theta)-COS(theta)))
    *(2*s2*(SIN(theta)-1.732051*COS(theta))
    + s3*SIN(theta)*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
```

```
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))

JACOBIAN[2,2] = 0.5773503*L*COS(q2-theta)
    *(s3*(SIN(theta)+1.732051*COS(theta))*(SIN(theta)-1.732051*COS(theta))^2
    + s3*(SIN(theta)+1.732051*COS(theta))*(COS(theta)+1.732051*SIN(theta))^2
    - s1*(SIN(theta)-1.732051*COS(theta))
    *((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))

JACOBIAN[3,3] = 2*L*(SIN(q3)*(SIN(theta)-1.732051*COS(theta))
    + COS(q3)*(COS(theta)+1.732051*SIN(theta)))
    /(s3*((SIN(theta)-1.732051*COS(theta))^2
    + (COS(theta)+1.732051*SIN(theta))^2)
    + s1*((SIN(theta)+1.732051*COS(theta))^2
    + (1.732051*SIN(theta)-COS(theta))^2)
    + 1.154701*s2*((SIN(theta)+1.732051*COS(theta))
    *(COS(theta)+1.732051*SIN(theta))
    + (SIN(theta)-1.732051*COS(theta))
    *(1.732051*SIN(theta)-COS(theta))))
```

Appendix D: MATLAB Code for Global Isotropy Index (GII)

This appendix contains the MATLAB implementation used to compute and visualize the Global Isotropy Index (GII) for the 3RRP mechanism. The script calculates the GII by analyzing the singular values of the Jacobian matrix across the workspace and visualizes the isotropy distribution.

Listing 2: MATLAB Code for GII Calculation and Visualization

```
%% ------------------------ Initialization ----------------------------
clear; clc;
% Define numerical values for constants
r = 300;  % Example value; replace with actual value as needed

% Assign default positive values to s1, s2, s3
% These can be adjusted based on system requirements
s1 = 1.0;
s2 = 1.0;
s3 = 1.0;

% Define joint variable ranges (in radians)
q1_min = 0; q1_max = 2*pi;
q2_min = 0; q2_max = 2*pi;
q3_min = 0; q3_max = 2*pi;

% Define number of samples per joint
num_samples = 60;  % Adjust for desired resolution and computational resources

% Generate joint variable samples
q1_samples = linspace(q1_min, q1_max, num_samples);
q2_samples = linspace(q2_min, q2_max, num_samples);
q3_samples = linspace(q3_min, q3_max, num_samples);

% Total number of joint combinations
total_combinations = num_samples^3;

% Preallocate arrays to store singular values
sigma_min_vals = zeros(total_combinations, 1);
sigma_max_vals = zeros(total_combinations, 1);

% Initialize tracking variables
min_sigma_min = Inf;
max_sigma_max = -Inf;

% Define constants for precision
sqrt3 = 1.7320508075688772;        % Approximation of sqrt(3)
inv_sqrt3 = 0.5773502691896257;    % 1/sqrt(3)
twice_inv_sqrt3 = 1.1547005383792517; % 2/sqrt(3)

%% --------------------- Parallel Computation -------------------------

% Create a parallel pool if not already open
if isempty(gcp('nocreate'))
    parpool; % Uses default settings; adjust 'parpool' parameters as needed
end

% Start parallel loop
parfor idx = 1:total_combinations
    % Convert linear index to subscript indices
    [i, j, k] = ind2sub([num_samples, num_samples, num_samples], idx);

    % Retrieve joint angles
    q1 = q1_samples(i);
    q2 = q2_samples(j);
    q3 = q3_samples(k);
```

```
57
58      %% -------------------- Forward Kinematics -----------------------
59
60      % Compute intermediate cosine and sine values
61      c11 = r * cos(q1);
62      c12 = r * sin(q1);
63      c21 = r * cos(q2);
64      c22 = r * sin(q2);
65      c31 = r * cos(q3);
66      c32 = r * sin(q3);
67
68      % Define K, L, and M based on forward kinematics
69      K = c12 + c32 + sqrt3 * c31 - 2 * c22 - sqrt3 * c11;
70      L = c11 + c31 + sqrt3 * c12 - 2 * c21 - sqrt3 * c32;
71      M = L * (L - sqrt3 * K) * c12 - L * (K + sqrt3 * L) * c11 - (L - sqrt3 *
            K) * (L * c22 - K * c21);
72
73      % Calculate x, y, and theta
74      denom_xy = sqrt3 * (K^2 + L^2);
75      if denom_xy == 0
76          % Avoid division by zero; assign NaN and continue
77          sigma_min_vals(idx) = NaN;
78          sigma_max_vals(idx) = NaN;
79          continue;
80      end
81      x = -M / denom_xy;
82      y = c22 - (K / L) * c21 - (K * M) / (sqrt3 * L * (K^2 + L^2));
83      theta = atan2(K, L);
84
85      %% ---------------------- Jacobian Calculation ----------------------
86
87      % Precompute sine and cosine of theta
88      sin_theta = sin(theta);
89      cos_theta = cos(theta);
90
91      % Precompute common terms
92      term1 = sin_theta + sqrt3 * cos_theta;
93      term2 = 1.732051 * sin_theta - cos_theta;
94      term3 = sin_theta - sqrt3 * cos_theta;
95      term4 = cos_theta + sqrt3 * sin_theta;
96
97      % Compute denominator for all Jacobian entries
98      denominator = s3 * (term3^2 + term4^2) + ...
99                    s1 * ( (sin_theta + sqrt3 * cos_theta)^2 + (1.732051 *
                          sin_theta - cos_theta)^2 ) + ...
100                   twice_inv_sqrt3 * s2 * ( (term1 * (cos_theta + sqrt3 *
                          sin_theta)) + (term3 * (1.732051 * sin_theta -
                          cos_theta)) );
101
102     % Check for zero denominator to avoid division by zero
103     if denominator == 0
104         sigma_min_vals(idx) = NaN;
105         sigma_max_vals(idx) = NaN;
106         continue;
107     end
108
109     %% Compute Jacobian Entries
110
111     % J(1,1)
112     numerator_J11 = -inv_sqrt3 * L * (sin(q1) * term1 - cos(q1) * (1.732051 *
            sin_theta - cos_theta)) * ...
113                     (2 * s2 * (cos_theta + sqrt3 * sin_theta) + s3 *
                            cos_theta * (term3^2 + term4^2));
```

```
    J11 = numerator_J11 / denominator;

    % J(1,2)
    numerator_J12 = -inv_sqrt3 * L * cos(q2 - theta) * ...
                    (s3 * term3^2 * (1.732051 * sin_theta - cos_theta) + ...
                     (cos_theta + sqrt3 * sin_theta) * (s1 * (term1)^2 + s1 *
                         (1.732051 * sin_theta - cos_theta)^2 + s3 * term4 *
                         (1.732051 * sin_theta - cos_theta)));
    J12 = numerator_J12 / denominator;

    % J(1,3)
    numerator_J13 = -inv_sqrt3 * L * (sin(q3) * term3 + cos(q3) * term4) * ...
                    (2 * s2 * (1.732051 * sin_theta - cos_theta) - s1 *
                        cos_theta * (term1^2 + (1.732051 * sin_theta -
                        cos_theta)^2));
    J13 = numerator_J13 / denominator;

    % J(2,1)
    numerator_J21 = -inv_sqrt3 * L * (sin(q1) * term1 - cos(q1) * (1.732051 *
        sin_theta - cos_theta)) * ...
                    (2 * s2 * term3 + s3 * sin_theta * (term3^2 + term4^2));
    J21 = numerator_J21 / denominator;

    % J(2,2)
    numerator_J22 = inv_sqrt3 * L * cos(q2 - theta) * ...
                    (s3 * term1 * term3^2 + s3 * term1 * term4^2 - s1 * term3
                        * (term1^2 + (1.732051 * sin_theta - cos_theta)^2));
    J22 = numerator_J22 / denominator;

    % J(2,3)
    numerator_J23 = inv_sqrt3 * L * (sin(q3) * term3 + cos(q3) * term4) * ...
                    (2 * s2 * (sin_theta + sqrt3 * cos_theta) + s1 *
                        sin_theta * (term1^2 + (1.732051 * sin_theta -
                        cos_theta)^2));
    J23 = numerator_J23 / denominator;

    % J(3,1)
    numerator_J31 = 2 * L * (sin(q1) * term1 - cos(q1) * (1.732051 * sin_theta
        - cos_theta));
    J31 = numerator_J31 / denominator;

    % J(3,2)
    numerator_J32 = -twice_inv_sqrt3 * L * cos(q2 - theta) * ...
                    (term1 * (cos_theta + sqrt3 * sin_theta) + term3 *
                        (1.732051 * sin_theta - cos_theta));
    J32 = numerator_J32 / denominator;

    % J(3,3)
    numerator_J33 = 2 * L * (sin(q3) * term3 + cos(q3) * term4);
    J33 = numerator_J33 / denominator;

    % Assemble the Jacobian matrix
    J_num = [J11, J12, J13;
             J21, J22, J23;
             J31, J32, J33];

    %% -------------------- Singular Value Decomposition ----------------

    % Check for invalid Jacobian entries
    if any(isnan(J_num), 'all') || any(isinf(J_num), 'all')
        sigma_min_vals(idx) = NaN;
        sigma_max_vals(idx) = NaN;
        continue;
```

```
    end

    % Perform Singular Value Decomposition
    try
        S = svd(J_num);
        sigma_min_vals(idx) = min(S);
        sigma_max_vals(idx) = max(S);
    catch
        % In case SVD fails, assign NaN
        sigma_min_vals(idx) = NaN;
        sigma_max_vals(idx) = NaN;
    end
end

%% ---------------------- Post-Processing -----------------------------

% Remove NaN entries resulting from invalid Jacobians
valid_indices = ~isnan(sigma_min_vals) & ~isnan(sigma_max_vals);
valid_sigma_min = sigma_min_vals(valid_indices);
valid_sigma_max = sigma_max_vals(valid_indices);

% Ensure there are valid entries to compute GII
if isempty(valid_sigma_min) || isempty(valid_sigma_max)
    error('No valid Jacobian matrices were found. Check the system parameters
        and joint ranges.');
end

% Calculate min_sigma_min and max_sigma_max
min_sigma_min = min(valid_sigma_min);
max_sigma_max = max(valid_sigma_max);

% Calculate GII
GII = min_sigma_min / max_sigma_max;

% Display the Global Isotropy Index
fprintf('Global Isotropy Index (GII): %.4f\n', GII);

%% ---------------------- Workspace Visualization --------------------

% Preallocate arrays for workspace positions
x_workspace = zeros(length(valid_sigma_min), 1);
y_workspace = zeros(length(valid_sigma_min), 1);

% Recompute x and y for valid configurations
parfor idx = 1:length(valid_sigma_min)
    % Retrieve the linear index of the valid configuration
    original_idx = find(valid_indices, 1) + idx - 1;

    % Convert linear index to subscript indices
    [i, j, k] = ind2sub([num_samples, num_samples, num_samples],
        find(valid_indices, 1, 'first') + idx - 1);

    % Retrieve joint angles
    q1 = q1_samples(i);
    q2 = q2_samples(j);
    q3 = q3_samples(k);

    % Compute intermediate cosine and sine values
    c11 = r * cos(q1);
    c12 = r * sin(q1);
    c21 = r * cos(q2);
    c22 = r * sin(q2);
    c31 = r * cos(q3);
```

```
    c32 = r * sin(q3);

    % Define K, L, and M based on forward kinematics
    K = c12 + c32 + sqrt3 * c31 - 2 * c22 - sqrt3 * c11;
    L = c11 + c31 + sqrt3 * c12 - 2 * c21 - sqrt3 * c32;
    M = L * (L - sqrt3 * K) * c12 - L * (K + sqrt3 * L) * c11 - (L - sqrt3 *
        K) * (L * c22 - K * c21);

    % Calculate x and y
    denom_xy = sqrt3 * (K^2 + L^2);
    if denom_xy == 0
        x_workspace(idx) = NaN;
        y_workspace(idx) = NaN;
    else
        x_workspace(idx) = -M / denom_xy;
        y_workspace(idx) = c22 - (K / L) * c21 - (K * M) / (sqrt3 * L * (K^2 +
            L^2));
    end
end

% Remove any NaN entries from workspace positions
valid_workspace = ~isnan(x_workspace) & ~isnan(y_workspace);
x_workspace = x_workspace(valid_workspace);
y_workspace = y_workspace(valid_workspace);
workspace_sigma_min = valid_sigma_min(valid_workspace);

% Plot the reachable workspace
figure;
scatter(x_workspace, y_workspace, 10, workspace_sigma_min, 'filled');
colorbar;
title('Workspace Visualization (x vs y)');
xlabel('x (mm)');
ylabel('y (mm)');
grid on;
axis equal;
colormap jet;
caxis([min_sigma_min, max_sigma_max]);
colorTitleHandle = get(colorbar, 'Title');
colorTitleString = 'Minimum \sigma';
set(colorTitleHandle, 'String', colorTitleString);


colorTitleString = 'Minimum \sigma';
set(colorTitleHandle, 'String', colorTitleString);
```

Appendix E: Autolev Code for Dynamic Derivations

This appendix contains the Autolev code used for deriving the dynamic equations of motion for the 3RRP mechanism. Separate implementations are provided for Kane's and Lagrange's methods.

E.1 Kane's Method

The following Autolev code derives the dynamic equations of motion for the 3RRP mechanism using Kane's method. This approach is efficient for systems with a large number of constraints and simplifies the computation by focusing on non-inertial forces.

Listing 3: Autolev Code for Kinematics and Dynamics

```
1
2  %     File:  RRP_Kane.al    [ http://www.autolev.com ]
3  %     Date:  31/12/2024
4  %   Author:  Yunus Emre Danabas / Sezer Kocaekiz / Gizem Doga Filiz
5  %  Question:  3
6  %----------------------------------------------------------------------
7  %       Default settings
8  AutoEpsilon 1.0E-14     % Rounds off to nearest integer
9  AutoZ       OFF         % Turn ON for large problems
10 Digits      7           % Number of digits displayed for numbers
11 DEGREES ON
12 UnitSystem kg, meter, sec
13 %----------------------------------------------------------------------
14 %       Newtonian, bodies, frames, particles, points
15 Newtonian   N
16 Bodies      S,T,V,E
17 Frame       A,B
18 Points      O,Z,P,Q,R
19 Variables   s{3}'', q{3}'', X'',Y'', theta''
20 MotionVariables'   u{9}'       % Configuration variables
21 %----------------------------------------------------------------------
22 %       Variables, constants, and specified
23 Specified  FE{3}, TS1,  TT1, TV1, TZ        % Contact forces
24 Constants  L
25 Constants  g=9.81
26 % ZEE_NOT= [FG1,FG2,FG3]
27 %----------------------------------------------------------------------
28 %       Mass and inertia
29 Mass S=mS, T=mT, V=mV, E=mE
30 Inertia S, IS11, IS22, IS33
31 Inertia T, IT11, IT22, IT33
32 Inertia V, IV11, IV22, IV33
33 Inertia E, IE11, IE22, IE33
34 %----------------------------------------------------------------------
35 %       Geometry relating unit vectors
36 SIMPROT(N,T,3,q1)
37 SIMPROT(N,S,3,q2)
38 SIMPROT(N,V,3,q3)
39 SIMPROT(N,E,3,theta)
40
41 SIMPROT(E,A,3,60)
42 SIMPROT(E,B,3,-60)
43 %----------------------------------------------------------------------
44 % Kinematical differential equations
45 q1' = u1
46 q2' = u2
47 q3' = u3
48
49 s1' = u4
50 s2' = u5
51 s3' = u6
```

```
52
53  x' = u7
54  y' = u8
55  theta' = u9
56
57  %----------------------------------------------------------------------
58  %       Position vectors
59  P_No_O> = 0>
60  P_O_Q> =  l*S1>
61  P_O_P> =  l*T1>
62  P_O_R> =  l*V1>
63
64  P_Z_P> =  s1*A1>
65  P_Q_Z> =  s2*E1>
66  P_Z_R> =  s3*B1>
67
68  P_O_So> = (0.33*L)*S1>
69  P_O_To> = (0.33*L)*T1>
70  P_O_Vo> = (0.33*L)*V1>
71
72  P_Z_Eo> = 0>
73  P_Z_Ao> = 0>
74  P_Z_Bo> = 0>
75
76  P_O_Z> = x*N1> + y*N2>
77
78  %----------------------------------------------------------------------
79  % Configuration Constraints
80  LOOP1> = P_Z_Q> + P_Q_O> + P_O_Z>
81  LOOP2> = P_Z_R> + P_R_O> +P_O_Z>
82  LOOP3> = P_Z_P> + P_P_O> + P_O_Z>
83
84  ZeroConfig[1] = DOT(LOOP1>,N1>)
85  ZeroConfig[2] = DOT(LOOP1>,N2>)
86  ZeroConfig[3] = DOT(LOOP2>,N1>)
87  ZeroConfig[4] = DOT(LOOP2>,N2>)
88  ZeroConfig[5] = DOT(LOOP3>,N1>)
89  ZeroConfig[6] = DOT(LOOP3>,N2>)
90  %----------------------------------------------------------------------
91  %       Angular velocities
92  w_T_N> = q1'*T3>
93  w_S_N> = q2'*S3>
94  w_V_N> = q3'*V3>
95
96  w_E_N> = theta'*E3>
97
98  w_B_E> = 0>
99  w_A_E> = 0>
100 %----------------------------------------------------------------------
101 %       Velocities
102
103 % Velocities
104 V_O_N> = 0>
105
106 V_Q_N> = dt(P_No_Q>,N)
107 V_P_N> = dt(P_No_P>,N)
108 V_R_N> = dt(P_No_R>,N)
109
110 V_Z_N> = dt(P_O_Z>,N) % x' y' buradan gelmeli
111
112 V_So_N> = dt(P_No_So>,N)
113 V_To_N> = dt(P_No_To>,N)
114 V_Vo_N> = dt(P_No_Vo>,N)
```

```
115
116 V_Eo_N> = V_Z_N>
117 %----------------------------------------------------------------------
118 % Motion constraints
119 dLOOP1> = dt(LOOP1>,N)
120 dLOOP2> = dt(LOOP2>,N)
121 dLOOP3> = dt(LOOP3>,N)
122
123 Dependent[1] = dot(dLOOP1>,N1>)
124 Dependent[2] = dot(dLOOP1>,N2>)
125
126 Dependent[3] = dot(dLOOP2>,N1>)
127 Dependent[4] = dot(dLOOP2>,N2>)
128
129 Dependent[5] = dot(dLOOP3>,N1>)
130 Dependent[6] = dot(dLOOP3>,N2>)
131
132 Constrain(Dependent[u4,u5,u6, u7,u8,u9])
133 %----------------------------------------------------------------------
134 %       Angular accelations
135 ALF_T_N> = dt(w_T_N>,N)
136 ALF_S_N> = dt(w_S_N>,N)
137 ALF_V_N> = dt(w_V_N>,N)
138 ALF_E_N> = dt(w_E_N>,N)
139 %----------------------------------------------------------------------
140 %       Accelerations of particles and mass centers of bodies
141 % A_No_N> = 0>
142 A_To_N> = dt(V_To_N>,N)
143 A_So_N> = dt(V_So_N>,N)
144 A_Vo_N> = dt(V_Vo_N>,N)
145
146 A_Z_N> = dt(V_Z_N>,N)
147 A_Eo_N> = A_Z_N>
148
149
150 A_P_N> = dt(V_P_N>,N)
151 A_Q_N> = dt(V_Q_N>,N)
152 A_R_N> = dt(V_R_N>,N)
153
154 %----------------------------------------------------------------------
155 %       Forces
156 Gravity( -g*N3> )
157 Force_Z>  = FE1*N1> + FE2*N2>
158 Torque_E> = TZ*N3>
159 %----------------------------------------------------------------------
160 % Torques
161 Torque_S> = TS1*S3>
162 Torque_T> = TT1*T3>
163 Torque_V> = TV1*V3>
164
165 %----------------------------------------------------------------------
166
167 JACOBIAN = [D(u7,u1), D(u7,u2),D(u7,u3); D(u8,u1), D(u8,u2),D(u8,u3);
        D(u9,u1), D(u9,u2),D(u9,u3)]
168
169 %----------------------------------------------------------------------
170 %       Equations of motion
171 Zero = Fr() + FrStar()
172 Kane( )
173
174 %----------------------------------------------------------------------
175
176
```

```
Input tFinal=10, integStp=0.1, absErr=1.0E-07, relErr=1.0E-07
Input L = 200 mm
Input IS33= 0.00012, IT33= 0.00012, IV33= 0.00012, IE33= 0.000050

Input q1=0 deg, q2=120 deg, q3= 240 deg
Input x= 0 mm , y= 0 mm, theta = 0 deg
Input TA = 0, TD = 0, uA = 0, uD = 0, Fx = 0, Fy = 0

%---------------------------------------------------------------------
% Quantities to be output from CODE
Output t sec, x mm, y mm, theta deg, u7 mm/s, u8 mm/s, u9 rad/s , q1 rad, q2
    rad, q3 rad, u1 rad/s, u2 rad/s, u3 rad/s


code dynamics() dinamik_yunus.m


%Record Autolev responses
Save RRP_Kane_Results_Yunus.all
```

E.2 Lagrange's Method

The following Autolev code derives the dynamic equations of motion for the 3RRP mechanism using Lagrange's method. This approach is based on energy principles and is well-suited for systems with relatively simple constraint equations.

Listing 4: Autolev Code for Dynamic Derivation Using Lagrange's Method

```

%     File:  RRP_Lag.al    [ http://www.autolev.com ]
%     Date:  31/12/2024
%   Author:  Yunus Emre Danabas / Sezer Kocaekiz / Gizem Doga Filiz
%  Question:  3
%---------------------------------------------------------------------
%       Default settings
AutoEpsilon 1.0E-14     % Rounds off to nearest integer
AutoZ       ON        % Turn ON for large problems
Digits      7           % Number of digits displayed for numbers
DEGREES ON
%---------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian   N
Bodies      S,T,V,E
Frame       A,B
Points      O,Z,P,Q,R
Variables   s{3}'', q{3}'', X'',Y'', theta''
Variables lambda{6} % Lagrangian multipliers
Constants alpha, beta % Baumgarte stabilization gains
%---------------------------------------------------------------------
%       Variables, constants, and specified
Specified  FE{3}, TS1,  TT1, TV1        % Contact forces
Constants  L
Constants  g=9.81
% ZEE_NOT= [FG1,FG2,FG3]
%---------------------------------------------------------------------
%       Mass and inertia
Mass S=mS, T=mT, V=mV, E=mE
Inertia S, IS11, IS22, IS33
Inertia T, IT11, IT22, IT33
Inertia V, IV11, IV22, IV33
Inertia E, IE11, IE22, IE33
%---------------------------------------------------------------------
%       Geometry relating unit vectors
```

```
SIMPROT(N,T,3,q1)
SIMPROT(N,S,3,q2)
SIMPROT(N,V,3,q3)
SIMPROT(N,E,3,theta)

SIMPROT(E,A,3,60)
SIMPROT(E,B,3,-60)
%----------------------------------------------------------------------
%       Position vectors
P_No_O> = 0>
P_O_Q> =  l*S1>
P_O_P> =  l*T1>
P_O_R> =  l*V1>

P_Z_P> =  s1*A1>
P_Q_Z> =  s2*E1>
P_Z_R> =  s3*B1>

P_O_So> = (0.33*L)*S1>
P_O_To> = (0.33*L)*T1>
P_O_Vo> = (0.33*L)*V1>

P_Z_Eo> = 0>
P_Z_Ao> = 0>
P_Z_Bo> = 0>

P_O_Z> = x*N1> + y*N2>

%----------------------------------------------------------------------
% Configuration Constraints
LOOP1> = P_Z_Q> + P_Q_O> + P_O_Z>
LOOP2> = P_Z_R> + P_R_O> +P_O_Z>
LOOP3> = P_Z_P> + P_P_O> + P_O_Z>

ZeroConfig[1] = DOT(LOOP1>,N1>)
ZeroConfig[2] = DOT(LOOP1>,N2>)
ZeroConfig[3] = DOT(LOOP2>,N1>)
ZeroConfig[4] = DOT(LOOP2>,N2>)
ZeroConfig[5] = DOT(LOOP3>,N1>)
ZeroConfig[6] = DOT(LOOP3>,N2>)
%----------------------------------------------------------------------
%       Angular velocities
w_T_N> = q1'*T3>
w_S_N> = q2'*S3>
w_V_N> = q3'*V3>

w_E_N> = theta'*E3>

w_B_E> = 0>
w_A_E> = 0>
%----------------------------------------------------------------------
%       Velocities

% Velocities
V_O_N> = 0>

V_Q_N> = dt(P_No_Q>,N)
V_P_N> = dt(P_No_P>,N)
V_R_N> = dt(P_No_R>,N)

V_Z_N> = dt(P_O_Z>,N) % x' y' buradan gelmeli

V_So_N> = dt(P_No_So>,N)
```

```
V_To_N> = dt(P_No_To>,N)
V_Vo_N> = dt(P_No_Vo>,N)

V_Eo_N> = V_Z_N>

%----------------------------------------------------------------------
% Motion constraints
dLOOP1> = dt(LOOP1>,N)
dLOOP2> = dt(LOOP2>,N)
dLOOP3> = dt(LOOP3>,N)

Dependent[1] = dot(dLOOP1>,N1>)
Dependent[2] = dot(dLOOP1>,N2>)

Dependent[3] = dot(dLOOP2>,N1>)
Dependent[4] = dot(dLOOP2>,N2>)

Dependent[5] = dot(dLOOP3>,N1>)
Dependent[6] = dot(dLOOP3>,N2>)

%----------------------------------------------------------------------
%       Angular accelations
ALF_T_N> = dt(w_T_N>,N)
ALF_S_N> = dt(w_S_N>,N)
ALF_V_N> = dt(w_V_N>,N)
ALF_E_N> = dt(w_E_N>,N)
%----------------------------------------------------------------------
%       Accelerations of particles and mass centers of bodies
% A_No_N> = 0>
A_To_N> = dt(V_To_N>,N)
A_So_N> = dt(V_So_N>,N)
A_Vo_N> = dt(V_Vo_N>,N)
A_Z_N> = dt(V_Z_N>,N)
A_Eo_N> = A_Z_N>

A_P_N> = dt(V_P_N>,N)
A_Q_N> = dt(V_Q_N>,N)
A_R_N> = dt(V_R_N>,N)

%----------------------------------------------------------------------
%       Forces
Gravity( -g*N3> )
Force_Z>  = FE1*N1> + FE2*N2> + FE3*N3>
%----------------------------------------------------------------------
% Torques
Torque_S> = TS1*S3>
Torque_T> = TT1*T3>
Torque_V> = TV1*V3>
%----------------------------------------------------------------------

%----------------------------------------------------------------------
% Langrangian and Generalized Force
KE = KE() % Kinetic energy
PE = 0
Lag = KE - PE
explicit(Lag)

ddLag =
    [dt(d(Lag,q1'));dt(d(Lag,q2'));dt(d(Lag,q3'));dt(d(Lag,s1'));dt(d(Lag,s2'));dt(d(Lag,s3')

dLag =
    [d(Lag,q1);d(Lag,q2);d(Lag,q3);d(Lag,s1);d(Lag,s2);d(Lag,s3);d(Lag,X);d(Lag,Y);d(Lag,thet

```

```
160 Work = dot(W_S_N>,Torque_S>) + dot(W_T_N>,Torque_T>) + dot(W_V_N>,Torque_V>) +
        dot(V_Z_N>,Force_Z>)
161
162 Q =
        [coef(Work,q1');coef(Work,q2');coef(Work,q3');coef(Work,s1');coef(Work,s2');coef(Work,s3'
163
164 %----------------------------------------------------------------------
165 % EoM for the Unconstaint System
166 Zero_EoM = ddLag - dLag - Q
167 %----------------------------------------------------------------------
168 % Supplementary code to check the EoM
169 LHS = ddLag - dLag
170 RHS = Q
171
172 %----------------------------------------------------------------------
173 % DAEs for the Constrained System
174 Lambda[1] = lambda1
175 Lambda[2] = lambda2
176 Lambda[3] = lambda3
177 Lambda[4] = lambda4
178 Lambda[5] = lambda5
179 Lambda[6] = lambda6
180
181 dZeroConfig[1,1] = d(ZeroConfig[1],q1)
182 dZeroConfig[1,2] = d(ZeroConfig[1],q2)
183 dZeroConfig[1,3] = d(ZeroConfig[1],q3)
184 dZeroConfig[1,4] = d(ZeroConfig[1],s1)
185 dZeroConfig[1,5] = d(ZeroConfig[1],s2)
186 dZeroConfig[1,6] = d(ZeroConfig[1],s3)
187 dZeroConfig[1,7] = d(ZeroConfig[1],X)
188 dZeroConfig[1,8] = d(ZeroConfig[1],Y)
189 dZeroConfig[1,9] = d(ZeroConfig[1],theta)
190
191 dZeroConfig[2,1] = d(ZeroConfig[2],q1)
192 dZeroConfig[2,2] = d(ZeroConfig[2],q2)
193 dZeroConfig[2,3] = d(ZeroConfig[2],q3)
194 dZeroConfig[2,4] = d(ZeroConfig[2],s1)
195 dZeroConfig[2,5] = d(ZeroConfig[2],s2)
196 dZeroConfig[2,6] = d(ZeroConfig[2],s3)
197 dZeroConfig[2,7] = d(ZeroConfig[2],X)
198 dZeroConfig[2,8] = d(ZeroConfig[2],Y)
199 dZeroConfig[2,9] = d(ZeroConfig[2],theta)
200
201
202 %--------
203 dZeroConfig[3,1] = d(ZeroConfig[3],q1)
204 dZeroConfig[3,2] = d(ZeroConfig[3],q2)
205 dZeroConfig[3,3] = d(ZeroConfig[3],q3)
206 dZeroConfig[3,4] = d(ZeroConfig[3],s1)
207 dZeroConfig[3,5] = d(ZeroConfig[3],s2)
208 dZeroConfig[3,6] = d(ZeroConfig[3],s3)
209 dZeroConfig[3,7] = d(ZeroConfig[3],X)
210 dZeroConfig[3,8] = d(ZeroConfig[3],Y)
211 dZeroConfig[3,9] = d(ZeroConfig[3],theta)
212
213 dZeroConfig[4,1] = d(ZeroConfig[4],q1)
214 dZeroConfig[4,2] = d(ZeroConfig[4],q2)
215 dZeroConfig[4,3] = d(ZeroConfig[4],q3)
216 dZeroConfig[4,4] = d(ZeroConfig[4],s1)
217 dZeroConfig[4,5] = d(ZeroConfig[4],s2)
218 dZeroConfig[4,6] = d(ZeroConfig[4],s3)
219 dZeroConfig[4,7] = d(ZeroConfig[4],X)
220 dZeroConfig[4,8] = d(ZeroConfig[4],Y)
```

```
dZeroConfig[4,9] = d(ZeroConfig[4],theta)


%--------

dZeroConfig[5,1] = d(ZeroConfig[5],q1)
dZeroConfig[5,2] = d(ZeroConfig[5],q2)
dZeroConfig[5,3] = d(ZeroConfig[5],q3)
dZeroConfig[5,4] = d(ZeroConfig[5],s1)
dZeroConfig[5,5] = d(ZeroConfig[5],s2)
dZeroConfig[5,6] = d(ZeroConfig[5],s3)
dZeroConfig[5,7] = d(ZeroConfig[5],X)
dZeroConfig[5,8] = d(ZeroConfig[5],Y)
dZeroConfig[5,9] = d(ZeroConfig[5],theta)

dZeroConfig[6,1] = d(ZeroConfig[6],q1)
dZeroConfig[6,2] = d(ZeroConfig[6],q2)
dZeroConfig[6,3] = d(ZeroConfig[6],q3)
dZeroConfig[6,4] = d(ZeroConfig[6],s1)
dZeroConfig[6,5] = d(ZeroConfig[6],s2)
dZeroConfig[6,6] = d(ZeroConfig[6],s3)
dZeroConfig[6,7] = d(ZeroConfig[6],X)
dZeroConfig[6,8] = d(ZeroConfig[6],Y)
dZeroConfig[6,9] = d(ZeroConfig[6],theta)

Zero_Constrained_EoM = Zero_EoM + transpose(dZeroConfig)*Lambda
%                          9x1            6x9                 6x1
%                          9x1                      9x6                 6x1
%-----------------------------------------------------------------------
% Units system for  CODE input/output conversions
UnitSystem kg,meter,sec
%-----------------------------------------------------------------------
% Quantities to be output from  CODE
Output t sec, q1 rad, q2 rad, q3 rad, s1 m, s2 m, s3 m, X m, Y m, theta rad
Output q1'' rad/s^2, q2'' rad/s^2, q3'' rad/s^2
Output lambda1 N*m, lambda2 N*m, lambda3 N*m, lambda4 N*m, lambda5 N*m,
    lambda6 N*m
%-----------------------------------------------------------------------
% Baumgarte Stabilization
q_vec = [q1; q2; q3; s1; s2; s3; x; y; theta]

temp = dZeroConfig*dt(q_vec)
%6x1        6x9        9x1

dtemp[1,1] = d(temp[1],q1)
dtemp[1,2] = d(temp[1],q2)
dtemp[1,3] = d(temp[1],q3)
dtemp[1,4] = d(temp[1],s1)
dtemp[1,5] = d(temp[1],s2)
dtemp[1,6] = d(temp[1],s3)
dtemp[1,7] = d(temp[1],X)
dtemp[1,8] = d(temp[1],Y)
dtemp[1,9] = d(temp[1],theta)

dtemp[2,1] = d(temp[2],q1)
dtemp[2,2] = d(temp[2],q2)
dtemp[2,3] = d(temp[2],q3)
dtemp[2,4] = d(temp[2],s1)
dtemp[2,5] = d(temp[2],s2)
dtemp[2,6] = d(temp[2],s3)
dtemp[2,7] = d(temp[2],X)
dtemp[2,8] = d(temp[2],Y)
dtemp[2,9] = d(temp[2],theta)
```

```
283
284 dtemp[3,1] = d(temp[3],q1)
285 dtemp[3,2] = d(temp[3],q2)
286 dtemp[3,3] = d(temp[3],q3)
287 dtemp[3,4] = d(temp[3],s1)
288 dtemp[3,5] = d(temp[3],s2)
289 dtemp[3,6] = d(temp[3],s3)
290 dtemp[3,7] = d(temp[3],X)
291 dtemp[3,8] = d(temp[3],Y)
292 dtemp[3,9] = d(temp[3],theta)
293
294 dtemp[4,1] = d(temp[4],q1)
295 dtemp[4,2] = d(temp[4],q2)
296 dtemp[4,3] = d(temp[4],q3)
297 dtemp[4,4] = d(temp[4],s1)
298 dtemp[4,5] = d(temp[4],s2)
299 dtemp[4,6] = d(temp[4],s3)
300 dtemp[4,7] = d(temp[4],X)
301 dtemp[4,8] = d(temp[4],Y)
302 dtemp[4,9] = d(temp[4],theta)
303
304 dtemp[5,1] = d(temp[5],q1)
305 dtemp[5,2] = d(temp[5],q2)
306 dtemp[5,3] = d(temp[5],q3)
307 dtemp[5,4] = d(temp[5],s1)
308 dtemp[5,5] = d(temp[5],s2)
309 dtemp[5,6] = d(temp[5],s3)
310 dtemp[5,7] = d(temp[5],X)
311 dtemp[5,8] = d(temp[5],Y)
312 dtemp[5,9] = d(temp[5],theta)
313
314 dtemp[6,1] = d(temp[6],q1)
315 dtemp[6,2] = d(temp[6],q2)
316 dtemp[6,3] = d(temp[6],q3)
317 dtemp[6,4] = d(temp[6],s1)
318 dtemp[6,5] = d(temp[6],s2)
319 dtemp[6,6] = d(temp[6],s3)
320 dtemp[6,7] = d(temp[6],X)
321 dtemp[6,8] = d(temp[6],Y)
322 dtemp[6,9] = d(temp[6],theta)
323
324
325
326
327 first_term = -dtemp*dt(q_vec)
328 % 6x1              6x9        9x1
329 second_term = -2*dt(dZeroConfig)*dt(q_vec)
330 %    6x1              6x9               9x1
331 third_term = -dt(dt(ZeroConfig))
332 fourth_term = -alpha*(dZeroConfig*dt(q_vec)-third_term)
333 fifth_term = -beta*ZeroConfig
334
335 gamma = first_term + second_term + third_term + fourth_term + fifth_term
336 % 6x1
337 extra_term = transpose(dZeroConfig)*Lambda
338 %   9x1                   9x6          6x1
339
340 algebraic_eqn_ft = dZeroConfig*dt(dt(q_vec))
341 %                         6x9          9x1
342
343 eqn[1] = LHS[1] - RHS[1] + extra_term[1]
344 eqn[2] = LHS[2] - RHS[2] + extra_term[2]
345 eqn[3] = LHS[3] - RHS[3] + extra_term[3]
```

```
eqn[4] = LHS[4] - RHS[4] + extra_term[4]
eqn[5] = LHS[5] - RHS[5] + extra_term[5]
eqn[6] = LHS[6] - RHS[6] + extra_term[6]
eqn[7] = LHS[7] - RHS[7] + extra_term[7]
eqn[8] = LHS[8] - RHS[8] + extra_term[8]
eqn[9] = LHS[9] - RHS[9] + extra_term[9]

eqn[10] = algebraic_eqn_ft[1] - gamma[1]
eqn[11] = algebraic_eqn_ft[2] - gamma[2]
eqn[12] = algebraic_eqn_ft[3] - gamma[3]
eqn[13] = algebraic_eqn_ft[4] - gamma[4]
eqn[14] = algebraic_eqn_ft[5] - gamma[5]
eqn[15] = algebraic_eqn_ft[6] - gamma[6]


Zee_Not = [ q1'', q2'', q3'', s1'', s2'', s3'', X'', Y'',theta'', lambda1,
    lambda2, lambda3, lambda4, lambda5, lambda6]


solve(eqn, [dt(dt(q_vec)); Lambda])
%code ode() RRP_lagrange.m
%-----------------------------------------------------------------------
% Record Autolev responses
Save RRP_Lag_Results_Yunus.all
```