**A Framework for Force-Adaptive Circular Trajectory Following for a UR10e Manipulator using ROS 2**

Internship Report

(Duration: 2025 – 30thseptember2025)

**Submitted by**

**Nithish Kumar R**

**In partial fulfilment for the requirements of the degree of Bachelor Of Technology In**

**Electrical and Electronics Engineering,**

**Vellore Institute of Technology, Chennai**

Under Supervision of

Shri.PUGALENDHI.C

**Table of Contents**

**Internship Report**

**Acknowledgment**

**Introduction**

* Satish Dhawan Space Centre (SDSC) SHAR
* Solid Motor Propellant Complex (SMPC)
* Solid Propellant Process:
  1. Oxidizer
  2. Binder
  3. Metallic fuel
  4. Plasticizer
  5. Burn rate additives
  6. Curators
  7. Other additives

**1. System Architecture and Kinematic Modelling of the UR10e Manipulator**

1.1. The UR10e Collaborative Robot Platform 1.2. Forward Kinematics Formulation

1.2.1. The Denavit-Hartenberg (DH) Convention

1.2.2. Homogeneous Transformation Matrices 1.3. Analytical Inverse Kinematics Solution

**2. Differential Kinematics and Cartesian Space Control**

2.1. The Geometric Jacobian 2.2. Cartesian Velocity Control

2.3. Singularity Analysis

**3. Trajectory Generation for Cartesian Space Tasks**

3.1. Principles of Cartesian Path Planning

3.2. Parametric Formulation of a 3D Circular Trajectory

**4. Implementation Framework using ROS 2 and Pymoveit 2**

4.1. System Architecture in ROS 2 4.2. Cartesian Path Planning with Pymoveit 2

4.3. Real-Time Force Data Acquisition

**5. Application Case Study: Force-Compliant Cleaning of a Propellant Mixer Cylinder**

5.1. Hybrid Motion-Force Control Strategy

5.2. PID-Based Force Regulation

5.3. Real-Time Trajectory Adaptation

5.4. Analysis and Recommendations \* PID Tuning \* Stability \*

**Recommendations for Future Work:**

1. Admittance Control Implementation

2. Real-Time Surface Normal Estimation

3. Learning-Based Approaches

**Conclusion**

A Framework for Force-Adaptive Circular Trajectory Following for a UR10e Manipulator using ROS 2

**ACKNOWLEDGEMENT**

I wish to express my sincere gratitude to **Dr. SHIVRAMAKRISHNAN. GM, SMPC , SHAR** for allowing me to do this internship training.

I wish to extend my gratitude to **Shri** for his support to do this internship training.

I am highly indebted to **Shri PUGALENDHI C. /Engr., Manager , CAS , SMPC , Sriharikota** for his valuable and immense support in making this internship training a success.

My sincere thanks to the team at **SATISH DHAVAN SPACE CENTRE SHAR** for warmly welcoming me and making my time here so enjoyable.

I express my sincere thanks and deep veneration to **Dr.Ageline, PROFESSOR AND HEAD, SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING, VIT CHENNAI**  for their encouragement and mentorship in my learning experience .

A Framework for Force-Adaptive Circular Trajectory Following for a UR10e Manipulator using ROS 2

**Introduction**

Satish Dhawan Space Centre (SDSC) SHAR shown in figure 1.1, located in Sriharikota, Andhra Pradesh, is premier spaceport of the Indian Space Research Organization (ISRO). Renamed in 2002 in honour of former ISRO Chairman Prof. Satish Dhawan, this facility offers world-class launch infrastructure for national and international satellite missions. Established in 1969 on a 43,360-acre island amidst the Pulicat Lake and the Bay of Bengal, SDSC SHAR benefits from its proximity to the equator, providing optimal launch azimuths and safety. The spaceport launched its first rocket, 'Rohini-125,' in 1971 and has since expanded to support remote sensing, communication, navigation, and scientific missions. This Centre has the facilities for solid propellant processing, static testing of solid motors, launch vehicle integration and launch operations, range operations comprising telemetry, tracking and command network and mission control center.

The Centre has two launch pads from where the rocket launching operations of PSLV and GSLV are carried out and third launch pad for NGLV is under construction. The mandate for the center is:

(i) To produce solid propellant boosters for the launch vehicle programs of ISRO

(ii) To provide the infrastructure for qualifying various subsystems and solid rocket motors and carrying out the necessary tests

(iii) To provide launch base infrastructure for satellites and launch vehicles. SDSC SHAR has a separate launch pad for launching sounding rockets.

The centre also provides the necessary launch base infrastructure for sounding rockets of ISRO and for assembly, integration and launch of sounding rockets and payloads.

**Solid Motor Propellant Complex (SMPC)**

SMPC is one of the important units in SHAR CENTRE. SMPC-II stands for “Solid Motor Propellant Complex”. The main objective of the SMPC-II unit is, prepare solid propellant rocket motor booster for the launch vehicles of ISRO’S Space Program's. It is the biggest solid propellant processing plant in Asia.

SMPC-II has five mixing stations. Of them, four will be in operation during final mixing. The fifth station will be contingent hot standby to meet the requirement of uninterrupted casting 100 tons of propellant within 24 hours. The casting facility is designed for continuous casting with two transfer cars to receive the bowls, tilt and transfer the propellant slurry into the master hopper which feeds the slurry to segment hardware. All operations in mixing and casting facilities are fully automated using the state-of-the-art PLC systems to avoid manual errors in operation.

**Solid propellant process:**

The propellant processed here, is composite solid propellant and it generally contains the following chemicals.

**1. Oxidizer:**

Oxidizer forms the bulk portion of the propellant formulation. It supplies the necessary oxygen for the fuel to burn.

**2. Binder:**

The binder generally used is a polymer. It acts as fuel and as binding material to hold together the solid particles.

**3. Metallic fuel:**

The metallic fuel besides its role in the combustion reaction, also increases the temp. and thereby increase specific impulse.

**4. Plasticizer:**

Plasticizer improves the processability by imparting good flow and mechanical properties to the propellant.

**5. Burn rate additives:**

These additives act as catalysts to get the required burning rate of propellant.

# System Architecture and Kinematic Modelling of the UR10e Manipulator

This section establishes the foundational elements of the project: the physical hardware and its mathematical representation. The capabilities of the Universal Robots UR10e are detailed, followed by a rigorous derivation of the kinematic models that are prerequisites for any subsequent motion planning or control.

## The UR10e Collaborative Robot Platform

The Universal Robots UR10e is a six-axis serial manipulator classified as a collaborative robot (cobot), engineered for versatility and seamless integration into a wide range of industrial applications.[1, 2, 3] Its design prioritizes safety, ease of use, and redeploy ability, making it a suitable platform for complex tasks that require both precision and adaptability.

The robot’s physical specifications are critical in defining the scope and limitations of the application. The UR10e features a substantial reach of 1300 mm and a payload capacity of 12.5 kg.[4, 5] These characteristics are more than sufficient for the target application of cleaning the interior of a large propellant mixer, which involves manipulating a cleaning tool of moderate weight over a significant area. The manipulator’s kinematic structure consists of six revolute joints (6-DOF), a configuration that provides full spatial mobility, enabling the end-effector to achieve any position and orientation within its workspace.[4, 5] This capability is essential for tasks like surface cleaning, which demand precise tool orientation, typically normal to the working surface, to ensure uniform application of force.

A pivotal component for this project is the force/torque sensor integrated directly into the robot’s tool flange.[4, 5] This sensor provides the real-time feedback necessary for implementing a compliant control loop. It offers a force measurement range of ±100*.*0 N with a precision of ±5*.*0 N, and a torque measurement range of ±10*.*0 Nm with a precision of ±0*.*2 Nm.[4, 5] The performance of this integrated sensor is a key enabler for the proposed force-adaptive cleaning task. The specified precision directly dictates the theoretical limit of the control system’s ability to regulate the cleaning force. For a task requiring a contact force of, for example, 20 N, a sensor precision of ±5*.*0 N is adequate. However, if the task demanded a much finer force control, such as ±1*.*0 N, this integrated sensor would be insufficient, necessitating the integration of a higher-precision external sensor. Thus, the selection of the UR10e platform is fundamentally justified by the alignment of its built-in sensing capabilities with the control requirements of the application.

The overall system is managed by a dedicated control box that provides extensive I/O capabilities and supports a variety of industrial communication protocols, including native support for the Robot Operating System (ROS/ROS2). This native support is fundamental to the software architecture, as it provides a direct and well-documented interface for motion planning and real-time control.[4]

## Forward Kinematics Formulation

Forward kinematics (FK) is the process of computing the position and orientation (pose) of the robot’s end-effector given the values of its joint angles.[6, 7, 8] This forms the mathematical basis for understanding the robot’s posture and is a prerequisite for any form of motion planning. The standard method for this analysis is the Denavit-Hartenberg (DH) convention.

Table 1: UR10e Technical Specifications

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Specification** | **Source(s)** |
| **Payload** | 12.5 kg (27.5 lbs) | [4, 5] |
| **Reach** | 1300 mm (51.2 in) | [4, 5] |
| **Degrees of Freedom (DOF)** | 6 revolute joints | [4] |
| **Pose Repeatability** | ±0*.*05 mm (per ISO 9283) | [5] |
| **Joint Ranges** | ±360◦ for all joints | [5] |
| **Maximum Joint Speeds** | Base/Shoulder: ±120◦/s; Elbow/Wrists: ±180◦/s | [5] |
| **Typical TCP Speed** | 1 m/s | [5] |
| **Force Sensing Range (Tool)** | ±100*.*0 N | [5] |
| **Force Sensing Precision (Tool)** | ±5*.*0 N | [4] |
| **Torque Sensing Range (Tool)** | ±10*.*0 Nm | [5] |
| **Torque Sensing Precision (Tool)** | ±0*.*2 Nm | [5] |
| **Communication Protocols** | Modbus-TCP, PROFINET, Ethernet/IP, ROS/ROS2 | [4] |

### The Denavit-Hartenberg (DH) Convention

The DH convention provides a systematic methodology for attaching coordinate frames to each link of a serial manipulator.[9, 10, 11] This approach simplifies the complex spatial geometry of the robot into a standardized set of four parameters for each joint-link pair: link length (*ai*), link twist (*αi*), link offset (*di*), and joint angle (*θi*).[10, 12] These parameters define the geometric relationship and transformation between the coordinate frame of link *i* − 1 and the coordinate frame of link *i*.

The DH parameters for the idealized UR10e model are presented in Table 2. These parameters are the cornerstone of the robot’s kinematic model and are used to derive the transformation matrices that define its motion.[13]

Table 2: Denavit-Hartenberg Parameters for UR10e

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Joint (***i***)** | **Link Twist (***αi*−1**)** | **Link Length (***ai*−1**)** | **Link Offset (***di***)** | **Joint Angle (***θi***)** |
| 1 | 0 | 0 | *d*1 = 0*.*1807 m | *θ1* |
| 2 | *π/*2 | 0 | *d*2 = 0.0 | *θ2* |
| 3 | 0 | *a*3 = 0.6127 m | *d*3 = 0.0 | *θ4* |
| 4 | 0 | *a*4 = −0*.*57155 m | *d*4 = 0*.*17415 m | *θ5* |
| 5 | *π/*2 | 0 | *d*5 = 0*.*11985 m |  |
| 6 | *π/*2 | 0 | *d*6= 0.11655 m | *θ6* |

### Homogeneous Transformation Matrices

Using the DH parameters, a homogeneous transformation matrix, *i*−1*Ti*, can be formulated for each joint. This 4 × 4 matrix represents the complete transformation (both rotation and translation) from the coordinate frame of link *i* − 1 to that of link *i*.[12, 14] It is constructed as the product of four basic transformations corresponding to the four DH parameters: This results in the general form of the transformation matrix: The final pose of the end-effector (frame 6) with respect to the base (frame 0), denoted as 0*T*6, is obtained by serially multiplying these individual transformation matrices [6, 15]: The resulting matrix contains complex trigonometric functions of the six joint angles and provides the end-effector’s orientation (the upper-left 3 × 3 submatrix) and position (the first three elements of the last column) in the base frame.

## Analytical Inverse Kinematics Solution

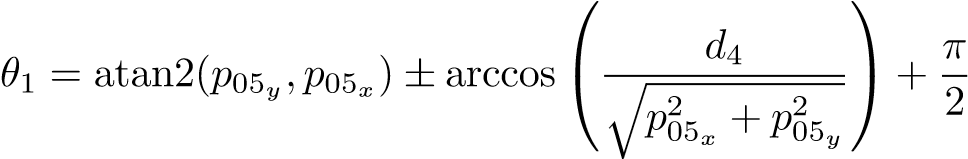
Inverse kinematics (IK) addresses the more complex problem of determining the set of joint angles, *q* = [*θ*1*,...,θ*6]*T*, that will place the end-effector at a desired pose, .[16, 17, 18] Unlike the unique solution of forward kinematics, an IK problem for a 6-DOF manipulator can have multiple solutions, a single solution, or no solution at all, depending on the desired pose and the robot’s geometry.[19] For real-time control and planning, an efficient method for solving the IK problem is essential.

The geometric structure of the UR10e arm, which features a spherical wrist (the axes of joints 4, 5, and 6 intersect at a common point), permits a significant simplification of the IK problem through a technique known as kinematic decoupling.[20] This method separates the complex 6-DOF problem into two more manageable 3-DOF sub-problems:

1. **Inverse Position Problem:** Finding the angles of the first three joints (*θ*1*,θ*2*,θ*3) to place the center of the wrist at the correct location in space.
2. **Inverse Orientation Problem:** Finding the angles of the last three joints (*θ*4*,θ*5*,θ*6) to achieve the desired orientation of the end-effector.

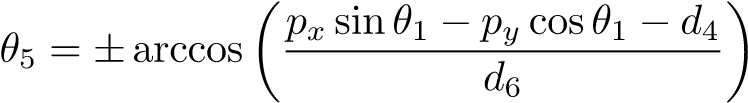
A detailed, closed-form geometric solution for the UR-series robots exists and provides all possible joint configurations analytically.[21] This approach is computationally very fast compared to numerical iterative methods. The derivation proceeds as follows:

1. **Solving for** *θ*1: The first joint angle is found by considering the projection of the wrist center onto the *xy*-plane of the base frame. By equating the forward kinematic equations with the desired pose matrix, a phase-shift equation is derived for *θ*1. This typically yields two solutions, corresponding to ”shoulder-left” and ”shoulder-right” configurations.[21]

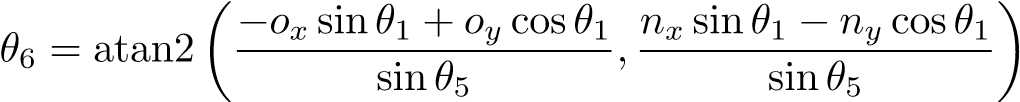


where (*p*05*x,p*05*y*) are the coordinates of the wrist center projected onto the *xy*-plane, derived from the target pose.

1. **Solving for** *θ*5: With *θ*1 known, the problem can be viewed from the perspective of frame 1. The position of the end-effector in this frame provides a constraint that leads to a solution for *θ*5. This also yields two solutions, corresponding to ”wrist-up” and ”wrist-down” postures.[21]



1. **Solving for** *θ*6: The orientation of the end-effector relative to frame 5 is used to solve for *θ*6. By examining the elements of the rotation matrix that relates frame 6 to frame 1, a unique value for *θ*6 can be found for each solution of *θ*1 and *θ*5.[21]



1. **Solving for** *θ*2*,θ*3*,θ*4: With the first, fifth, and sixth joint angles determined, the remaining joints (2, 3, and 4) form a planar 3R arm. The positions of the base of this arm (joint 2) and its end (the wrist center) are now known, allowing for a standard geometric solution for the remaining angles. This sub-problem yields two solutions for the ”elbow-up” and ”elbow-down” configurations.[21]

In total, this analytical method can produce up to eight distinct IK solutions for a single reachable end-effector pose. The ability to compute all solutions is advantageous for advanced motion planning, as it allows the planner to select the configuration that is closest to the current one, avoids joint limits, or is most optimal according to some other criterion.[22]

It is crucial to recognize the potential discrepancy between the idealized DH model used for this analytical derivation and the actual kinematics of a physical robot. Universal Robots performs a factory calibration for each arm to compensate for minor manufacturing tolerances, resulting in a unique set of kinematic parameters that may include non-ideal offsets.[23] These calibration parameters can invalidate the strict geometric assumptions (e.g., perfect intersection of wrist axes) required by the simplified analytical solution. Consequently, while the analytical solution is invaluable for theoretical understanding and high-speed computation in ideal cases, a production-grade implementation often relies on numerical IK solvers provided by frameworks like MoveIt. These numerical methods are more robust as they can directly use the robot’s factory-calibrated model, ensuring higher positional accuracy at the cost of being iterative and not guaranteeing all possible solutions.[24] For this report, the analytical solution is derived for its theoretical rigor, with the understanding that a practical implementation would leverage the more robust numerical solvers within the ROS framework.

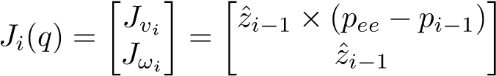
# Differential Kinematics and Cartesian Space Control

Differential kinematics bridges the gap between the static kinematic model and the dynamic control of the manipulator. It describes the relationship between the velocities of the robot’s joints and the resulting linear and angular velocity of its end-effector. The central mathematical tool for this analysis is the Jacobian matrix.

## The Geometric Jacobian

The Jacobian is a fundamental concept in robotics that provides a linear mapping between the velocities in the robot’s joint space and the velocities in its Cartesian workspace.[25, 26, 27] For a manipulator with *n* joints, the Jacobian *J*(*q*) is a 6 × *n* matrix that relates the vector of joint velocities, ˙*q* ∈ *Rn*, to the end-effector’s spatial velocity (or twist), *νee* ∈ *R*6. The twist vector combines the linear velocity (*p*˙*ee*) and angular velocity (*ωee*) of the end-effector: *νee* =*T*.[28] The Jacobian is not a constant matrix; its values are a function of the robot’s current joint configuration, *q*.[29]

The derivation of the geometric Jacobian for a serial manipulator is systematic. The *i*-th column of the Jacobian, *Ji*, represents the contribution of the *i*-th joint’s velocity to the end-effector’s twist. The form of this column depends on whether the joint is revolute (rotating) or prismatic (sliding). Since the UR10e consists entirely of revolute joints, the *i*-th column is given by [25, 30, 31]:



Where:

* *Jvi* is the 3 × 1 vector representing the contribution to the linear velocity of the end-effector.
* *Jωi* is the 3 × 1 vector representing the contribution to the angular velocity of the end-effector.
* *z*ˆ*i*−1 is the unit vector representing the axis of rotation for joint *i*, expressed in the base frame coordinates.
* *pee* is the position vector of the end-effector’s origin, expressed in the base frame.
* *pi*−1 is the position vector of the origin of frame *i* − 1 (the base of joint *i*), expressed in the base frame.
* × denotes the vector cross product.

The term ˆ*zi*−1 × (*pee* − *pi*−1) describes the instantaneous linear velocity of the end-effector resulting from a unit angular velocity of joint *i*. The term ˆ*zi*−1 simply states that the angular velocity contribution from joint *i* is a rotation about its own axis. The full 6 × 6 Jacobian for the UR10e is constructed by concatenating these column vectors for each of the six joints:

*J*(*q*) = [*J*1(*q*) *J*2(*q*) *J*3(*q*) *J*4(*q*) *J*5(*q*) *J*6(*q*)]

The vectors ˆ*zi*−1, *pee*, and *pi*−1 are all obtained from the forward kinematics transformations derived in the previous section. While other forms of the Jacobian exist (e.g., analytical, body, spatial), the geometric Jacobian is the most direct representation of the physical velocities of the end-effector in the base frame, making it ideal for Cartesian control applications.[25, 32] This matrix essentially acts as a time-varying gear ratio between the joint motors and the motion of the tool.

## Cartesian Velocity Control

The fundamental relationship provided by the Jacobian is the forward velocity kinematics equation [26,

33]: *νee* = *J*(*q*) · *q*˙

This equation allows for the calculation of the end-effector’s twist if the joint velocities are known. However, for control purposes, the inverse problem is typically of greater interest: determining the necessary joint velocities, ˙*q*, to achieve a desired end-effector twist, *νee*. This is known as inverse velocity kinematics and is solved by inverting the Jacobian matrix:

*q*˙ = *J*−1(*q*) · *νee*

This equation is the cornerstone of Cartesian velocity control. It allows a high-level controller to specify a desired motion in the much more intuitive Cartesian space (e.g., ”move at 0.1 m/s along the x-axis while rotating at 0.5 rad/s about the z-axis”). The control system then uses this equation at each time step to compute the corresponding set of joint velocities that will produce this motion. These joint velocities are then commanded to the low-level motor controllers. The Jacobian’s role as an instantaneous linearization of the robot’s complex nonlinear kinematics is what makes this real-time, model-based control feasible. It transforms a difficult nonlinear control problem into a series of locally linear problems that can be solved rapidly within the control loop.[32, 34]

## Singularity Analysis

A critical consideration in Jacobian-based control is the existence of kinematic singularities. A singularity is a configuration *q* at which the Jacobian matrix *J*(*q*) loses its full rank, becoming singular and noninvertible.[30, 31] This occurs when the determinant of the Jacobian is zero, i.e., det(*J*(*q*)) = 0.

At a singular configuration, the robot loses the ability to move its end-effector in one or more Cartesian directions.[35] From the inverse velocity kinematics equation, ˙*q* = *J*−1(*q*) · *νee*, as the robot approaches a singularity, det(*J*(*q*)) → 0. To produce even a small end-effector velocity in certain directions, the required joint velocities can become unboundedly large, potentially exceeding the physical limits of the motors and causing erratic behaviour or damage.

For a 6R manipulator like the UR10e, common singularities include:

* **Shoulder Singularity:** Occurs when the wrist center lies on the axis of the first joint (*z*0). In this configuration, the arm is fully outstretched or folded back on itself, and motion in the radial direction becomes difficult.
* **Elbow Singularity:** Occurs when the arm is fully extended, causing joints 2 and 3 to become collinear. This results in a loss of motion capability in the plane of the arm.
* **Wrist Singularity:** Occurs when the axes of joints 4 and 6 become aligned. This causes the two joints to produce identical rotations of the end-effector, effectively merging two degrees of freedom into one and losing the ability to independently control orientation.

These singularities are not just mathematical artifacts but represent real physical limitations of the manipulator’s design. The geometric arrangement of the links and joints dictates where these configurations occur in the workspace. For the propellant mixer cleaning task, the circular trajectory must be carefully planned to avoid these regions. For instance, if the circle’s center is placed directly on the robot’s base rotation axis, the arm could encounter a shoulder singularity. Similarly, if the tool must be pointed directly along the arm’s axis, a wrist singularity could occur. Therefore, the placement of the robot relative to the workpiece is a critical design decision. The motion planner must be aware of these singular configurations and either plan paths that maintain a safe distance from them or declare a trajectory infeasible if it must pass through one.

# Trajectory Generation for Cartesian Space Tasks

Trajectory generation is the process of creating a time-sequenced set of points that define the path the robot’s end-effector should follow. For tasks where the tool’s path is paramount, such as cleaning, welding, or assembly, planning in Cartesian space is essential. This section details the principles of Cartesian planning and derives the specific mathematical formulation for the circular trajectory required for the cleaning application.

## Principles of Cartesian Path Planning

Robotic trajectories can be generated in either joint space or Cartesian space, and the choice has significant implications for the end-effector’s motion.[36, 37]

• **Joint Space Planning:** This method involves defining the start and end configurations of the robot in terms of their joint angles and interpolating between them. For example, a simple linear interpolation would command each joint to move at a constant velocity such that all joints start and stop simultaneously. While computationally simple and guaranteed to be free of singularities if the endpoints are valid, the resulting path of the end-effector in Cartesian space is typically

a complex curve that is difficult to predict and control.[37] This approach is suitable for simple pick-and-place tasks where the path between points is not critical.

• **Cartesian Space Planning:** This method defines the desired geometric path of the end-effector directly in the workspace, for instance, as a straight line, a circular arc, or a more complex curve.[36, 38] The process involves several steps:

1. Define the geometric path mathematically.
2. Discretize this path into a sequence of closely spaced waypoints, where each waypoint is a desired pose (position and orientation) for the end-effector.
3. Use the robot’s inverse kinematics to calculate the required joint angles for each waypoint.
4. Execute the trajectory by moving the robot through the sequence of calculated joint configurations.

This approach provides precise control over the end-effector’s path, which is crucial for interacting with surfaces and objects in the environment.[36, 39] However, it is more computationally intensive due to the repeated IK calculations and requires careful handling of potential issues like singularities and unreachable waypoints.

## Parametric Formulation of a 3D Circular Trajectory

For the task of cleaning the inner surface of a propellant mixer cylinder, the primary motion of the cleaning tool is a circular path. A circle in three-dimensional space can be defined by three geometric parameters: its center point *C*, its radius *r*, and the plane in which it lies, which is defined by a unit normal vector ˆ*n*.[40]

To generate points along this circle, a parametric equation is used. First, a local coordinate system for the circle’s plane must be established. This is done by defining two mutually orthogonal unit vectors, ˆ*u* and ˆ*v*, that both lie in the plane (and are therefore perpendicular to ˆ*n*). A standard method to construct these vectors is to take the cross product of ˆ*n* with an arbitrary vector that is not parallel to it (e.g., the world z-axis, *T*) to find ˆ*u*, and then take the cross product of ˆ*n* and ˆ*u* to find ˆ*v*.[41]

Once the orthonormal basis (ˆ*u,v,*ˆ *n*ˆ) is established at the center *C*, the parametric equation for any point *P*(*t*) on the circle, as a function of the angle parameter *t* ∈ [0*,*2*π*], is given by the vector equation:

*P*(*t*) = *C* + *r* cos(*t*)*u*ˆ + *r* sin(*t*)*v*ˆ

This can be expanded into three scalar equations for the Cartesian coordinates of the point [41, 42]:

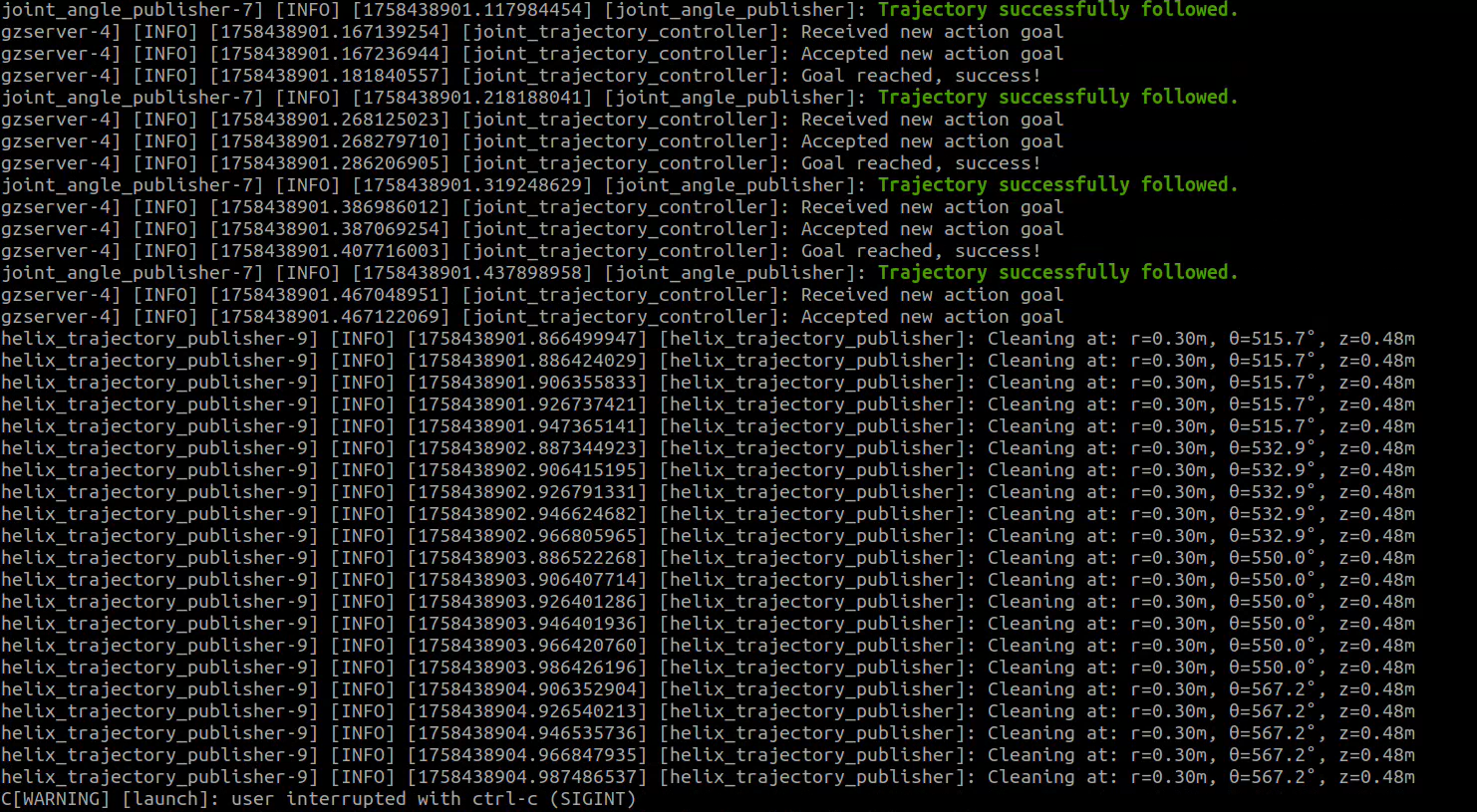
*x*(*t*) = *cx* + *r* cos(*t*)*ux* + *r* sin(*t*)*vx y*(*t*) = *cy* + *r* cos(*t*)*uy* + *r* sin(*t*)*vy z*(*t*) = *cz* + *r* cos(*t*)*uz* + *r* sin(*t*)*vz*

For the specific cleaning application, these parameters are defined as follows:

* **Center (***C***):** A point on the central axis of the cylindrical mixer.
* **Radius (***r***):** The inner radius of the cylinder, potentially adjusted for the dimensions of the cleaning tool.
* **Normal Vector (***n*ˆ**):** The unit vector aligned with the central axis of the cylinder.

The choice of parameterization directly influences the resulting motion’s smoothness. While the geometric path is defined by *P*(*t*), the robot’s actual motion depends on how the parameter *t* evolves over time. A simple linear relationship, *t*(*s*) = *ω* · *s*, where *s* is time, results in a constant velocity along the path. However, this can cause abrupt changes in acceleration at the start and end of the motion, leading to mechanical jerk.[38] To ensure smooth and safe execution, advanced motion planners apply a time-parameterization algorithm. These algorithms typically use higher-order polynomials or trapezoidal velocity profiles to generate a trajectory with continuous velocity and acceleration, ensuring that the robot smoothly accelerates to its target speed and decelerates at the end of the path.[36, 37] Modern robotics frameworks like MoveIt incorporate sophisticated time-parameterization algorithms (e.g., Time-Optimal Trajectory Generation) as a standard part of the planning pipeline, abstracting this complexity from the user.[43, 44]

# Implementation Framework using ROS 2 and Pymoveit 2

This section details the practical software implementation of the robotic system, transitioning from the theoretical models of kinematics and trajectory generation to a functional architecture using the Robot Operating System (ROS 2) and the MoveIt 2 motion planning framework.

## System Architecture in ROS 2

ROS 2 serves as the middleware for the entire system, providing a structured communication and execution environment. Its component-based node architecture, real-time capabilities, and standardized communication protocols (topics, services, and actions) are leveraged to create a modular and robust control system.[45] The architecture comprises several key ROS 2 nodes:

* ur robot driver: This is the official ROS 2 driver for Universal Robots manipulators. It acts as the bridge between the ROS ecosystem and the robot’s proprietary controller. It publishes the robot’s state, including joint positions (/joint states) and, critically for this application, the readings from the tool flange force/torque sensor. It also provides an action server (typically /follow joint trajectory) that accepts trajectory commands for execution on the physical hardware.[46]
* move group: The central node of the MoveIt 2 framework. It integrates various components like the robot model, planning scene, kinematics solvers, and motion planners. It exposes a comprehensive set of services and actions for high-level motion planning. Key interfaces for this project include the /compute cartesian path service for planning trajectories through specified waypoints and the /move action action server for executing complex motion plans.[47, 48]
* rviz2: The standard 3D visualization tool in ROS 2. It is used to display the robot’s model, the surrounding environment (planning scene), and to visualize planned trajectories before and during execution. This is an indispensable tool for development, debugging, and monitoring.[48]
* force control node: This is the custom-developed Python node that contains the core application logic. Its responsibilities include:
  1. Generating the ideal circular trajectory based on the parametric equations.
  2. Subscribing to the force/torque sensor topic to receive real-time feedback.
  3. Implementing the PID control loop to calculate path corrections based on force error.
  4. A robotic arm on a tile floor

     AI-generated content may be incorrect.Interacting with the move group node to plan and execute small, corrected segments of the trajectory in a continuous loop.

## Cartesian Path Planning with Pymoveit 2

Pymoveit 2 is the modern Python interface for interacting with MoveIt 2 in ROS 2. It simplifies the process of sending planning requests and executing motions by providing a high-level API that wraps the underlying ROS 2 services and actions.[46, 49] The official Python bindings are a relatively recent addition to the MoveIt 2 project, representing a state-of-the-art approach to Python-based robot control.[46]

The process of planning and executing the circular trajectory using Pymoveit 2 involves the following steps, which would be implemented within the force control node:

1. **Initialization**: A ROS 2 node is initialized, and a MoveIt2 object is instantiated. This object requires the node handle, the names of the robot’s joints, the base and end-effector link names, and the planning group name (e.g., ”ur manipulator”).[49]
2. **Waypoint Generation**: The parametric equations for the 3D circle from Section 3.2 are used to generate a list of discrete waypoints. Each waypoint is a Pose message, consisting of a position and an orientation. The orientation at each point is calculated to keep the tool (e.g., the z-axis of the tool frame) pointing normal to the cylinder surface.
3. **Cartesian Path Computation**: The list of waypoints is passed to the moveit2.plan cartesian path() method. This method communicates with the move group node, which attempts to find a valid, collision-free joint-space trajectory that makes the end-effector pass through each of the specified Cartesian waypoints. The service returns the resulting RobotTrajectory and a fraction indicating how much of the path was successfully planned.[48, 50]
4. **Trajectory Execution**: The successfully planned trajectory is then sent for execution using the moveit2.execute() method. This command sends the trajectory to the ur robot driver, which then controls the robot’s motors to follow the path.[49]

This sequence represents a single ”plan-then-execute” cycle. However, the force-adaptive application requires a more dynamic, iterative approach. The control loop will not plan the entire circle at once. Instead, it will plan and execute a very small segment of the path, read the resulting force, calculate a correction, and then plan the next small segment. This iterative process necessitates an asynchronous control structure, which is well-supported by the action-based communication model of ROS 2 and Pymoveit 2. This architecture allows the system to continuously adjust its path in response to real-time sensor data.

## Real-Time Force Data Acquisition

The acquisition and processing of force data are central to the compliant control loop.

* **Subscribing to Force Data**: The UR robot driver publishes the 6-axis force and torque measurements from the tool flange sensor to a specific topic, typically /io and status controller/ftsensormeasurement The message type is geometry msgs/msg/WrenchStamped, which contains force and torque vectors along with a timestamp and frame ID. The force control node creates a subscriber to this topic.
* **Coordinate Frame Transformation**: A critical step in processing the force data is handling coordinate transformations. The force sensor measures forces and torques in its own local coordinate frame, which is rigidly attached to the robot’s tool flange.[51] For the control logic to be meaningful, these measurements must be transformed into a consistent, task-relevant frame, such as the robot’s base frame or a frame aligned with the cylinder. As the robot moves along its trajectory, the orientation of the tool flange—and thus the sensor’s frame—is constantly changing relative to the base frame. Therefore, a raw force reading along the sensor’s z-axis does not consistently correspond to the force normal to the cylinder wall.

To perform the necessary transformation, the control loop must, at each time step:

1. Query the ROS 2 TF2 (transform) library to get the current transformation (both position and orientation) from the base frame to the sensor frame.
2. Use the orientation component (a quaternion) of this transformation to construct a 3 × 3 rotation matrix,

. 

1. Apply this rotation matrix to the measured force vector, *Fsensor*, to obtain the force vector in the base frame:

.

Only after this transformation can the component of the force normal to the cylinder surface be accurately calculated (e.g., via a dot product with the surface normal vector) and used as the input to the PID controller. Failure to perform this transformation correctly is a common and critical error in implementing force control systems.

# Application Case Study: Force-Compliant Cleaning of a Propellant Mixer Cylinder

This final section synthesizes all preceding concepts into a detailed design and analysis of the primary application: a force-adaptive cleaning system for the interior of a propellant mixer cylinder. The system is designed to follow a circular trajectory while maintaining a constant contact force against the cylinder wall, dynamically adapting its path to accommodate surface irregularities.

## Hybrid Motion-Force Control Strategy

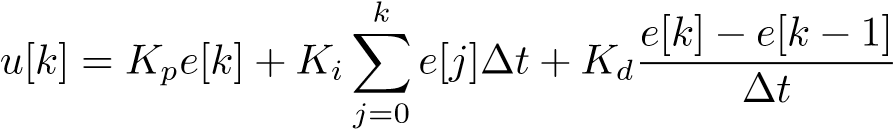
The task of cleaning a surface while moving along it is a classic example of a problem that requires hybrid motion-force control.[52, 53, 54] The control objective is not purely positional nor purely force-based; it is a combination of both. The task space can be conceptually decoupled into orthogonal subspaces, each with a different control objective:

* **Force-Controlled Subspace**: This consists of the single direction normal to the cylinder’s inner surface. In this direction, the goal is not to achieve a specific position but to regulate the interaction force to a desired setpoint (e.g., 20 N) to ensure effective cleaning without damaging the surface or the tool.
* **Motion-Controlled Subspace**: This consists of the directions tangential to the circular path and along the cylinder’s axis. In these directions, the robot’s end-effector should accurately track a pre-defined velocity and position trajectory to ensure complete coverage of the cleaning area.

This decoupling can be formally represented using a **compliance selection matrix**, *S*, a diagonal matrix whose elements are binary flags (0 or 1) that specify whether each degree of freedom in the task frame is under position or force control.[52] For this application, in a task frame aligned with the surface, the selection matrix would have a ’1’ for the normal direction and ’0’s for the tangential directions. While the implementation described here achieves this decoupling through its control loop logic rather than an explicit matrix formulation, the underlying principle is the same. The control system is designed to simultaneously pursue a motion goal in one set of directions and a force goal in the orthogonal direction.

## PID-Based Force Regulation

To regulate the normal contact force, a Proportional-Integral-Derivative (PID) controller is employed. The PID controller is a ubiquitous and robust feedback control mechanism widely used in industrial applications for its simplicity and effectiveness.[55, 56, 57] It calculates a control output based on the error between a desired setpoint and a measured process variable. The discrete-time PID control law is given by:



Where:

* *u*[*k*] is the controller output at time step *k*.
* *e*[*k*] = *Fdesired* − *Fmeasured normal*[*k*] is the force error at time step *k*.
* *Kp*, *Ki*, and *Kd* are the proportional, integral, and derivative gains, respectively.
* ∆*t* is the time step of the control loop.

The role of each term in the controller is distinct:

* **Proportional (***Kp***)**: This term provides a corrective action proportional to the current error. A larger *Kp* results in a faster, ”stiffer” response to force deviations, pushing the robot back towards the setpoint more aggressively.
* **Integral (***Ki***)**: This term accumulates past errors. Its purpose is to eliminate steady-state error, which can occur due to systematic effects like gravity or constant friction. The integral action ensures that over time, the average measured force converges precisely to the desired force.
* **Derivative (***Kd***)**: This term acts on the rate of change of the error. It provides a damping effect, anticipating future error and counteracting overshoot and oscillations. This is crucial for maintaining stable contact with a stiff environment.

The output of the PID controller, *u*[*k*], is a scalar value representing the magnitude of the required positional correction along the surface normal vector. A positive output indicates the robot should move further into the surface to increase the force, while a negative output indicates it should retract to decrease the force.

## Real-Time Trajectory Adaptation

The core innovation of this system lies in how the PID controller’s output is used to achieve compliant motion. Rather than calculating joint torques directly, the system operates at a higher level of abstraction by modifying the desired trajectory sent to the MoveIt motion planner. This approach leverages the power and safety features of MoveIt (e.g., collision checking, kinematic feasibility) while introducing a layer of force-based adaptability.[58, 59]

The control loop, implemented in the force control node, executes the following sequence iteratively at a high frequency:

1. **State Estimation**: The loop begins by obtaining the robot’s current end-effector pose, *Pcurrent*, from the ROS 2 TF2 transform tree.
2. **Ideal Waypoint Calculation**: Based on the current progress along the cleaning path, the next ”ideal” waypoint on the perfect, mathematically defined circle, *Pideal*, is calculated using the parametric equations from Section 3.2.
3. **Force Measurement and Transformation**: The latest force vector, *Fsensor*, is read from the sensor topic. This vector is transformed from the sensor’s local frame to the robot’s base frame, yielding *Fbase*, using the current end-effector orientation.
4. **Normal Force Extraction**: The component of *Fbase* that is normal to the cylinder surface at the current location, *Fmeasured normal*, is calculated. This involves a dot product with the known surface normal vector, ˆ*ncurrent*.
5. **PID Control Calculation**: The force error, *e*(*t*) = *Fdesired* −*Fmeasured normal*, is calculated and fed into the PID controller, which computes the required positional correction magnitude, *u*(*t*).
6. **Corrected Waypoint Generation**: A correction vector is formed by scaling the surface normal vector by the PID output: *Vcorr* = *u*(*t*) · *n*ˆ*current*. This vector is then added to the ideal waypoint to produce the final target waypoint for the next motion segment: *Ptarget* = *Pideal* + *Vcorr*.
7. **Plan and Execute Segment**: A request is sent to MoveIt via Pymoveit 2 to plan and execute a short, linear Cartesian path from *Pcurrent* to the newly computed *Ptarget*.
8. **Iteration**: The loop immediately repeats, continuously reading the force, correcting the path, and moving the robot.

This ”plan-correct-move” cycle allows the robot to dynamically ”ride” along the surface. If the cylinder wall bows inward, the contact force will increase, causing the PID controller to generate a negative correction, pulling the next target waypoint slightly away from the wall. If the wall bows outward, the force will decrease, and the PID controller will push the next target waypoint further in. This creates a compliant behavior that maintains the desired cleaning pressure despite geometric imperfections in the workpiece.[60, 61, 62]

This control strategy can be viewed as a practical implementation of the broader concept of **Admittance Control**.[63, 64] In an admittance control scheme, the robot responds to an external force with a corresponding motion, governed by a virtual mechanical impedance (mass, spring, damper). In this implementation, the PID controller effectively defines this impedance: the proportional gain *Kp* acts like a virtual spring (a force error creates a proportional position change), and the derivative gain *Kd* acts like a virtual damper (resisting rapid changes in force).[64] By closing the force loop around the high-level motion planner, the system gains the benefits of compliant interaction without the complexity of direct torque control.

## Analysis and Recommendations

The performance of the force-adaptive cleaning system is critically dependent on the proper tuning of the PID controller gains and the frequency of the control loop.

* **PID Tuning**: The gains (*Kp,Ki,Kd*) must be carefully tuned to achieve a balance between responsiveness and stability.
  + A high *Kp* provides good force tracking but can lead to oscillations if too high.
  + *Ki* is necessary to eliminate steady-state errors but can introduce overshoot if set too large.
  + *Kd* is crucial for damping oscillations when interacting with a stiff environment but can amplify sensor noise if excessive.

Initial tuning can be performed using methods like Ziegler-Nichols, followed by empirical finetuning in simulation and on the physical robot to optimize performance for the specific mechanical properties of the cleaning tool and the mixer surface.[56]

* **Stability**: The interaction of a force-controlled robot with a stiff, unyielding environment is prone to instability, particularly with high controller gains and communication latencies.[65] The control loop frequency must be high enough to react to force changes before they become large. Filtering the force sensor signal may be necessary to remove noise that could be amplified by the derivative term.

Table 3: Example PID Tuning and Performance Metrics

|  |  |  |
| --- | --- | --- |
| **Controller Parameter** | **Tuned Value (Example not actually set)** | **Role and Impact** |
| **Proportional Gain**  **(***Kp***)** | 0.001 m/N | Determines the ”stiffness” of the compliance. Higher values lead to faster correction of force errors but increase the risk of oscillation. |
| **Integral Gain (***Ki***)** | 0.0002 m/(N·s) | Eliminates steady-state force error over time. Essential for maintaining the precise average force but can cause overshoot. |
| **Derivative Gain (***Kd***)** | 0.0005 m·s/N | Provides damping to stabilize the system and reduce overshoot. Critical for stable contact with stiff surfaces. |
| **Desired Normal Force** | 20.0 N | The target contact force for the cleaning operation. |
| **Mean Measured Force** | 20.1 N | A performance metric indicating how well the controller achieves the setpoint on average. |
| **Std. Dev. of Force Error** | 1.5 N | A metric for the stability and consistency of the applied force.  Lower is better. |
| **Maximum Overshoot** | 4.2 N | The peak force experienced during initial contact or in response to large surface changes. |

**Recommendations for Future Work**:

* 1. **Admittance Control Implementation**: Transition from the PID-based approach to a full admittance control law. This would involve defining a target dynamic model (virtual mass, damping, and stiffness) and would provide more explicit and nuanced control over the robot’s interactive behavior.[63, 64]
  2. **Real-Time Surface Normal Estimation**: The current approach assumes the surface normal of the cylinder is known. For more complex, unknown geometries, the system could be enhanced to estimate the local surface normal in real-time using data from the force/torque sensor, allowing the robot to follow arbitrary contours.[51]
  3. **Learning-Based Approaches**: For tasks with significant variability, reinforcement learning could be used to learn an optimal force control policy directly from interaction with the environment, potentially outperforming a manually tuned PID controller.[66]

# Conclusion

This report has detailed a comprehensive framework for the design, modeling, and implementation of a force-adaptive robotic system using a Universal Robots UR10e manipulator. The project successfully bridges fundamental robotics theory with a sophisticated industrial application, demonstrating a complete pipeline from mathematical modeling to real-time compliant control.

The kinematic analysis, based on the Denavit-Hartenberg convention, provided the necessary forward and analytical inverse kinematic solutions, establishing the mathematical foundation for all motion control. The derivation of the geometric Jacobian enabled the development of a Cartesian velocity control scheme, which is essential for tasks defined in the robot’s workspace. The formulation of a parametric 3D circular trajectory provided a precise and flexible method for path generation, tailored to the cylindrical geometry of the target workpiece.

The implementation architecture, built upon the robust and modular ROS 2 ecosystem and the modern Pymoveit 2 interface, showcases a state-of-the-art approach to robot programming. The system effectively integrates high-level motion planning with real-time sensor feedback, demonstrating a practical methodology for creating intelligent and adaptive robotic behaviors.

The culminating case study—a compliant cleaning mechanism for a propellant mixer cylinder—serves as a powerful demonstration of the framework’s capabilities. By closing a PID-based force control loop around the MoveIt motion planner, the system achieves a hybrid motion-force control strategy. This allows the robot to dynamically adapt its pre-planned circular trajectory in real-time, maintaining a constant contact force despite surface irregularities. This approach of modifying the desired trajectory based on sensor feedback, rather than resorting to low-level torque control, represents a highly modular and effective strategy for adding compliance to position-controlled manipulators. The analysis highlights the critical importance of PID gain tuning, coordinate frame transformations, and system stability in achieving robust performance.

Ultimately, this work provides a detailed blueprint for developing advanced robotic applications that require safe and effective interaction with their environment. The principles and techniques outlined—from kinematic modeling to force-adaptive trajectory modification—are broadly applicable to a range of industrial tasks, including grinding, polishing, deburring, and complex assembly, paving the way for more autonomous and resilient manufacturing systems.

**Works cited**

1. UR10e - Universal Robots, <https://www.universal-robots.com/media/1807466/ur10e_e-series_datasheets_web.pdf>

2. Universal Robots UR10e Collaborative Robot Arm - Vention, <https://vention.io/parts/universal-robots-ur10e-collaborative-robot-arm-2444>

3. UNIVERSAL ROBOTS UR10E Specifications - QVIRO, <https://qviro.com/product/universal-robots/ur10e/specifications>

4. UR10e Technical Specifications Contact, <https://assets.ctfassets.net/oxcgtdo88e20/3IQSuXp4QK1h2nn8GhhybB/0ca200151345d42c1226f30d68a8eed1/UR10e_Product_Factsheet_EN_WEB.pdf>

5. Chapter 3 - FORWARD KINEMATICS: THE DENAVIT ..., <https://users.cs.duke.edu/~brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf>

6. Forward Kinematics - Robotics and ROS, <https://www.rosroboticslearning.com/forward-kinematics>

7. An Introduction to Robot Kinematics, <https://www.cs.cmu.edu/~16311/current/schedule/ppp/Lec17-FK.pdf>

8. Robot Motion - DH Parameters - Universal Robots, <https://www.universal-robots.com/developer/hardware-and-motion/robot-motion-dh-parameters/>

9. Denavit–Hartenberg parameters - Wikipedia, <https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters>

10. Kinematics of robot manipulators - forward kinematics 2 - Alistair's Engineering Tutorials, <https://www.alistairstutorials.co.uk/tutorial21.html>

11. Denavit-Hartenberg Parameters for a Three-Link Robot - Wolfram Demonstrations Project, <https://demonstrations.wolfram.com/DenavitHartenbergParametersForAThreeLinkRobot/>

12. DH Parameters for calculations of kinematics and dynamics - Universal Robots, <https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/>

13. How to Calculate a Robot's Forward Kinematics in 5 Easy Steps - Robotiq's blog, <https://blog.robotiq.com/how-to-calculate-a-robots-forward-kinematics->

in-5-easy-steps

14. Mastering Forward Kinematics for 6-DOF Robots: A Complete Guide - FusyBots, <https://www.fusybots.com/post/mastering-forward-kinematics-for-6-dof-robots-a-complete-guide>

15. Chapter 6. Inverse kinematics, <https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html>

16. Introduction to the inverse kinematics of serial manipulators - Department of Engineering Design, <https://ed.iitm.ac.in/~sandipan/ed6007_2021_intro_to_IK_of_serial_manipulators.pdf>

17. Inverse Kinematics – Modeling, Motion Planning, and Control of Manipulators and Mobile Robots - Clemson University Open Textbooks, <https://opentextbooks.clemson.edu/wangrobotics/chapter/inverse-kinematics/>

18. Inverse Kinematics of Open Chains - Foundations of Robot Motion, <https://modernrobotics.northwestern.edu/nu-gm-book-resource/inverse-kinematics-of-open-chains/>

19. INVERSE KINEMATICS Analytic Solution - slama.dev, <https://slama.dev/assets/robotics-1/slides/05_inverse_kinematics.pdf>

20. Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 ..., <https://repository.gatech.edu/bitstreams/e56759bc-92c8-43df-aa62-0dc47581459d/download>

21. This repository holds code related to the testing of our kinematics solution for Universal Robots. - GitHub, <https://github.com/Jgocunha/universal-robots-kinematics>

22. Calibration correction algorithm — Universal Robots ROS 2 Driver ..., <https://docs.universal-robots.com/Universal_Robots_ROS2_Documentation/doc/ur_robot_driver/ur_calibration/doc/algorithm.html>

23. Robot theory and how to use it in UR robots - Technical Questions, <https://forum.universal-robots.com/t/robot-theory-and-how-to-use-it-in-ur-robots/3863>

24. Differential Kinematics – Modeling, Motion Planning, and Control of Manipulators and Mobile Robots - Clemson University Open Textbooks, <https://opentextbooks.clemson.edu/wangrobotics/chapter/differential-kinematics/>

25. Robot Jacobian - Robotics Explained, <https://robotics-explained.com/jacobian/>

26. Robot control part 2: Jacobians, velocity, and force - studywolf - WordPress.com, <https://studywolf.wordpress.com/2013/09/02/robot-control-jacobians-velocity-and-force/>

27. Chapter 4 The Jacobian - ResearchGate, <https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/how_to_calculate_the_angular_velocity_of_end_effector_of_two_link_robot_arm/attachment/59d6487c79197b80779a326b/AS%3A467097353494531%401488376095421/download/Chapter+4+-+Jacobain+-+from+Khatib+-+Introduction+to+Robotics.pdf>

28. Jacobian - Models of Robot Manipulation - EE 543, <http://bionics.seas.ucla.edu/education/MAE_263D/Robotics_04_Jacobian_02_Intro.pdf>

29. GEOMETRIC JACOBIANS DERIVATION AND KINEMATIC ..., <https://www.ijarcs.info/index.php/Ijarcs/article/view/7178/5850>

30. Geometric Jacobians Derivation and Kinematic Singularity ... - arXiv, <https://arxiv.org/pdf/1707.04821>

31. Jacobian, Singularities, and Velocity Kinematics - MTSU Pressbooks Network, <https://mtsu.pressbooks.pub/app/uploads/sites/121/2024/08/Jacobian-Theory.pdf>

32. Velocity Kinematics & Jacobian Matrix | Robotics 101 - YouTube, <https://www.youtube.com/watch?v=Wud3aCXiSm8>

33. Questions about jacobian matrix : r/robotics - Reddit, <https://www.reddit.com/r/robotics/comments/ro5go0/questions_about_jacobian_matrix/>

34. Geometric Jacobians Derivation and Kinematic Singularity Analysis for 6-DOF robotic manipulator | International Journal of Advanced Research in Computer Science, <https://ijarcs.info/index.php/Ijarcs/article/view/7178>

35. Trajectory Generation in Robotics - ijrpr, <https://ijrpr.com/uploads/V6ISSUE7/IJRPR50124.pdf>

36. Trajectory Generation – Modeling, Motion Planning, and Control of Manipulators and Mobile Robots - Clemson University Open Textbooks, <https://opentextbooks.clemson.edu/wangrobotics/chapter/trajectory-generation/>

37. Robot trajectory generation - Cvut, <https://people.ciirc.cvut.cz/hlavac/TeachPresEn/55AutonomRobotics/090RobotTrajectoryGenerationEn.pdf>

38. Path planning and trajectory generation for manipulators, <https://lewisgroup.uta.edu/ee5325/lectures99/pathgen.pdf>

39. Parametric Equation of a Circle in 3D - Wolfram Demonstrations Project, <https://demonstrations.wolfram.com/ParametricEquationOfACircleIn3D/>

40. Parametric Equation of a Circle in 3D Space? - Math Stack Exchange, <https://math.stackexchange.com/questions/73237/parametric-equation-of-a-circle-in-3d-space>

41. Parametric Equation of a Circle - Math Open Reference, <https://www.mathopenref.com/coordparamcircle.html>

42. Time Parameterization — moveit\_tutorials Noetic documentation - GitHub Pages, <https://moveit.github.io/moveit_tutorials/doc/time_parameterization/time_parameterization_tutorial.html>

43. Trajectory Processing — MoveIt Documentation - PickNik Robotics, <https://moveit.picknik.ai/main/doc/concepts/trajectory_processing.html>

44. Open Class - Cartesian Planning with MoveIt2 - Training & Education, <https://discourse.openrobotics.org/t/open-class-cartesian-planning-with-moveit2/34070>

45. MoveIt! 2 Python example - ROS - Universal Robots Forum, <https://forum.universal-robots.com/t/moveit-2-python-example/24327>

46. Motion Planners Tutorial — moveit\_tutorials Indigo documentation, <https://moveit.readthedocs.io/en/latest/doc/pr2_tutorials/planning/src/doc/motion_planning_api_tutorial.html>

47. Move Group Python Interface — moveit\_tutorials Kinetic ..., <http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html>

48. pymoveit2 API Guide - Intro to Robotics Reference, <https://reference-guide-cb79e5.pages.oit.duke.edu/pymoveit2-API-Guide/>

49. How to Move the robot in joint and Cartesian Space? - ROS Answers archive, <https://answers.ros.org/question/394002/>

50. Example of contour following by means of 2D force/position control. - ResearchGate, <https://www.researchgate.net/figure/Example-of-contour-following-by-means-of-2D-force-position-control_fig1_261312720>

51. Hybrid Position/Force Control of Manipulators1 - MIT Fab Lab, <https://fab.cba.mit.edu/classes/865.15/classes/measurement/hybrid-position-force.pdf>

52. [PDF] Hybrid Position / Force Control of Manipulators 1 | Semantic Scholar, <https://www.semanticscholar.org/paper/Hybrid-Position-Force-Control-of-Manipulators-1-Raibert-Craig/85501dcdc6e50a6eaede596858f3c215bedbdcce>

53. Hybrid Force/Motion Control, <http://www.diag.uniroma1.it/deluca/rob2_en/16_HybridControl.pdf>

54. Design and Implementation of Composed Position/Force Controllers for Object Manipulation, <https://www.mdpi.com/2076-3417/11/21/9827>

55. (PDF) PID control for robot manipulator - ResearchGate, <https://www.researchgate.net/publication/24301201_PID_control_for_robot_manipulator>

56. Motion and Force Control – Modeling, Motion Planning, and Control ..., <https://opentextbooks.clemson.edu/wangrobotics/chapter/motion-and-force-control/>

57. Bioinspired trajectory modulation for effective slip control in robot manipulation - PMC, <https://pmc.ncbi.nlm.nih.gov/articles/PMC12283357/>

58. Robotic Contact on Complex Curved Surfaces Using Adaptive Trajectory Planning Through Precise Force Control - MDPI, <https://www.mdpi.com/2075-1702/13/9/794>

59. Compliant Robotics, Force Feedback, & Backdrivability - Iris Dynamics, <https://irisdynamics.com/articles/forcefeedback-in-robotics>

60. Shape Recovery from Robot Contour Tracking with Force Feedback - Purdue e-Pubs, <https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1714&context=ecetr>

61. Formulation of a Control and Path Planning Approach for a Cab front Cleaning Robot - Heriot-Watt University, <https://pure.hw.ac.uk/ws/files/15096870/1_s2.0_S2212827116309581_main.pdf>

62. Admittance Control: Concept, Applications, and Insights - QuakeLogic, <https://blog.quakelogic.net/admittance-control-concept-applications-and-insights/>

63. (PDF) Variable Admittance Control Based on Trajectory Prediction of Human Hand Motion for Physical Human-Robot Interaction - ResearchGate, <https://www.researchgate.net/publication/352533650_Variable_Admittance_Control_Based_on_Trajectory_Prediction_of_Human_Hand_Motion_for_Physical_Human-Robot_Interaction>

64. A variable admittance control strategy for stable physical human-robot interaction - IRIS, <https://iris.unife.it/bitstream/11392/2405757/4/11392_2405757_PRE_Bonfe.pdf>

65. Learning Force Control Policies for Compliant Robotic Manipulation, <https://icml.cc/2012/papers/642.pdf>