Syrian Arab Republic
Ministry of Higher Education and Scientific Research
Manara University-Faculty of Engineering
Robotic and Intelligent Systems Department

Simulation of a Mobile Manipulator for Pick-and-Place Tasks in Static Environments

Prepared by:

Beilassan Hdewa
Lana Al wazzeh
Zain Alabidin Shbani

Supervised by

PhD. Mohamad Kheir Abdullah Mohamad, PhD. Essa Alghannam

2024-2025

ملخص

يتناول هذا المشروع تطوير محاكاة لمنصة روبوتية متنقلة تعتمد على روبوت KUKA youBot، بهدف تنفيذ مهام الالتقاط والوضع في بيئات ساكنة ومنظمة. تم تزويد المنصة بقاعدة متحركة تعتمد على عجلات mecanum، ما يتيح ثلاث درجات حرية للحركة على المستوى الأفقي: الانتقال الطولي (على المحور X)، الانتقال الجانبي(على المحور Y)، والدوران حول المحور الرأسي(Yaw) ، دون الحاجة إلى إعادة توجيه جسم الروبوت بالكامل. هذه القدرات تمنح النظام قدرة عالية على المناورة في المساحات الضيقة والبيئات المعقدة. كما يحتوي الروبوت على ذراع آلية بخمس درجات حرية، تسمح بالتحكم الكامل في موضع واتجاه الـ (End-Effector). يُعد هذا النوع من الأذرع مثاليًا للتطبيقات في البيئات الصناعية أو التعليمية المقيدة مثل خطوط التجميع المؤتمتة، مختبرات الأبحاث الجامعية، ومحطات الفرز والتغليف، حيث تتطلب المهام دقة عالية ضمن حيّز مكاني محدود.

تم تطوير نموذج حركي وديناميكي متكامل لكل من القاعدة والذراع، مع اعتماد استراتيجية تحكم تعتمد على تغذية راجعة للحالة باستخدام مراقب (Observer) وتقنية وضع الأقطاب Pole) (Placement، مما يضمن استجابة ديناميكية مستقرة وتتبع دقيق للمسارات المرجعية. وقد تم توليد هذه المسارات باستخدام برنامج MATLAB، وتخزينها في ملفات بصيغة CSV، بحيث يحتوي كل ملف على التكوين الكامل للروبوت في كل لحظة زمنية، بما في ذلك زاوية دوران القاعدة، موضعها، زوايا المفاصل، زوايا العجلات، وحالة القبضة.

تم تنفيذ المحاكاة باستخدام CoppeliaSim ضمن المشهد المُعد مسبقًا Scene6_youBot_Cube، والذي يحتوي على نموذج كامل للروبوت youBot بخصائص ديناميكية ومكعب مادي تفاعلي. أتاح هذا المشهد دمجًا مباشرًا لأكواد التحكم دون الحاجة إلى إعادة بناء النموذج يدويًا. تم تنفيذ المسارات المستخرجة من ملفات CSV خطوة بخطوة وفي الزمن الحقيقي باستخدام المحرك الفيزيائي ODE. وللتحقق من دقة النموذج الميكانيكي بشكل مستقل عن المشهد الجاهز، تم إجراء محاكاة بصرية ستاتيكية باستخدام RViz في بيئة ROS1 Noetic. تم تصميم النموذج الميكانيكي الأولي للروبوت، بما في ذلك الأبعاد الدقيقة للوصلات وهيكل القاعدة، باستخدام برنامج CATIA V5. وقد ضمنت عملية النمذجة عبر هذا البرنامج دقة الأشكال الهندسية قبل تحويل التصميم إلى صيغة URDF/XACRO الخاصة بمحاكاة ROS. تم إنشاء مشهد مستقل لعرض النموذج الكامل بصيغة

URDF/XACRO للروبوت youBot، وقد تم من خلاله التأكد من تطابق أبعاد الوصلات، ومواضع المفاصل، ومواقع العجلات مع التصميم المقصود. مكّن هذا الإعداد من التحقق الصلب من البنية الفيزيائية للنموذج .

أظهرت نتائج المحاكاة في CoppeliaSim قدرة النظام على تنفيذ عمليات الالتقاط والوضع المنسقة بدقة عالية، دون تجاوز حدود المفاصل أو فشل في التحكم. ويُبرز المشروع نجاح الدمج بين تخطيط الحركة، التحكم المعتمد على المراقب، والمحاكاة متعددة البيئات، مما يمهد الطريق لتطبيقات روبوتية ذكية وقابلة للتوسّع في البيئات الصناعية والتعليمية المتقدمة.

Abstract

This project presents the development of an entire simulation framework for a mobile robot platform based on the KUKA youBot, to perform pick-and-place tasks in static and structured environments. The platform is equipped with a Mecanum-wheeled base, enabling three degrees of planar freedom of movement: translation forward and backward along the X-axis, lateral translation along the Y-axis, and rotation about the vertical axis (Yaw) without requiring complete body reorientation. This provides the robot with great maneuverability in confined spaces and complex environments. The robot is also equipped with a 5-DOF robotic manipulator, providing full end-effector position and orientation control. This kind of manipulator is perfect for application in confined industrial or educational environments like automated assembly lines, university research labs, and sorting or packaging stations.

A complete kinematic and dynamic model was developed for the base and the arm, whereas the control approach was Observer-Based State Feedback via Pole Placement. This ensures stable dynamic response and accurate trajectory tracking. All reference trajectories were generated in MATLAB and stored as CSV files, each of which corresponds to the overall configuration of the robot at every time step, i.e., chassis orientation, base position, joint angles, wheel angles, and gripper state.

Simulation was performed with Scene6_youBot_Cube in CoppeliaSim, which is a preconfigured scene that contains a full youBot model with dynamic properties and an interactive physical cube. The scene allowed for direct integration of control scripts without manual reconstruction of the robot. The trajectories of the CSV files were executed step-by-step in real time using the ODE physics engine. To verify the accuracy of the mechanical model independently from the preconfigured scene, a static visualization was conducted using RViz in the ROS1 Noetic environment. The initial mechanical design of the robot, including precise link dimensions and chassis structure, was modeled using CATIA V5. This CAD modeling process ensured accurate geometry before translating the design into URDF/XACRO format for ROS-based simulations. An independent scene was created to visualize the full URDF/XACRO model of the youBot, confirming that link sizes, joint positions, and wheel locations were consistent with the intended design. This setup enabled a solid confirmation of the robot's geometry.

The simulation results from CoppeliaSim confirmed the system's ability of conducting coordinated pick-and-place operations with high precision and

without joint limit violation or control failure. The project demonstrates successful integration of motion planning, observer-based control, and multi-environment simulation, and paves the way for scalable implementation of intelligent mobile manipulation systems in high-end industrial and educational environments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

MM Mobile **M**anipulator

DOF Degrees **o**f **F**reedom

CS Configuration **S**pace

TS Task **S**pace

FK Forward **K**inematics

IK Inverse **K**inematics

RPY Roll-**P**itch-**Y**aw

1. Introduction

In modern automation ecosystems, the demand for robotic systems capable of performing complex manipulation tasks in dynamic and semi-structured environments has been steadily increasing. However, even in structured static environments such as factory floors, laboratories, and logistics warehouses, the need for flexible, mobile, and autonomous solutions remains critical. Robots in such contexts must navigate precisely, identify and locate target objects, and execute manipulation tasks with accuracy and consistency, all without direct human intervention. To meet these challenges, mobile manipulators have emerged as powerful robotic platforms that combine the spatial reach of mobile bases with the dexterity of articulated robotic arms.

Among these, the KUKA YouBot, a 5-degree-of-freedom mobile manipulator with an omnidirectional base, offers a compact yet capable platform suitable for a wide range of manipulation and navigation tasks. Its design enables it to operate efficiently in confined spaces, making it a suitable candidate for automated pick-and-place operations in structured static environments where objects are located at known or predictable positions but still require precise handling and coordination.

Developing and deploying such robotic systems in real-world settings, however, presents considerable challenges. Integration of motion planning, localization, perception, and control in a single coordinated framework is non-trivial, especially when seeking to ensure smooth interaction between the navigation and manipulation subsystems. Moreover, real-world testing can be time-consuming, risky, and expensive, particularly during the early design and development phases.

For these reasons, simulation plays a central role in robotic system design. By leveraging simulated environments, researchers and engineers can rigorously test control algorithms, validate kinematic and dynamic models, fine-tune planning strategies, and anticipate integration issues, all before deploying a single physical component. Simulation enables safe, rapid, and cost-effective iteration, serving as a bridge between conceptual design and physical implementation.

This project aims to develop a comprehensive simulation of the KUKA YouBot platform. The objective is to design and validate an autonomous system capable of performing pick-and-place tasks within structured environments, using a modular simulation framework that integrates:

- full-body kinematic and dynamic modeling,
- motion planning for both the mobile base and manipulator arm,
- perception and localization,
- and closed-loop control strategies.

Through systematic testing and performance evaluation across various scenarios, this simulation provides a reliable foundation for future physical deployment. Additionally, it serves as a reference model for researchers and practitioners seeking efficient mobile manipulation solutions in semi-industrial or academic contexts.

1.1. Problem Statement:

Autonomous robots must handle real-world variability. A mobile manipulator operating in structured and static environments must integrate navigation, object detection, and precise manipulation while maintaining reliability. Developing such a system in simulation helps validate control strategies before physical implementation.

1.2. Research Question:

How can a mobile manipulator effectively perform pick-and-place operations in structured static environments?

1.3. Aims of The Project

The primary aim of this project is to develop and validate a comprehensive simulation of a mobile manipulator for reliable and accurate pick-and-place operations in structured static environments in an autonomous setting. This simulation would serve as an intermediary step toward the practical implementation of the manipulator after having developed a more valid framework with kinematic modeling, motion planning, perception, and control strategies.

1.4. Objectives:

- Develop a simulation of a mobile manipulator to perform and evaluate pick-and-place tasks in structured static environments.
- Model the complete kinematics and dynamics of both the mobile base and the manipulator arm.
- Implement and test suitable control strategies for integrated navigation and manipulation.
- Simulate perception, localization, and motion planning
- Generate smooth trajectories for the end-effector and base using suitable planning algorithms.
- Evaluate the robot's performance in varying static environments.
- Outline the basic mechanical structure and electrical system components for future physical implementation.

1.5. Significance of the Study

This study contributes to the growing field of intelligent mobile robotics by addressing the challenges of coordinated mobility and manipulation in controlled environments. The ongoing simulation of this system allows the researcher to

bring into perspective a wider variety of design options and to check performance under a number of constraints beyond the limitations of hardware. The findings obtained from this simulated environment would, in later stages, feed the design of an actual prototype. The project is also a reference framework for researchers and engineers working on task automation in semi-structured or industrial domains.

2. Literature Review

Mobile manipulators integrate mobility and manipulation so that robots can carry out tasks needing both navigation and object handling. In a static environment, these systems commonly support structured tasks, such as pick-and-place operations in industrial and laboratory settings. This review highlights recent developments in the simulation, design, and control of mobile manipulators, emphasizing their usefulness in executing pick-and-place operations in static and controlled environments.

2.1. Background and Early Developments

In the last decade, mobile manipulators have evolved considerably in terms of their design and their functionality. Early models were often limited to either mobility or manipulation coordination, but then research and industrial advancements have rolled out with more sophisticated control systems, modular structures, and improved perception capabilities over time. Figure 1 summarizes key developments in mobile manipulators from 2009 to 2021, highlighting notable platforms that laid the groundwork for today's advanced systems used in simulation and real-world applications [1].

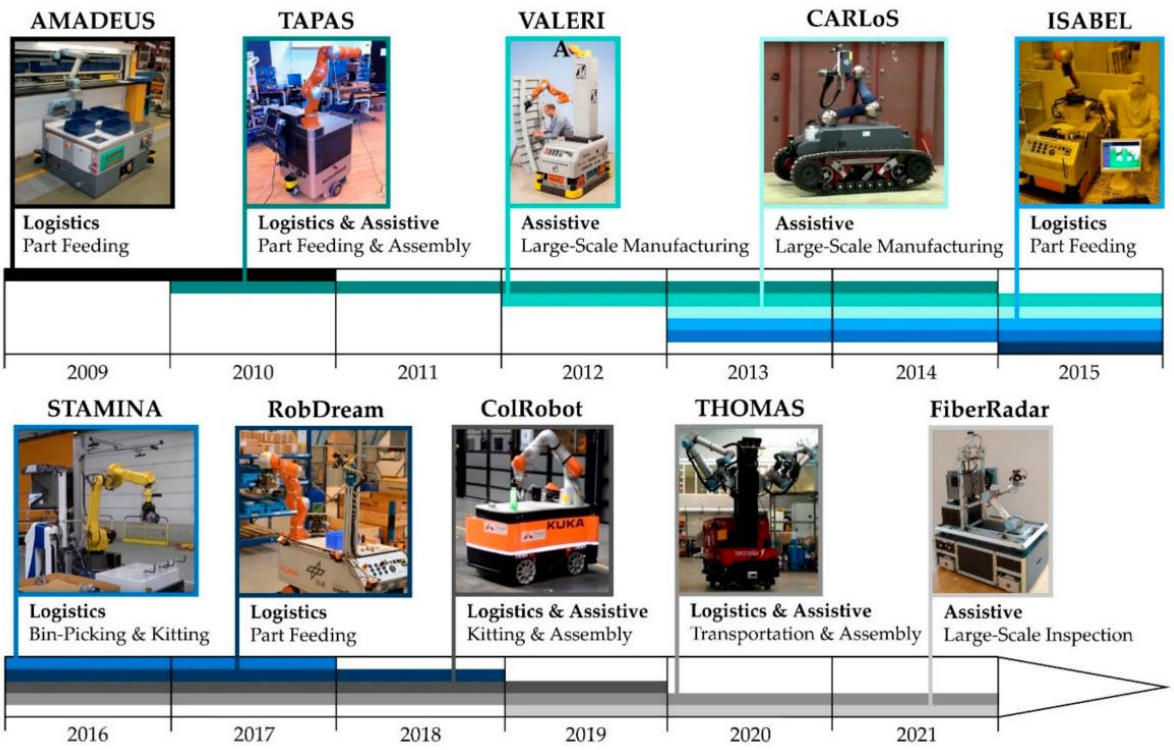


Figure 1: Related research projects from 2009 to 2021 [1]

2.2. Mobile Manipulator Architecture:

The mobile base and manipulator arm require a highly interactive design due to their requirements for stability and dexterity. Engemann et al. developed a holonomic mobile manipulator, OMNIVIL, meant for flexible production environments. The omnidirectional movement and modular hardware provide more adaptability and accuracy in tight areas [1]. Similarly, Datta et al. built an autonomous mobile manipulator for manufacturing. The robot navigates and handles objects in an industrial environment using proprioceptive and exteroceptive sensors, including stereo-vision cameras [2]. Stretch is a lightweight mobile manipulator intended for domestic environments, developed by Kemp et al.; it has a telescoping robotic arm mounted on wheels for navigation through crowded indoor areas [3]. Oftadeh et al. proposed a system architecture that integrates a 6-DOF manipulator with a four-wheeled mobile platform in real-time on a Linux operating system with MATLAB/Simulink used for deterministic and accurate manipulation [4].

2.3. Perception and Environmental Understanding

Perception is very crucial for the complete autonomous operation. Yamamoto et al. developed the Human Support Robot (HSR), a small and safe platform to use in domestic environments. HSR employs a dual-wheel caster-drive mechanism and a whole-body motion control system, enabling it to perform tasks like opening drawers and fetching objects [5]. Ciocarlie et al. exhibited a mobile manipulation platform driven by motor-disordered individuals resorting to head-tracking and one-button interfaces: how autonomous sub-modules can enhance performance in a cluttered environment [6]. Recently, researchers began installing advanced vision systems on mobile manipulators to execute real-time manipulation on moving objects, such as items on conveyor belts, an essential feature for flexible manufacturing systems [7]. The use of soft tactile sensors integrated into robotic grippers has allowed more delicate and adaptive handling of objects, especially important for processes like logistics and warehouse automation, where variability between objects is high [8].

2.4. Path Planning and Navigation

Navigating in dynamic environments is an inherently difficult problem. Holmberg and Khatib designed a holonomic mobile robot that consists of a powered caster vehicle (PCV) base that enables smooth and precise motion coordination along with manipulation by the onboard manipulator [9]. Xu et al. proposed a method to plan a sequence of base positions for a mobile manipulator performing multiple pick-and-place tasks, enhancing efficiency by removing redundant motions and

optimizing reachability [10]. An integrated reactive control scheme proposed by Haviland et al. results in considering the arm and the base as a single entity, which enhances navigation and manipulation capabilities in a dynamic situation [11]. M2Diffuser is a cutting-edge, diffusion-based**,** scene-conditioned generative model tailored for mobile manipulation planning in 3D environments. It is specifically designed to produce coordinated whole-body trajectories, including both the mobile base and the manipulator arm, based on 3D sensory input. [12].

2.5. Applications and Use Cases Mobile

manipulators have applications in many fields. For instance, in industrial applications, robots such as OMNIVIL and the robot mobile AMR developed by Datta et al. perform logistics and assembly tasks [1], [2]. In domestic settings, HSR assists in daily activities and improves the quality of life for the elderly and the people with disabilities [5]. Assistive robots empower users with limited mobility to interact with their environment effectively [6]. TidyBot++, an open-source holonomic mobile manipulator, handles teleoperation and demonstration collection, speeding up policy learning in domestic task automation [13]. In He et al.'s the dynamic human-robot collaboration, through "on-the-go" handovers, a mobile robot hands over an object to human drivers without stopping, employing a control system for synchronization of the manipulator and the platform for smooth object transfer [14].

2.6. Industry 4.0 and Future Directions

The role of mobile manipulators in Industry 4.0 was studied, emphasizing the relevance of their compatibility with IoT-based infrastructures and adaptability to smart production lines [15]. One important trend involves the AI-driven design of task-specific robot morphologies aiming to enhance adaptability further in such tasks as door-opening and object-handling. Although considerable development has simmered, real-time performance, generalization of learning-based models, and energy efficiency remain areas of concern [16]. Future works should focus on robust perception, efficient planning, and safe human-robot interaction to spur its widespread acceptance.

3. System Overview of the KUKA youBot Mobile Manipulator

In this project, we focus on the analysis and simulation of the **KUKA youBot**, a mobile manipulator manufactured by KUKA Laboratories. KUKA has designed this flexible, modular facility for research, and teaching, and for prototype devices. The KUKA youBot integrates two tightly coupled yet functionally distinct subsystems:

(1) a 5-DOF robotic manipulator that is serially structured, and

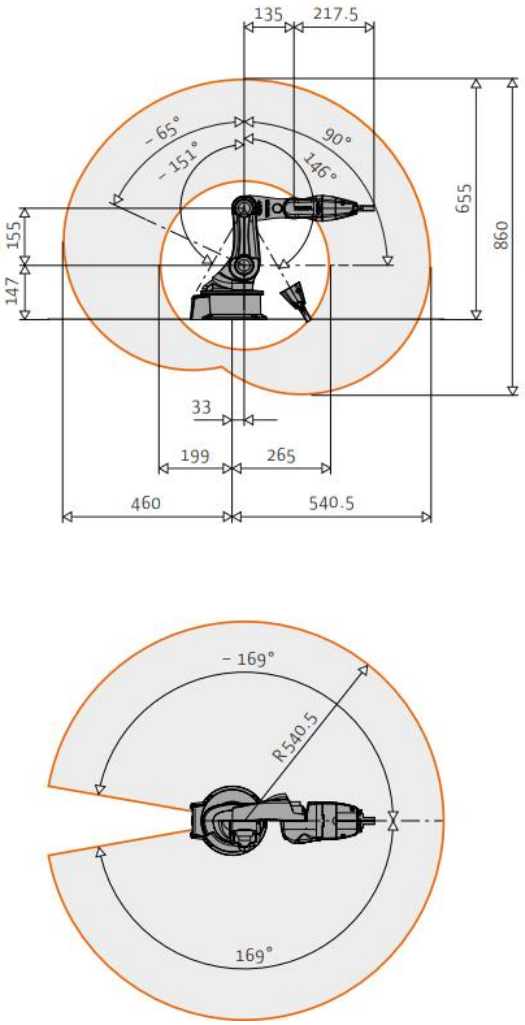
(2) a holonomic mobile platform that is actuated by omnidirectional wheels.

In its basic configuration, the KUKA youBot consists of an omnidirectional mobile platform on which a 5-axis robot arm with a gripper is installed. Importantly, the **arm, gripper, and mobile base can also be used individually**, offering flexibility in experimentation, system integration, and software development. This modularity is one of the platform's key strengths, making it well-suited for isolated or integrated subsystem analysis [17].

Figure 2: The entire image of the KUKA youBot shows both the mobile base and the 5-DOF arm with a gripper [17].

The youBot arm is a five-joint serial manipulator constructed using a magnesium casting structure, which guarantees mechanical rigidity at a lightweight construction. It provides a vertical reach of 655 millimeters, accommodates a workspace of 0.513 m^3, and supports a payload capacity of 0.5 kilograms. Two independent stepper motors are responsible to actuate the gripper, which offers a stroke of 20 millimeters and a range of 70 millimeters. The angular limits of each joint range from ±65° to ±169°; and each joint is able to rotate up to 90°/s. High positioning accuracy is ensured through a 0.1 mm repeatability rating, while control communication is handled via 1 ms EtherCAT cycles, allowing for real-time operation and synchronization [17].

Figure 3: The youBot arm [17].



youBot arm

Serial kinematics	5 axes
Height	655 mm
Work envelope	0.513 m³
Weight	5.8 kg
Payload	0.5 kg
Structure	Magnesium Cast
Positioning repeatability	0.1 mm
Communication	EtherCAT: 1 ms cycle
Voltage connection	24 V DC
Drive train power limitable to	80 W

Axis data	Range	Speed
Axis 1 (A1)	+/- 169°	90°/s
Axis 2 (A2)	+ 90°/- 65°	90°/s
Axis 3 (A3)	+ 146°/- 151°	90°/s
Axis 4 (A4)	+/- 102°	90°/s
Axis 5 (A5)	+/- 167°	90°/s

Gripper	Detachable, 2 fingers
Gripper stroke	20 mm
Gripper range	70 mm
Motors	2 independent stepper motors

Figure 4: Detailed schematic of the KUKA youBot arm showing joint structure, angles, axis data and gripper dimensions [17].

The youBot mobile platform works together with the arm by using a holonomic omnidirectional drive system based on four KUKA omni wheels. The system allows for free movement along the x-, y-, and θ-axes in a smooth and independent way, which is a great quality in trying to manage confined or cluttered environments. Supporting an onboard payload of 20 kg, the mobile platform has a length of 580 mm, a width of 376 mm, and a total height of 140 mm. The onboard PC is embedded with Intel® Atom dual-core processor, 2 GB RAM, and 32 GB SSD for local computational power in an ideal setting for autonomous operation and offers runtime for up to 90 minutes on charge using maintenance-free lead-acid batteries (24V, 5Ah) [17].

Figure 5: The youBot mobile platform [17].

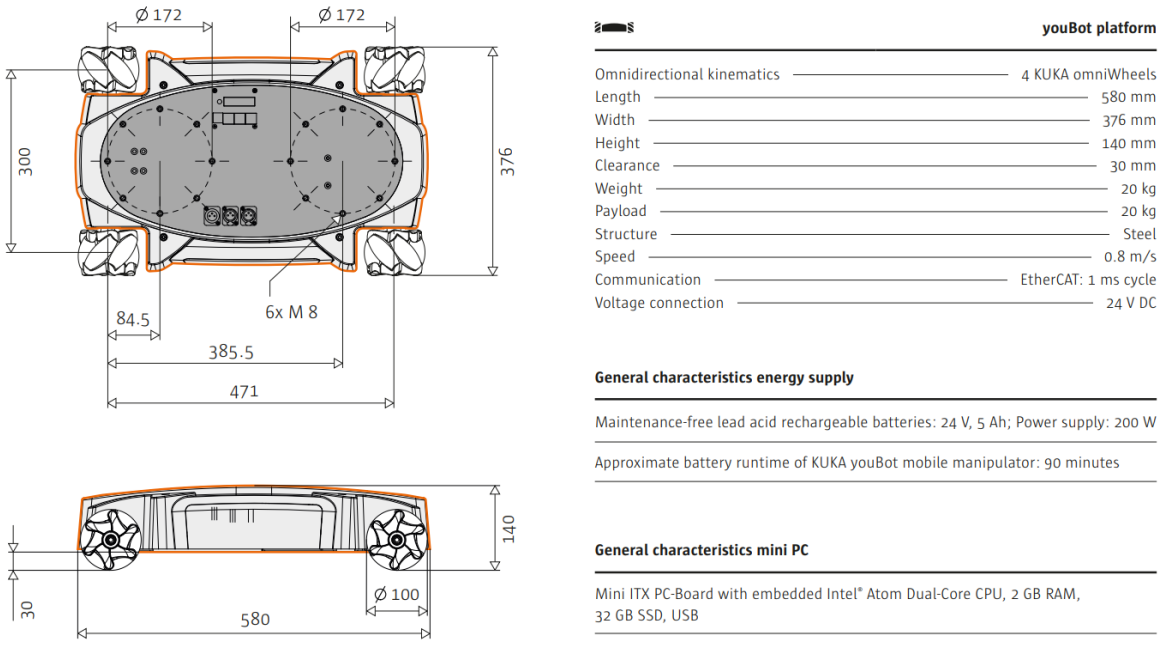


Figure 6: Detailed schematic of the KUKA youBot platform showing wheel configuration, dimensions and general characteristics [17].

With modularity and open-source compatibility, youBot is an adaptable platform that can be for all kinds of research using motion planning, robot control, human-robot interaction, and cooperative swarm robotics. It is commonly used to benchmark and evaluate such algorithms with respect to trajectory tracking, real-time control, and manipulation under constraints [17].

Given its modular design, the KUKA youBot allows for the independent as well as integrated analysis of its arm and mobile base subsystems.
In the following sections, we will present a structured kinematic analysis in three stages:

- First, we examine the robotic arm independently, focusing on forward and inverse kinematics, differential motion using Jacobians, and singularity analysis.
- Second, we analyze the mobile base, modeled as a 2D holonomic drive system, deriving its kinematic equations for planar motion including translation and rotation.
- Finally, we integrate both subsystems into a unified kinematic model, enabling coordinated mobile manipulation and trajectory planning.

3.1. Degrees of Freedom and Kinematic Redundancy

The youBot is a 5-revolute joint manipulator on a 3-DOF holonomic base (x-y translation and yaw rotation), making an overall 8-DOF system—three short of the six necessary to achieve arbitrary end-effector positions in 3D space. This structural redundancy is not merely passive; it actively enhances functional flexibility, stability, and reachability. The fifth joint dissociates base posture from wrist orientation so that the robot can enjoy consistent tool alignment in fixed base conditions. Meanwhile, additional degrees of freedom enable whole-body motion planners and inverse kinematics solvers to shape arm posture dynamically, which has damping on kinematic stresses and avoids impossible or unsafe posture for lower-DOF robots.

This type of configuration allows the robot to perform advanced operations—angled insertions, side grasps, and overhead operations—without the need to micro-reposition the chassis. The system's null-space flexibility allows it to impose joint limits, reduce manipulability, and avoid sudden decelerations or oscillating joint velocities while the base is in motion. This makes the platform extremely valuable in crowded or tight spaces.

. [18]

Configuration	**Total DOF**	**Workspace Coverage**	**Orientation Freedom**	**Singularity Handling**	**Operational Cost**
3 DOF arm + 3 DOF base	6	Good positional reach but limited orientation freedom due to base constraints	Limited—requires constant base repositioning for orientation control	Vulnerable to singularities, especially in cluttered environments	Low hardware cost, but high control effort due to frequent base adjustments
4 DOF arm + 3 DOF base	7	Better positional and orientation reach compared to 6-DOF, but still requires base movement for full poses	Partial orientation freedom, still requires base movement for certain orientations	Some null-space margin, but still prone to singularities	Balanced between cost and performance, moderate complexity

5 DOF arm + 3 DOF base	8	Full 6D pose control with minimal base movement, highly flexible	Good—can achieve independent yaw, pitch, and roll control, but still constrained by base DOF	Strong redundancy, better ability to avoid singularities	Higher capability with moderate cost, strong control flexibility

Table1 : Comparison of Robotic Configurations Based on DOF and Performance Characteristics

Task Scenario	**Role of Extra DOF**	**Result**
Diagonal shelf loading	Wrist orientation without chassis shift	Faster task cycles
Under-table wiring	Tool tilts upward while base stays low	Safe access in tight spaces
Overhead bolting	Elbow folds to maintain vertical torque	Reduced mechanical stress
Grasping inside deep bins	Base stays fixed, wrist adjusts angle	Minimal path planning effort
Continuous picking on conveyors	Orientation is decoupled from base movement	Smooth operation at speed

Table 2: Operational Benefits of Kinematic Redundancy in Complex Task Scenarios

4. Kinematic Analysis

Kinematics is the study of motion without considering the forces that lead to the movements. In robotics, kinematic analysis involves understanding how the positions, orientations, velocities, and accelerations of different components of a robot relative to the configuration of its joint or actuator are analyzed. This is a fundamental aspect concerning motion planning, trajectory generation as well as control in manipulators which require accuracy and coordination.

The kinematic model provides important information for the mobile manipulator, like the KUKA youBot, for precise positioning of the end effector in the manipulation operations. Since the robot has both a serial robotic arm and a

holonomic mobile base, a systematic and modular approach is necessary for understanding and controlling its overall system.

This section provides a comprehensive kinematic study of the KUKA youBot system in three stages:

- ***Section 4.1*** focuses on a robotic arm where it first introduces the coordinate frames and Denavit-Hartenberg (DH) parameters and then proceeds into the derivation of forward and inverse kinematics, differential kinematics through Jacobian matrices, and singularity analysis.
- ***Section 4.2*** focuses on a mobile base, and how it considers the 2D holonomic platform. It presents its velocity model, motion equations, and considerations for navigation and path tracking.
- ***Section 4.3*** integrates both subsystems into a unified kinematic model that will serve as the basis for coordinated manipulation and base motion, particularly in tasks such as pick-and-place in static environments.

This structured kinematic framework serves as the mathematical and conceptual basis for the simulation, control, and performance evaluation of the mobile manipulator in the later stages of the project.

4.1. Kinematics of the Robotic Arm

The kinematic analysis begins with the analysis of the 5-DOF serial robotic arm of the KUKA youBot. These subsections include setting up coordinate frames, forward and inverse kinematics solutions, differential motion analysis, and singularity detection. All these processes develop based on the DH modeling and become the basis of forthcoming control and simulation efforts.

4.1.1. Coordinate Frames and DH Parameters

In a kinematic model of a serial robotic manipulator based on the Denavit-Hartenberg (DH) convention, coordinate frames are assigned. This systematic procedure defines the spatial transformations between links to set the base for forward and inverse kinematics, differential analyses, and motion control. The KUKA youBot arm consists of five revolute joints that form a 5-DOF serial manipulator. The modeling process using the DH convention involves three fundamental steps:

Step 1 - Assigning DH Coordinate Frames

For every joint $\boldsymbol{i}$, a local coordinate frame $\boldsymbol{i}$ is attached:

- The $\boldsymbol{z_i}$ axis is aligned with the axis of rotation of joint $\boldsymbol{i+1}$.
- The $\boldsymbol{x_i}$ axis lies along the common normal between the axes $\boldsymbol{z_{i-1}}$ and $\boldsymbol{z_i}$.

- The $\boldsymbol{y_i}$ axis is assigned via the right-hand rule.
- The origin of frame $\boldsymbol{i}$ is located at the intersection of the $\boldsymbol{z_i}$ and $\boldsymbol{x_i}$ axes.

This systematic frame assignment ensures consistency and facilitates the extraction of DH parameters [19].

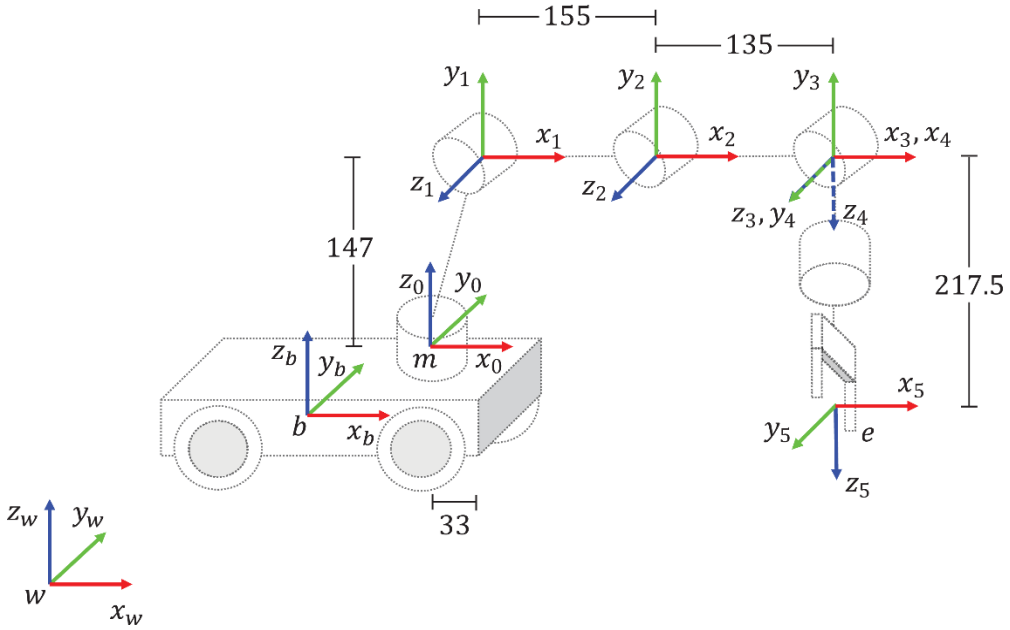


Figure 7: Denavit–Hartenberg Coordinate Frame Assignment for the KUKA youBot Arm [19].

Step 2 - Identifying the DH Parameters

After assigning coordinate frames, the following four DH parameters are extracted for each joint:

- $\boldsymbol{a_i}$: The distance between $\boldsymbol{z_{i-1}}$ *and* $\boldsymbol{z_i}$ axes along the $\boldsymbol{x_i}$ axis.
- $\boldsymbol{\alpha_i}$: The angle between $\boldsymbol{z_{i-1}}$ *and* $\boldsymbol{z_i}$ axes about the $\boldsymbol{x_i}$ axis.
- $\boldsymbol{d_i}$: The distance between $\boldsymbol{x_{i-1}}$ *and* $\boldsymbol{x_i}$ axes along the $\boldsymbol{z_{i-1}}$ axis.
- $\boldsymbol{\theta_i}$**:** The angle between $\boldsymbol{x_{i-1}}$ *and* $\boldsymbol{x_i}$ axes about the $\boldsymbol{z_{i-1}}$ axis.

Step 3 - Computing the DH Transformation Matrix

Each joint is associated with a homogeneous transformation matrix $\boldsymbol{A_i}$, which transforms the pose from frame $\boldsymbol{i-1}$ to frame $\boldsymbol{i}$. These matrices will be used in the next section to compute the overall forward kinematics of the manipulator.

joint i	a_i (mm)	α_i (rad)	d_i (mm)	θ_i (rad)
1	$l_1 = 33$	pi/2	$l_2 = 147$	θ_1
2	$l_3 = 155$	0	0	θ_2
3	$l_4 = 135$	0	0	θ_3
4	0	pi/2	0	θ_4
5	0	0	$l_5 = 217.5$	θ_5

Table3 : Numerical DH Parameters of the KUKA youBot Arm [17] - [20]

The local coordinate transformations between successive links of a robotic arm are defined by the individual transformation matrices $\boldsymbol{A_i}$, which in turn are associated with the joints. While this section was primarily devoted to defining the geometric model of the manipulator according to the DH convention, the next step will be to concatenate these matrices to obtain the full forward kinematic chain. The pose of the end-effector concerning the base frame can then be determined, which is of great importance in motion planning and control.

4.1.2. Forward Kinematics

In robotic manipulators, forward kinematics is the computation of the pose (position and orientation) of the end-effector with respect to the base frame given a set of joint parameters. For serial-link robots such as the KUKA youBot arm, this is usually accomplished by a sequence of chained spatial transformations relating each joint frame to the next.

The main character of this algorithm is a homogeneous transformation matrix, which is a 4 × 4 matrix that describes both rotation and translation between coordinate frames. Its general expression:

$$\boldsymbol{T_{i-1}^{i}} = \begin{bmatrix} \boldsymbol{R_{i-1}^{i}} & \boldsymbol{P_{i-1}^{i}} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \qquad \textbf{(4.1)}$$

Where $\boldsymbol{R_{i-1}^{i}} \in \boldsymbol{R^{3\times3}}$ is the rotation matrix from $\boldsymbol{i-1}$ to frame $\boldsymbol{i}$, and $\boldsymbol{P_{i-1}^{i}} \in \boldsymbol{R^{3\times1}}$ is the position vector of the origin of frame $\boldsymbol{i}$ with respect to frame $\boldsymbol{i-1}$. This representation is fundamental to modern robotic modeling and simulation techniques [21].

The whole transformation from the base of KUKA youBot arm to the end-effector is then:

$$\boldsymbol{T_0^5 = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5} \qquad \textbf{(4.2)}$$

Each matrix $\boldsymbol{T_{i-1,i}}$ is computed using Denavit-Hartenberg (DH) conventions, which encodes the geometry of a serial manipulator using four parameters:

- $\boldsymbol{a_i}$: link length.
- $\boldsymbol{\alpha_i}$: twist angle.
- $\boldsymbol{d_i}$: link offset.
- $\boldsymbol{\theta_i}$**:** joint angle.

The homogeneous transformation between adjacent frames is defined as a sequence of four elementary rigid-body operations:

$$\boldsymbol{T_{i-1,i} = Rot_z(\theta_i)\ Trans_z(d_i)\ Trans_x(a_i)\ Rot_x(\alpha_i)} \qquad \textbf{(4.3)}$$

$$\boldsymbol{T_{i-1,i}} = \begin{bmatrix} \boldsymbol{c_{\theta_i}} & \boldsymbol{-s_{\theta_i}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{s_{\theta_i}} & \boldsymbol{c_{\theta_i}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{d_i} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{a_i} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_{\alpha_i}} & \boldsymbol{-s_{\alpha_i}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{s_{\alpha_i}} & \boldsymbol{c_{\alpha_i}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

$$\boldsymbol{T_{i-1,i}} = \begin{bmatrix} \boldsymbol{c_{\theta_i}} & \boldsymbol{-s_{\theta_i} c_{\alpha_i}} & \boldsymbol{s_{\theta_i} s_{\alpha_i}} & \boldsymbol{a_i c_{\theta_i}} \\ \boldsymbol{s_{\theta_i}} & \boldsymbol{c_{\theta_i} c_{\alpha_i}} & \boldsymbol{-c_{\theta_i} s_{\alpha_i}} & \boldsymbol{a_i s_{\theta_i}} \\ \boldsymbol{0} & \boldsymbol{s_{\alpha_i}} & \boldsymbol{c_{\alpha_i}} & \boldsymbol{d_i} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

This ordered composition corresponds to:

1. a rotation about the z-axis of frame $\boldsymbol{i-1}$.
2. a translation along that same z-axis.
3. a translation along the x-axis of frame $\boldsymbol{i}$.
4. and a rotation about the x-axis of frame $\boldsymbol{i}$ [18], [21].

Using this standard form guarantees consistency in the modeling of robot links, and enables accurate derivation of the overall end-effector pose as a function of joint variables.

Once the complete form of the forward kinematics transformation matrix **T05** is computed, it contains the global pose of the end effector with respect to the base frame. This homogeneous matrix contains both the spatial position and orientation of the robot's tool center point (TCP) and is given as:

$$T_0^5 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.4)$$

Here,

- the upper-left 3×3submatrix R (elements r_{ij}) represents the rotation matrix, which defines the orientation of the end-effector with respect to the base frame.
- The last column vector $[xyz]^T$ represents the position of the end-effector (its origin) in the base frame **{b}**.
- The last row makes this a homogeneous matrix, the one that can perform multiplication, subsequently transforming points in 3D space.

Since the reasons for which this matrix is so important in robotics applications is as follows:

- ***Position Tracking***: The components x, y, z are monitored during motion planning and path following to determine the precise position of the end-effector [18].
- ***Orientation Control***: The rotation matrix R is used to keep a gripper or tool in an appropriate orientation since this is especially important in manipulation tasks like pick-and-place or insertion [18].
- ***Trajectory Planning***: The entire pose matrix serves as reference point for calculating interpolation trajectories, either in Cartesian or in joint space [18].

We have also successfully implemented the motion simulation of the KUKA youBot robotic arm in MATLAB. The simulation serves to dynamically visualize the evolution of the manipulator through space with changing joint angles over time. It serves as validation for the Denavit-Hartenberg-based kinematics model and an intuitive understanding of the end-effector trajectory.
The motion simulation demonstrates the real-time configuration of the robotic arm by:

- Interpolating between an initial and final joint configuration;
- Computing the corresponding transformation matrices $T_{05}(t)$ at each time step;
- Plotting the resulting end-effector pose using MATLAB's ***plot3*** and ***animatedline***.

The complete form of the forward kinematics transformation matrix **T05** in a symbolic form becomes:

$$T_0^5 = \begin{bmatrix} s_1 s_5 + c_{234} c_1 c_5 & c_5 s_1 - c_{234} c_1 s_5 & s_{234} c_1 & c_1(l_1 + l_4 c_{23} + l_3 c_2) + l_5 s_{234} c_1 \\ c_{234} c_5 s_1 - c_1 s_5 & -c_1 c_5 - c_{234} s_1 s_5 & s_{234} s_1 & s_1(l_1 + l_4 c_{23} + l_3 c_2) + l_5 s_{234} s_1 \\ s_{234} c_5 & -s_{234} s_5 & -c_{234} & l_2 + l_4 s_{23} + l_3 s_2 - l_5 c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To transition from the symbolic representation of the forward kinematics to a numerical evaluation, the angle values $(pi/2, -0.5, 1.0, -1.2, pi/2)$ are substituted into the derived transformation matrices. These angles are defined as ***theta_final*** in radians and are used to compute the corresponding end-effector poses.

$$T_0^1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 33 \\ 0 & 1 & 0 & 147 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_1^2 = \begin{bmatrix} 0.88 & 0.48 & 0 & 136.03 \\ -0.48 & 0.88 & 0 & -74.31 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} 0.54 & -0.84 & 0 & 72.94 \\ 0.84 & 0.54 & 0 & 113.60 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_3^4 = \begin{bmatrix} 0.36 & 0 & -0.93 & 0 \\ -0.93 & 0 & -0.36 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^5 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 217.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And using this equation $T_0^5 = T_{01} T_{12} T_{23} T_{34} T_{45}$ To evaluate the whole Transformation Matrix:

$$T_0^5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -0.76 & -0.64 & 147.38 \\ 0 & 0.64 & -0.76 & -28.94 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

4.1.3. Inverse Kinematics

When analytical solutions are difficult, unavailable, or need to be refined, numerical inverse kinematics techniques are used. Usually employing root-finding or optimization algorithms, these iterative methods resolve the nonlinear equations connecting joint variables to the end-effector pose.

The Newton–Raphson method is a popular and efficient technique that reduces the pose error by linearizing the forward kinematics at an initial guess and iteratively updating joint variables. Formally, the objective is to determine θsuch that: $g(\theta) = x_d - f(\theta) = 0$ for a robot with joint variables θ and forward kinematics $x = f(\theta)$.

The method updates from an initial guess, θ_0:

$$\theta_{k+1} = \theta_k + J^{\dagger}(\theta_k)(x_d - f(\theta_k))$$

where the pseudoinverse of the Jacobian J evaluated at θ_K is $J^{\dagger}$.

Until the error norm falls below a predetermined threshold, this iterative process is repeated. By using the pseudoinverse to manage redundancy and singularities, the numerical IK approach can also apply extra optimization criteria to choose the best solution when there are several.

Numerical IK enhances the analytical solution for the KUKA YouBot arm by:

Providing solutions for poses outside the analytical model's reach or when the robot's geometry is approximated, refining initial analytical solutions for higher precision, Handling complex constraints and avoiding singularities during motion.

Convergence requires an appropriate initial guess, which is frequently the solution from the previous timestep or a home configuration [18].

To validate the numerical inverse kinematics algorithm and analyze the robot's joint behavior during motion, we define a circular path for the KUKA YouBot end-effector in the **Y–Z plane**. This trajectory type simulates tasks such as welding, inspection, or painting, where precise and smooth motion is required along a curved surface. The circle is mathematically defined by the parametric equations:

$$y(\theta) = y_c + rcos(\theta), \qquad z(\theta) = z_c + rsin(\theta)$$

Here, y_c and z_c represent the center of the circle, r is the radius, and $\theta \in [0,2\pi]$ varies incrementally. For each value of θ, a desired end-effector pose is formulated, and the **numerical inverse kinematics** is applied using the Newton–

Raphson method to iteratively compute the corresponding joint configuration that minimizes the pose error.

Once the inverse kinematics solutions are obtained along the full trajectory, we extract and plot the **joint angle profiles** $\theta_i(t)$ for all five joints. These profiles reveal how each joint evolves throughout the motion. In addition, we compute the **joint velocity profiles** $\dot{\theta}_i(t)$ using finite differences between consecutive joint positions. These velocity plots provide insight into the smoothness and feasibility of the trajectory from a dynamic control perspective.

Both the angular position and velocity profiles are essential to verify the kinematic performance of the robot and to ensure that the motion respects actuator limits and avoids abrupt changes, which could lead to instability or wear in physical systems [18].

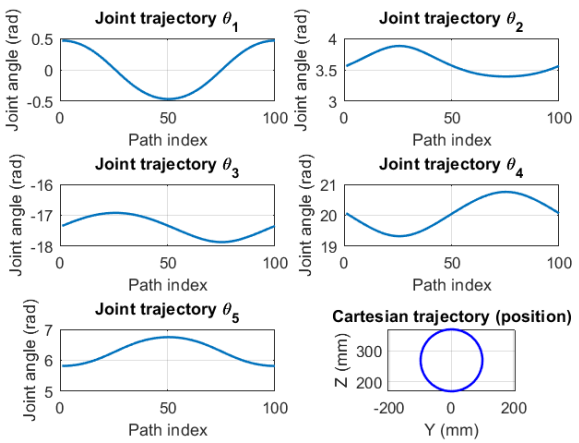


Figure10 : KUKA YouBot Joint Angles and Cartesian Path

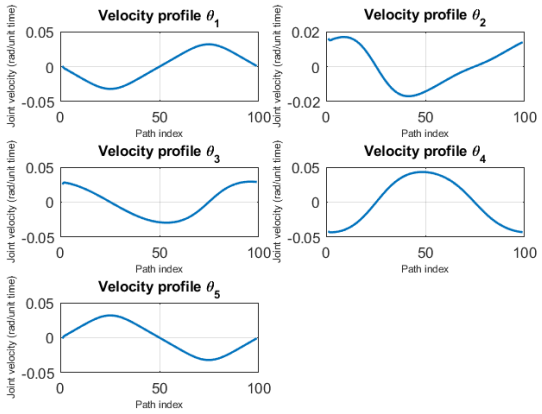


Figure9 : KUKA YouBot Joint Velocity Profiles

Figure8 : youBot experimental result

4.1.4. Differential Kinematics

It is a basic part of the analysis and control of robotic manipulators' differential kinematics for the describing relationship with the velocities of the joints of the robot as well as linear and angular velocities of its end-effector. Velocity-level analysis finds applications in path tracking, visual servo, and force control for real-time applications in unstructured or dynamic environments. The Jacobian matrix is the tool employed in differential kinematics. This is represented as $\boldsymbol{J(q)}$ and it relates the joint velocities $\dot{\boldsymbol{q}}$ to the spatial velocity $\boldsymbol{v}$ of the end-effector through the linear relationship.

$$\boldsymbol{v = J(q) \cdot \dot{q}} \qquad \textbf{(4.49)}$$

Here, $\boldsymbol{v \in R^6}$ is a 6D representation of the velocity vector with linear $\boldsymbol{(v_x, v_y, v_z)}$ and angular $\boldsymbol{(\omega_x, \omega_y, \omega_z)}$ velocities, while $\dot{\boldsymbol{q}} \boldsymbol{\in R^n}$ is the joint velocity vector for the manipulator with **n** degrees of freedom.

There are three main purposes of the Jacobian:

1. ***Velocity Mapping***: For transformation of target Cartesian velocities of the end-effector to corresponding joint velocities, thus making it the core element for velocity control strategies [22].
2. ***Force Mapping***: The transpose $\boldsymbol{J^T(q)}$ takes Cartesian forces and torques from the end-effector back to joint torques. This is utilized in compliant control and interaction applications with the environment [23].
3. ***Singularity Analysis***: The manipulator reaches some configurations with singularities, wherein the Jacobian matrix loses ranks, thus not allowing certain Cartesian velocities to be reached, affecting dexterity and control. Such singularities should be identified and avoided in motion planning [22].
4. ***Redundant Manipulation***: Where there are more than a sufficient number of joints on a system for movements (redundant manipulators), the Jacobian will help with the redundancy resolution. Additional optimization criteria, such as energy efficiency or obstacle avoidance, can also be implemented via null-space projections [24].

The Jacobian is essential to the smooth and accurate trajectory control of the 5-DOF robotic arm in the KUKA youBot. This implies that when the arm is put into a dynamic environment, it can be controlled effectively. Furthermore, this will allow for upcoming tasks such as inverse kinematics, compliance control, and manipulation in uncertain environments.

4.1.4.1. Derivation of the Jacobian Matrix

To find the Jacobian matrix of the KUKA youBot arm, it is imperative to analyze its structure and understand how the individual joints influence the position and orientation of the end-effector. The Jacobian matrix $\boldsymbol{J(q)}$ provides a linear mapping between joint velocities $\boldsymbol{v \epsilon R^6}$ and the spatial velocity $\boldsymbol{\dot{q} \epsilon R^5}$ (linear and angular) of the end-effector:

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} = \boldsymbol{J(q)} \cdot \boldsymbol{\dot{q}}$$

This Jacobian can be developed by using the geometric techniques described in the Denavit-Hartenberg (DH) convention, which we have previously established in forward kinematics. Since this arm contains five revolute joints, the Jacobian matrix will be **6×5** matrix.

For a manipulator with revolute joints, each column $\boldsymbol{J_i}$ of the Jacobian is calculated as:

$$\boldsymbol{J_i} = \begin{bmatrix} \boldsymbol{Z_{i-1} \times (O_n - O_{i-1})} \\ \boldsymbol{Z_{i-1}} \end{bmatrix} \quad \textbf{(4.50)}$$

Where:

- $\boldsymbol{O_{i-1}}$ is the origin of frame $\boldsymbol{i-1}$ expressed in the base frame.
- $\boldsymbol{Z_{i-1}}$ is the unit vector along the axis of joint $\boldsymbol{i}$, also expressed in the base frame.
- $\boldsymbol{O_n}$ is the origin of the end-effector frame (frame 5 in our case).

The Jacobian matrix is then constructed as:

$$\boldsymbol{J(q)} = [\boldsymbol{J_1\, J_2\, J_3 J_4\, J_5}] \quad \textbf{(4.51)}$$

This method requires computing all transformation matrices $\boldsymbol{T_0^1, T_0^2, \ldots., T_0^5}$ using the symbolic DH parameters, which have already been defined for the KUKA youBot.

we consider a specific movement of the KUKA youBot arm from an initial to a final joint configuration.

Initial state: $\theta = [\boldsymbol{0, 0, 0, 0, 0}]$ to ***Final state:*** $\boldsymbol{\theta} = [\boldsymbol{pi/2, -0.5, 1, -1.2, pi/2}]$

	q1_dot	q2_dot	q3_dot	q4_dot	q5_dot
vx	-0.15	0.00	0.00	0.00	0.00
vy	0.00	0.18	0.10	0.17	0.00
vz	0.00	0.11	-0.02	-0.14	0.00
wx	0.00	1.00	1.00	1.00	0.00
wy	0.00	-0.00	-0.00	-0.00	-0.64
wz	1.00	0.00	0.00	0.00	-0.76

Figure 11: Jacobian Matrix for a Specific Joint Configuration of the KUKA youBot [by the author].

4.1.4.2. Interpretation and Physical Meaning of the Jacobian Matrix

Even aside from the mathematical definition, the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$ is an instantaneous motion capability of the manipulator at any certain configuration. The Jacobian can otherwise be defined as a mapping by which small proper joint movements translate into differential motions of the end effector in Cartesian space.

Importantly, the Jacobian provides insights into:

- ***Dexterity and directionality:*** It informs us of which Cartesian directions will be reached for a configuration and those that will not.
- ***Sensitivity:*** While small joint velocity can create a large Cartesian velocity because Jacobians obtain larger values in some directions, this is useful in precision surgical or micro-manipulation tasks [18].
- ***Force transmission:*** Because forces applied at the end-effector reflect back into joint torques via the transpose Jacobian, the structure of $\boldsymbol{J}(\boldsymbol{q})$ gives much headway in mechanical advantage-disadvantage analysis [18].

A manipulator loses one or more degrees of freedom at such singular configurations, meaning that the Jacobian becomes rank-deficient. This will affect both the possibility of motion and the amplification of applied forces. Near singularities, a highly sensitive end-effector may lead to very dangerous situations in applications involving high precision, for instance, robotic surgery or microscale assembly [10]. Using the Jacobian, one obtains the possibility of null-space motion for redundant manipulators, which means that the manipulator produces joint movements without changing the posture of the end-effector. This

characteristic can be exploited to optimize secondary goals like obstacle avoidance or energy minimization by projecting commands in the null space of the Jacobian [18].
This interpretation in the physical sense gives more to the Jacobian as an entity as more than just a number matrix-it gives it more of a window into the robot's mechanical behavior and control capability when it is moving.

4.1.4.3. Applications in Velocity Control

The velocity control for a robotic system is basically the smooth and accurate movement of the end-effector of a manipulator. In this context, the Jacobian matrix plays a pivotal role in establishing a linear mapping between joint space velocities and Cartesian space velocities. This gives access to joint velocities $\dot{\boldsymbol{q}}$, which are necessary in order to realize an end-effector velocity requirement $\boldsymbol{v}$ in space.

The relationship is defined as:

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}}$$

To control the end-effector velocity, we solve this equation for $\dot{\boldsymbol{q}}$:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\boldsymbol{q}) \cdot \boldsymbol{v} \qquad \textbf{(4.52)}$$

Where $\boldsymbol{J}^{+}$ is the **Moore-Penrose pseudoinverse** of the Jacobian, particularly useful when the robot has redundant degrees of freedom or when the Jacobian is not square.

Velocity Control Loop

This is how it goes in a realistic control loop:

- A desired velocity $\boldsymbol{v}_{\textbf{desired}}$ according to the plans of the high-level planner.
- Evaluating the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ for each time.
- Calculating joint velocities $\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}(\boldsymbol{q}) \cdot \boldsymbol{v}_{\textbf{desired}}$
- These joint velocities are directed to the actuators of the robot.

4.1.4.4. Jacobian Singularity and Rank Deficiency

The Jacobian matrix plays a pivotal role in determining a robotic manipulator's ability to move and exert forces in its workspace. However, certain configurations, called singularities, lead to a loss of rank in the Jacobian and the absence of certain degrees of freedom (***DOF***) in the motion of the end effector. Those poses exhibit very extreme effects in terms of controllability and manipulability with respect to the robot.

The Jacobian matrix $\boldsymbol{J(q)}$ will become rank-deficient indicating that its rank is lesser than that of the number of degrees of freedom of the task space. This means that any set of joint speeds will not be able to generate one or more Cartesian component speeds.

Mathematically, this assertion translates into the following: as the determinant of $\boldsymbol{JJ^T}$ (or its reduced variant for non-square matrices) approaches zero:

$$\boldsymbol{rank\big(J(q)\big) < min(m,n)} \quad \boldsymbol{(4.53)}$$

Where:

- $\boldsymbol{m}$ is the task space dimension.
- $\boldsymbol{n}$ is the number of joints.

When this happens, the manipulator may experience:

- ***Loss of mobility*** in one or more Cartesian directions.
- ***Infinite joint velocities*** required to maintain certain end-effector motions.
- ***Poor force transmission*** characteristics due to loss of control authority.

These conditions often arise when:

- Joint axes become aligned (e.g., in a planar configuration).
- The end-effector reaches the boundary of the workspace.

Getting close to or at singularities will bring about some numerical instability in the inverse kinematics and control algorithm, for example, when the Moore-Penrose pseudoinverse $\boldsymbol{J^+}$ becomes ill-conditioned. The velocity commands will become unbounded joint velocities. Force control strategies fail due to insufficient torque generation capabilities [18].

4.1.5. Orientation Representation: Euler Angles and RPY

In robotic systems, especially for articulated manipulators such as the KUKA youBot, representing the orientation of the end-effector is as crucial as determining its position. The position in Cartesian space can thus be defined with three linear coordinates x, y, and z, but orientation in three-dimensional space is determined more complicatedly because of its various freedoms of rotation. Different methods for representing orientation exist, including rotation matrices and quaternions to axis-angle and angle-based representations. Among all these methods, Euler Angles and Roll-Pitch-Yaw (***RPY***) representation are the most commonly applied to industrial and academic robots due to the ease of interpretation and simplicity of the mathematical formulation. Both Euler and RPY methods represent orientation in terms of three serial runs of simple rotations about well-defined axes. These sequential rotations transform a body from an initial reference frame to its current orientation. Even though they look quite attractive, these representations are not free from complications, like the risk of encountering singularities (e.g., gimbal lock) and sensitivity to rotation order. This section introduces Vue representation alongside the Euler Angles and Roll-Pitch-Yaw (RPY) representation, discusses the mathematical ground of these methods, their use in robot kinematics, and also considers their limitations in relation to singularity analysis.

4.1.5.1. Euler Angles

The orientation of a rigid body in three-dimensional space can be described classically and with a widely used method called Euler angles. In robotics, they are often used to specify the orientation of the end-effector with regard to a fixed reference frame. An orientation using Euler angles is specified through a combination of three elemental rotations around the selected axes.

The ZXZ Euler convention describes orientation through:

1. A rotation about the Z-axis ($\boldsymbol{\omega}$),
2. Followed by a rotation about the new X-axis ($\boldsymbol{\theta}$),
3. And finally, a rotation about the new Z-axis ($\boldsymbol{\varphi}$).

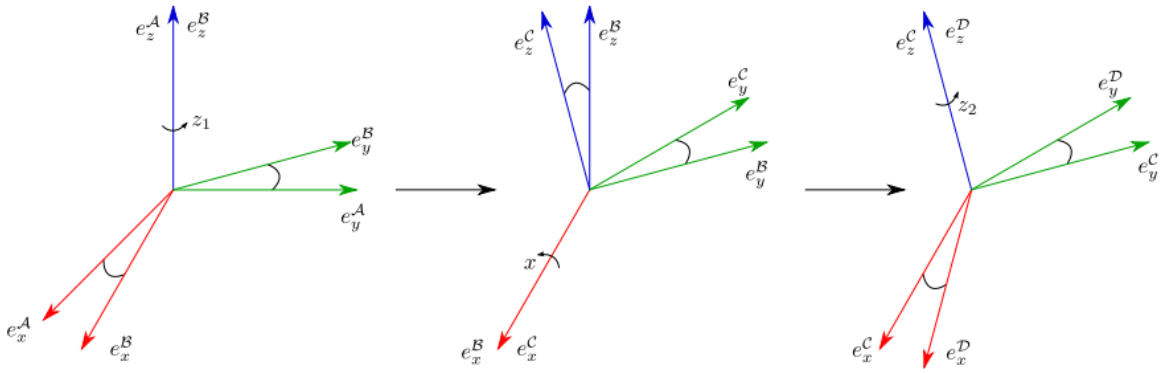


Figure 12: ZXZ Euler Angles as three successive rotations around z, x, and z axes [29].

The composite rotation matrix $\boldsymbol{R}$ for $\boldsymbol{ZXZ}$ Euler angles $(\boldsymbol{\omega}, \boldsymbol{\theta}, \boldsymbol{\varphi})$ is expressed as:

$$\boldsymbol{R} = \boldsymbol{R_z}(\boldsymbol{\omega}) \cdot \boldsymbol{R_{x_\omega}}(\boldsymbol{\theta}) \cdot \boldsymbol{R_{z_\theta}}(\boldsymbol{\varphi})$$

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{c_\omega} & \boldsymbol{-s_\omega} & \boldsymbol{0} \\ \boldsymbol{s_\omega} & \boldsymbol{c_\omega} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} . \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_\theta} & \boldsymbol{-s_\theta} \\ \boldsymbol{0} & \boldsymbol{s_\theta} & \boldsymbol{c_\theta} \end{bmatrix} . \begin{bmatrix} \boldsymbol{c_\varphi} & \boldsymbol{-s_\varphi} & \boldsymbol{0} \\ \boldsymbol{s_\varphi} & \boldsymbol{c_\varphi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{c_\omega c_\varphi} - \boldsymbol{s_\omega c_\theta s_\varphi} & \boldsymbol{-c_\omega s_\varphi} - \boldsymbol{s_\omega c_\theta c_\varphi} & \boldsymbol{s_\omega s_\theta} \\ \boldsymbol{s_\omega c_\varphi} + \boldsymbol{c_\omega c_\theta s_\varphi} & \boldsymbol{-s_\omega s_\varphi} + \boldsymbol{c_\omega c_\theta c_\varphi} & \boldsymbol{-c_\omega s_\theta} \\ \boldsymbol{s_\theta s_\varphi} & \boldsymbol{s_\theta c_\varphi} & \boldsymbol{c_\theta} \end{bmatrix} \quad \boldsymbol{(4.54)}$$

Euler angles are compact and relatively intuitive; however, they exhibit limitations such as ***gimbal lock***, which occurs when two of the three rotation axes become aligned, resulting in a loss of one degree of freedom. This degeneracy can severely impact computations in inverse kinematics and singularity analysis, where a complete and unique representation of orientation is essential.

4.1.5.2. Roll-Pitch-Yaw (RPY)

Roll-Pitch-Yaw angle representation is a convention commonly used in robotics to understand the orientation of a motion with the vehicle or tool movement more intuitively. RPY angles define a sequence of three elemental rotations around the principal axes of the base or world frame, commonly called fixed axes, unlike Euler angles, which might make use of moving axes depending on the convention used.

In the RPY convention, orientation is expressed through:

1. A **roll** angle $\boldsymbol{\gamma}$, representing rotation about the X-axis,
2. Followed by a **pitch** angle $\boldsymbol{\beta}$, a rotation about the Y-axis,

3. And finally, a **yaw** angle $\boldsymbol{\alpha}$, a rotation about the Z-axis.

The composite rotation matrix $\boldsymbol{R}$ from these three rotations is:

$$\boldsymbol{R} = \boldsymbol{R}_{z}(\boldsymbol{\alpha}) \cdot \boldsymbol{R}_{y_{\alpha}}(\boldsymbol{\beta}) \cdot \boldsymbol{R}_{x_{\beta}}(\boldsymbol{\gamma})$$

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{c_\alpha} & \boldsymbol{-s_\alpha} & \boldsymbol{0} \\ \boldsymbol{s_\alpha} & \boldsymbol{c_\alpha} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} . \begin{bmatrix} \boldsymbol{c_\beta} & \boldsymbol{0} & \boldsymbol{s_\beta} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{-s_\beta} & \boldsymbol{0} & \boldsymbol{c_\beta} \end{bmatrix} . \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_\gamma} & \boldsymbol{-s_\gamma} \\ \boldsymbol{0} & \boldsymbol{s_\gamma} & \boldsymbol{c_\gamma} \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{c_\alpha c_\beta} & \boldsymbol{-s_\alpha c_\gamma + c_\alpha s_\beta s_\gamma} & \boldsymbol{s_\alpha s_\gamma + c_\alpha s_\beta c_\gamma} \\ \boldsymbol{s_\alpha c_\beta} & \boldsymbol{c_\alpha c_\gamma + s_\alpha s_\beta s_\gamma} & \boldsymbol{-c_\alpha s_\gamma + s_\alpha s_\beta c_\gamma} \\ \boldsymbol{-s_\beta} & \boldsymbol{c_\beta s_\gamma} & \boldsymbol{c_\beta c_\gamma} \end{bmatrix} \quad (\mathbf{4.55})$$

This corresponds to the ZYX fixed-axis (or extrinsic) rotation sequence, which is often used in serial robotics, mobile platforms, and manipulators like the KUKA youBot due to its intuitive link with human-like motion: roll (tilt side to side), pitch (nod forward/back), and yaw (turn left/right).

RPY angles offer practical benefits in simulation and visualization since they align well with common control strategies and sensor outputs (e.g., IMUs and motion capture systems). However, like Euler angles, they are not free from limitations, particularly gimbal lock, which occurs when the pitch angle approaches **±90°**, causing a loss of one degree of rotational freedom and making certain Jacobians ill-conditioned near these configurations [25].

In robotic systems such as the KUKA youBot 5-DOF mobile manipulator, accurate orientation representation plays a foundational role in ensuring precise control and motion planning. Both Euler angles and Roll-Pitch-Yaw (RPY) sequences offer compact, three-parameter descriptions of orientation in 3D space. While Euler angles are often favored in theoretical modeling due to their mathematical compactness, RPY angles are more intuitive for interpreting real-world orientations, especially in mobile platforms and aerial vehicles. However, both representations suffer from gimbal lock, where two of three rotational axes become aligned and lose one degree of freedom, severely impacting the reliability of Jacobian-based control algorithms. This becomes an important consideration when analyzing kinematic singularities, where the manipulator may lose controllability or may give infinite joint velocities to a finite motion of the end effector. An understanding of the attribute of orientation representations near singular configurations becomes mandatory for trajectory and velocity planning

and appreciably more important in the case of the KUKA youBot, whose hybrid structure (5-DOF arm mounted on a mobile base) results in arm-related and base-related singularities.

In the following section, we investigate the nature and implications of singularities in the context of the youBot's kinematic structure, analyze their effect on motion and control, and provide both mathematical and illustrative examples to demonstrate their impact.

4.1.6. Singularity Analysis

In robotic manipulators, kinematic singularities in configurations for which the Jacobian matrix loses rank so that one or more degrees of freedom in the end-effector's motion are lost. At a singularity, the manipulator either becomes unable to move in certain directions or requires infinite joint velocities to achieve a desired end-effector velocity. Therefore, control of the manipulator in such conditions becomes impossible.

Singularities are typically classified into three categories:

- ***Workspace (or boundary) singularities:*** These occur when the manipulator reaches the edge of its reachable workspace.
- ***Internal (or configuration-dependent) singularities:*** Arise due to specific joint alignments that cause loss of mobility (for example, collinear joints in serial chains).
- ***Tool singularities:*** Associated with the orientation of the end effector, especially for wrist-type configurations where the axes of rotation align.

In the context of the KUKA youBot 5-DOF arm, the joints may become collinear or aligned so that Jacobian loses row full rank. Singularities may arise for youBot in orientation singularities as well in the gimbal lock situation when one uses Euler or RPY angles to define orientation due to it having a spherical wrist configuration.

It is very pertinent to understand and determine singular configurations for the following purposes:

- To ensure safe trajectory planning.
- To guarantee the stability of control in specific velocity or force control situations.

- To facilitate the generation of smooth and feasible inverse kinematics solutions.

4.1.6.1. Mathematical Foundation of Singularities

In robotic systems, a manipulation singularity is described within certain geometric arrangements wherein the manipulator is not able to move or exert force in a certain direction. These singularities are directly related to the properties of the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$, which describes the relation between joint velocities $\dot{\boldsymbol{q}}$ and end-effector velocities $\dot{\boldsymbol{x}}$. As shown in equation below:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}} \qquad \textbf{(4.56)}$$

The manipulator attains singular configurations when the Jacobian loses its rank, that is, when its rank is lower than its maximum possible value. This indicates that there exist certain directions of motion or force at the end-effector that no longer achievable regardless of joint inputs. For example, in the case of square Jacobians equal to the degrees of freedom of the end-effector, they are singular if the determinant of $\boldsymbol{J}$ is equal to zero.

$$\boldsymbol{det(J) = 0} \qquad \textbf{(4.57)}$$

In most general cases, particularly for redundant or underactuated robots such as the KUKA youBot 5-DOF manipulator, where Jacobians are non-square, singularities are brought about by checking for rank deficiency or evaluating the singular values of the matrix. Mathematically, at singular configurations, small joint movements cause large and unpredictable movements of the end effector, certain desired velocities of the end effector become unachievable. This has consequences for trajectory tracking, force control, and inverse kinematics computation, demanding the design of singularity-robust control algorithms [18].

4.1.6.2. Singularity Conditions in KUKA youBot 5-DOF Arm

One primary condition leading to singularities in the youBot arm is the ***collinearity of joint axes***. Such alignment can cause a loss of rank in the Jacobian matrix, indicating that the manipulator has lost degrees of freedom to produce a movement or apply force in a certain direction. It is particularly critical during activities where the end-effector position or orientation should be specified accurately [26]. Additionally, ***the youBot's wrist configuration*** can lead to singularities when the wrist joints align in a manner that reduces the manipulator's dexterity. Just like

most other singularities related to the wrist, this scenario usually occurs when the end-effector approaches certain expected orientations, thus limiting the robot from encountering such desired poses, due to issues of control [27].

To prevent the occurrence of instabilities in control, one really has to understand these singular configurations when it comes to effective motion planning and control. Being cognizant of these configurations will ensure smoother operations.

4.1.6.3. Orientation-Related Singularities and Gimbal Lock

In robotic manipulators, it is common to represent orientations by Euler angles or by Roll-Pitch-Yaw (RPY) sequences because they have an intuitive interpretation when using these representations. However, it also has the negative consequence of becoming susceptible to a phenomenon called Gimbal Lock, a specific type of kinematic singularity.

Understanding Gimbal Lock

Gimbal Lock occurs when two of the three rotational axes in a three-gimbal system become aligned, effectively losing one degree of freedom. In the case of Euler angles, this occurs most commonly when the pitch angle approaches either of the two values, **±90** degrees, aligning the roll and yaw axes together and thus making some orientations unattainable. Such events can lead to erratic robot motion and velocities, which can negatively affect both the robot arm and the end effector as well as the process.

Implications for KUKA youBot 5-DOF

The KUKA youBot 5-DOF manipulator is, due to this limited degree of freedom, quite susceptible to orientation singularities in that, concerning some configurations, the robot may present indeterminate end-effector orientations, primarily in the use of Euler or RPY representations.

Mitigation Strategies

To address the challenges posed by Gimbal Lock, alternative representations such as quaternions might be advocated for orientation representation. Quaternion's representation for orientation does not have any singularity, because its representation is smooth and continuous for the rotations without Gimbal Lock [28]. Moreover, the control algorithms based on quaternion would increase the

robustness and reliability of manipulating robots, especially as applied in complex orientation movements.

4.2. Kinematics of the Mobile Base

A kinematic model of a mobile robot governs how wheel speeds map to robot velocities, while a dynamic model governs how wheel torques map to robot accelerations. In this section, we ignore the dynamics and focus on the kinematics. We also assume that the robots roll on hard, flat, horizontal ground without skidding (i.e., tanks and skid-steered vehicles are excluded). The mobile robot is assumed to have a single rigid-body chassis (not articulated like a tractortrailer) with a configuration $T_{sb} \in SE(2)$ representing a chassis-fixed frame $\{b\}$ relative to a fixed space frame $\{s\}$ in the horizontal plane. We represent T_{sb} by the three coordinates $q = [\phi \quad x \quad y]^T$. We also usually represent the velocity of the chassis as the time derivative of the coordinates, $\dot{q} = [\dot{\phi} \quad \dot{x} \quad \dot{y}]^T$. Occasionally it will be convenient to refer to the chassis' planar twist $V_b = [\omega_{bz} \quad v_{bx} \quad v_{by}]^T$ expressed in {b}, where:

$$V_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

$$\dot{q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix}$$

Figure 13: A typical wheel that rolls

Figure 14: An omniwheel.

Figure 15: A mecanum wheel.

4.2.1. Types of Wheeled Mobile Robots

Wheeled mobile robots may be classified in two major categories, omnidirectional and nonholonomic. Omnidirectional mobile robots have no equality constraints on

the chassis velocity $\dot{q} = [\dot{\phi} \quad \dot{x} \quad \dot{y}]^T$, while nonholonomic robots are subject to a single Pfaan velocity constraint A(q) ˙q = 0. For a car-like robot, this constraint prevents the car from moving directly sideways. Despite this velocity constraint, the car can reach any (ϕ, x, y) configuration in an obstacle-free plane. In other words, the velocity constraint cannot be integrated to an equivalent configuration constraint, and therefore it is a nonholonomic constraint. Whether a wheeled mobile robot is omnidirectional or nonholonomic depends in part on the type of wheels it employs (Figures 13, 14, 15). Nonholonomic mobile robots employ conventional wheels, such as you might find on your car: the wheel rotates about an axle perpendicular to the plane of the wheel at the wheel's center, and optionally it can be steered by spinning the wheel about an axis perpendicular to the ground at the contact point. The wheel rolls without sideways slip, which is the source of the nonholonomic constraint on the robot's chassis.

Omnidirectional wheeled mobile robots typically employ either omniwheels or mecanum wheels. An omniwheel is a typical wheel augmented with rollers on its outer circumference. These rollers spin freely about axes in the plane of the wheel and tangential to the wheel's outer circumference, and they allow sideways sliding while the wheel drives forward or backward without slip in that direction. Mecanum wheels are similar, except that the spin axes of the circumferential rollers are not in the plane of the wheel. The sideways sliding allowed by omniwheels and mecanum wheels ensures that there are no velocity constraints on the robot's chassis. Omniwheels and mecanum wheels are not steered, only driven forward or backward. Because of their small diameter rollers, omniwheels and mecanum wheels work best on hard, flat ground. The issues in the modeling, motion planning, and control of wheeled mobile robots depend intimately on whether the robot is omnidirectional or nonholonomic.[29].

4.2.2. Kinematic Modeling of Omnidirectional Robots

4.2.2.1. Inverse Kinematic Model

An omnidirectional mobile robot must have at least three wheels to achieve an arbitrary three-dimensional chassis velocity $\dot{q} = [\dot{\phi} \quad \dot{x} \quad \dot{y}]^T$, since each wheel has only one motor (controlling its forward–backward velocity). (Figures 17, 18) show two omnidirectional mobile robots, one with three omniwheels and one with four mecanum wheels. Also shown are the wheel motions obtained by driving the wheel motors as well as the free sliding motions allowed by the rollers. Two important questions in kinematic modeling are the following.

(a) Given a desired chassis velocity $\dot{q}$, at what speeds must the wheels be driven?
(b) Given limits on the individual wheel driving speeds, what are the limits on the chassis velocity $\dot{q}$?

To answer these questions, we need to understand the wheel kinematics illustrated in Figure (16) In a frame $\hat{x}_w - \hat{y}_w$ at the center of the wheel, the linear velocity of the center of the wheel is written $v = (v_x, v_y)$, which satisfies:

$$\begin{bmatrix} \boldsymbol{v_x} \\ \boldsymbol{v_y} \end{bmatrix} = \boldsymbol{v_{drive}} \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} + \boldsymbol{v_{slide}} \begin{bmatrix} -\mathbf{\sin \gamma} \\ \mathbf{\cos \gamma} \end{bmatrix}$$

where denotes the angle at which free "sliding" occurs (allowed by the passive rollers on the circumference of the wheel), v_{drive} is the driving speed, and v_{slide} is the sliding speed. For an omniwheel = 0 and, for a mecanum wheel, typically γ = ±45. Solving the previous Equation, we get:

$$\boldsymbol{v_{drive}} = \boldsymbol{v_x} + \boldsymbol{v_y} \tan \boldsymbol{\gamma}$$

$$\boldsymbol{v_{slide}} = \frac{\boldsymbol{v_y}}{\cos \boldsymbol{\gamma}}$$

Letting r be the radius of the wheel and u be the driving angular speed of the wheel,

$$\boldsymbol{u} = \frac{\boldsymbol{v_{drive}}}{\boldsymbol{r}} = \frac{\mathbf{1}}{\boldsymbol{r}} \left(\boldsymbol{v_x} + \boldsymbol{v_y} \tan \boldsymbol{\gamma} \right)$$

To derive the full transformation from the chassis velocity $\dot{q} = [\dot{\theta} \quad \dot{x} \quad \dot{y}]^T$ to the driving angular speed u_i for wheel i, refer to the notation illustrated in

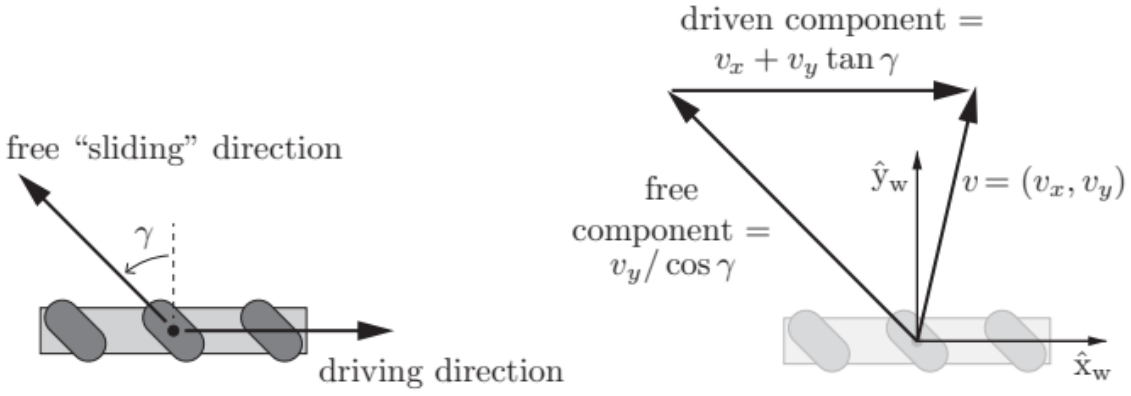


Figure 16: (Left) The driving direction and the direction in which the rollers allow the wheel to slide freely. For an omniwheel = 0 and, for a mecanum wheel, typically = ±45. (Right) The driven and free sliding speeds for the wheel velocity $v = (v_x, v_y)$ expresseed in the wheel frame $\hat{x}_w - \hat{y}_w$, where the $\hat{x}_w$-axis is aligned with the forward driving direction.

$$\boldsymbol{u_i = h_i(\phi)\dot{q}}$$

$$= \begin{bmatrix} \frac{1}{r_i} & \frac{\tan\gamma_i}{r_i} \end{bmatrix} \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ \sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} -y_i & 1 & 0 \\ x_i & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

Reading from right to left: the first transformation expresses $\dot{q}$ as V_b; the second transformation produces the linear velocity at the wheel in {b}; the third transformation expresses this linear velocity in the wheel frame $\hat{x}_w - \hat{y}_w$; and the final transformation calculates the driving angular velocity using the previous two Equations, we get $h_i(\theta)$:

$$\boldsymbol{h_i(\phi) = \frac{1}{r_i \cos\gamma_i} \begin{bmatrix} x_i \sin(\beta_i + \gamma_i) - y_i \cos(\beta_i + \gamma_i) \\ \cos(\beta_i + \gamma_i + \phi) \\ \sin(\beta_i + \gamma_i + \phi) \end{bmatrix}^T}$$

For an omnidirectional robot with $m \geq 3$ wheels, the matrix $H(\phi) \in R^{m\times 3}$ mapping a desired chassis velocity $\dot{q} \in R^3$ to the vector of wheel driving speeds

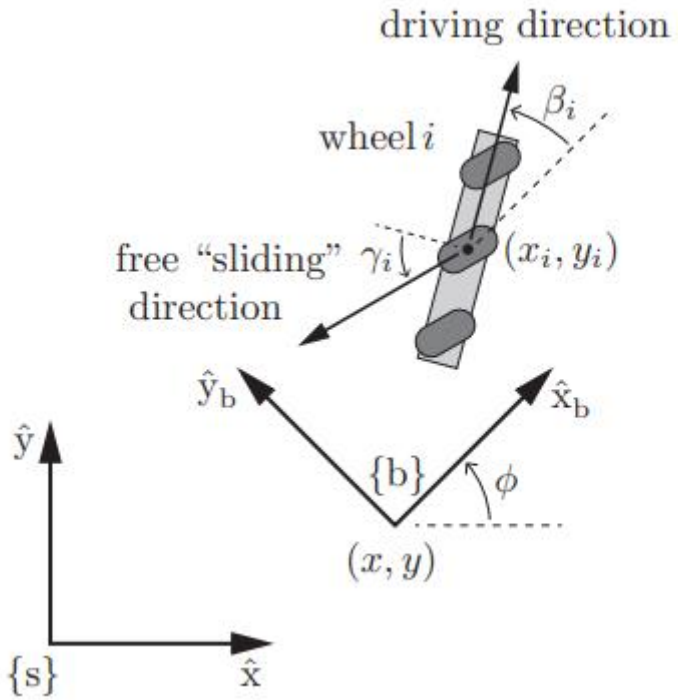


Figure 17: The fixed space frame {s}, a chassis frame {b} at (ϕ, x, y) in {s}, and wheel i at (x_i, y_i) with driving direction β_i, both expressed in {b}. The sliding direction of wheel i is defined by γ_i.

$u \in R^m$ is constructed by stacking the m rows $h_i(\phi)$:

$$\boldsymbol{u = H(\phi)\dot{q}} = \begin{bmatrix} \boldsymbol{h_1(\phi)} \\ \boldsymbol{h_2(\phi)} \\ \vdots \\ \boldsymbol{h_m(\phi)} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\phi}} \\ \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{y}} \end{bmatrix}$$

We can also express the relationship between u and the body twist V_b. This mapping does not depend on the chassis orientation:

$$\boldsymbol{u = H(0)V_b} = \begin{bmatrix} \boldsymbol{h_1(0)} \\ \boldsymbol{h_2(0)} \\ \vdots \\ \boldsymbol{h_m(0)} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega_{bz}} \\ \boldsymbol{v_{bx}} \\ \boldsymbol{v_{by}} \end{bmatrix}$$

The wheel positions and headings (β_i, x_i, y_i) in {b}, and their free sliding directions i, must be chosen so that $H(0)$ is rank 3. For example, if we constructed a mobile robot of omniwheels whose driving directions and sliding directions were all aligned, the rank of $H(0)$ would be 2, and there would be no way to controllably generate translational motion in the sliding direction.

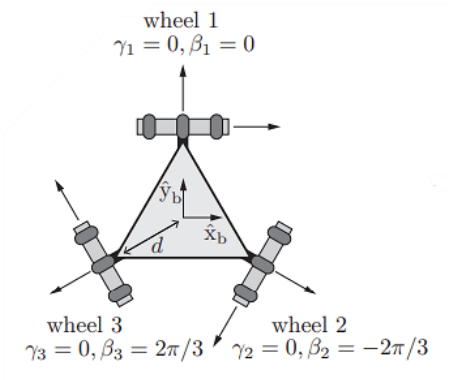


Figure 19: Kinematic models for mobile robots with three omniwheels.

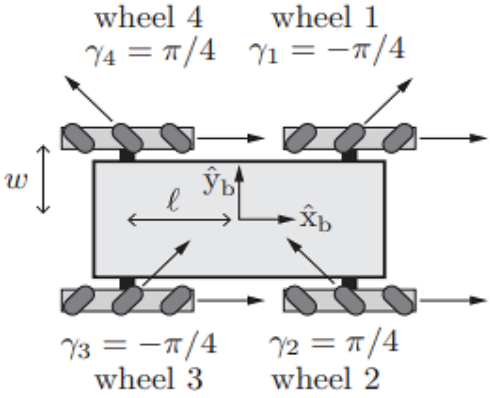


Figure 18: Kinematic models for mobile robots with four mecanum wheels.

In the case $m > 3$, as for the four-wheeled youBot of Figure (18), choosing u such that the previous Equation is not satisfied for any $V_b \in R^3$ implies that the wheels must skid in their driving directions.

Using the notation in Figure (18), the kinematic model of the mobile robot with four mecanum wheels is:

$$\boldsymbol{u = H(0)V_b} = \frac{\boldsymbol{1}}{\boldsymbol{r}} \begin{bmatrix} \boldsymbol{-\ell - w} & \boldsymbol{1} & \boldsymbol{-1} \\ \boldsymbol{\ell + w} & \boldsymbol{1} & \boldsymbol{1} \\ \boldsymbol{\ell + w} & \boldsymbol{1} & \boldsymbol{-1} \\ \boldsymbol{-\ell - w} & \boldsymbol{1} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega_{bz}} \\ \boldsymbol{v_{bx}} \\ \boldsymbol{v_{by}} \end{bmatrix}$$

For the mecanum robot, to move in the direction $+\hat{x}_b$, all wheels drive forward at the same speed; to move in the direction $+\hat{y}_b$, wheels 1 and 3 drive backward and wheels 2 and 4 drive forward at the same speed; and to rotate in the counterclockwise direction, wheels 1 and 4 drive backward and wheels 2 and 3 drive forward at the same speed. Note that the robot chassis is capable of the same speeds in the forward and sideways directions.

If the driving angular velocity of wheel i is subject to the bound $|u_i| \leq u_{i,max}$, i.e.,

$$-u_{i,max} \leq u_i = h_i(0)V_b \leq u_{i,max},$$

then two parallel constraint planes defined by $-u_{i,max} = h_i(0)V_b$ and $u_{i,max} = h_i(0)V_b$ are generated in the three-dimensional space of body twists. Any V_b between these two planes does not violate the maximum driving speed of wheel i, while any V_b outside this slice is too fast for wheel i. The normal direction to the constraint planes is ${h_i}^T(0)$, and the points on the planes closest to the origin are $-u_{i,max}{h_i}^T(0)/\|h_i(0)\|^2$ and $u_{i,max}{h_i}^T(0)/\|h_i(0)\|^2$ [18] [30].

4.2.2.2. Forward Kinematic Model

The forward kinematic model calculates the motion of the robot (i.e., linear and angular velocity) based on the known or measured wheel angular velocities.

Mathematical Form:

$$\boldsymbol{\xi = J^{\dagger} \cdot u}$$
$$\boldsymbol{V_b = r \cdot J^{\dagger} \cdot u}$$

Where:

- $\boldsymbol{\xi = [v_x, v_y, \omega_z]^T}$ is the chassis/body velocity.
- $\boldsymbol{u = [\omega_1, \omega_2, \ldots, \omega_n]^T}$ is the vector of wheel angular velocities.
- $\boldsymbol{J^{\dagger}}$ is the pseudo-inverse of the Jacobian matrix.

General Formula for the Pseudo-Inverse:

There are two cases depending on the dimensions of $\boldsymbol{J}$:

1. **Overdetermined system (more equations than unknowns, m > n)**
   This is typical for a robot with **4** wheels (**4** equations) and 3 DoF (3 variables):
   $$\boldsymbol{J^{\dagger} = (J^T J)^{-1} J^T}$$
2. **Underdetermined system (more unknowns than equations, $m < n$)**

This is less common in mobile robots, but for completeness:

$$J^{\dagger} = J^{T}(JJ^{T})^{-1}$$

The Pseudo-Inverse of the Jacobian (J) Computation

By Letting $a = \ell + w, J$ becomes:

$$J = \begin{bmatrix} -a & 1 & -1 \\ a & 1 & 1 \\ a & 1 & -1 \\ -a & 1 & 1 \end{bmatrix}$$

$$J^{T}J = \begin{bmatrix} -a & a & a & -a \\ 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -a & 1 & -1 \\ a & 1 & 1 \\ a & 1 & -1 \\ -a & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4a^{2} & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

Since the matrix is diagonal:

$$(J^{T}J)^{-1} = \begin{bmatrix} \frac{1}{4a^{2}} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}$$

$$J^{\dagger} = (J^{T}J)^{-1}J^{T} = \begin{bmatrix} \frac{1}{4a^{2}} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} -a & a & a & -a \\ 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{-1}{4a} & \frac{1}{4a} & \frac{1}{4a} & \frac{-1}{4a} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{-1}{4} & \frac{1}{4} & \frac{-1}{4} & \frac{1}{4} \end{bmatrix}$$ [18].

4.2.3. Combined System Kinematic Analysis:

For a robot arm mounted on a mobile base, mobile manipulation describes the coordination of the motion of the base and the robot joints to achieve a desired motion at the end-effector. Typically, the motion of the arm can be controlled more precisely than the motion of the base, so the most popular type of mobile manipulation involves driving the base, parking it, letting the arm perform the precise motion task, then driving away. In some cases, however, it is

advantageous, or even necessary, for the end-effector motion to be achieved by a combination of motion of the base and motion of the arm.

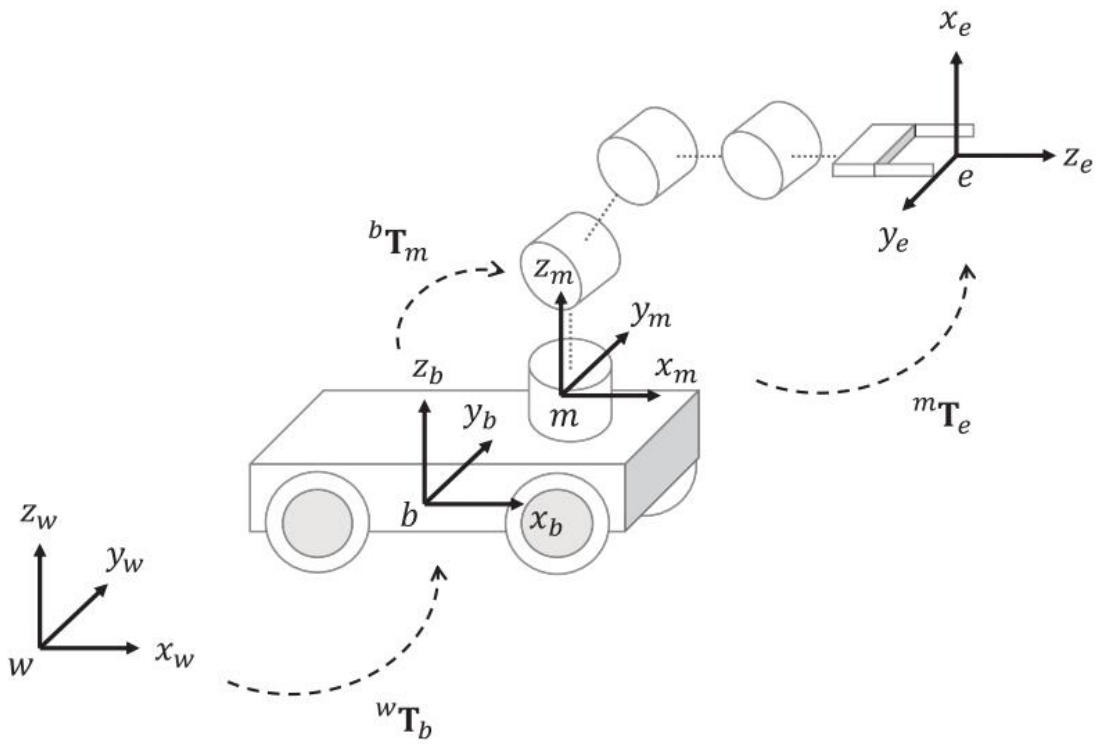


Figure 20 : Kinematic chain of a mobile manipulators [18].

Defining the fixed space frame $\{\boldsymbol{s}\}$, the chassis frame $\{\boldsymbol{b}\}$, a frame at the base of the arm $\{\boldsymbol{0}\}$, and an end-effector frame $\{\boldsymbol{e}\}$, the configuration of $\{\boldsymbol{e}\}$ in $\{\boldsymbol{s}\}$ is

$$\boldsymbol{X(q,\theta) = T_{sb}(q)T_{b0}T_{0e}(\theta)}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{n}}$ is the set of arm joint positions for the $\boldsymbol{n}$-joint robot, $\boldsymbol{T_{0e}(\theta)}$ is the forward kinematics of the arm, $\boldsymbol{T_{b0}}$ is the fixed offset of $\{\boldsymbol{0}\}$ from $\{\boldsymbol{b}\}$, $\boldsymbol{q} = (\boldsymbol{\phi}, \boldsymbol{x}, \boldsymbol{y})$ is the planar configuration of the mobile base, and

$$\boldsymbol{T_{sb}(q)} = \begin{bmatrix} \boldsymbol{cos\phi} & \boldsymbol{-sin\phi} & \boldsymbol{0} & \boldsymbol{x} \\ \boldsymbol{sin\phi} & \boldsymbol{cos\phi} & \boldsymbol{0} & \boldsymbol{y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{z} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

where $\mathbf{z}$ is a constant indicating the height of the $\{\boldsymbol{b}\}$ frame above the floor.

$$\boldsymbol{T_{sb}(q)} = \begin{bmatrix} \boldsymbol{cos\phi} & \boldsymbol{-sin\phi} & \boldsymbol{0} & \boldsymbol{x} \\ \boldsymbol{sin\phi} & \boldsymbol{cos\phi} & \boldsymbol{0} & \boldsymbol{y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0.14} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

$$\boldsymbol{T_{b0}} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0.15} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$$

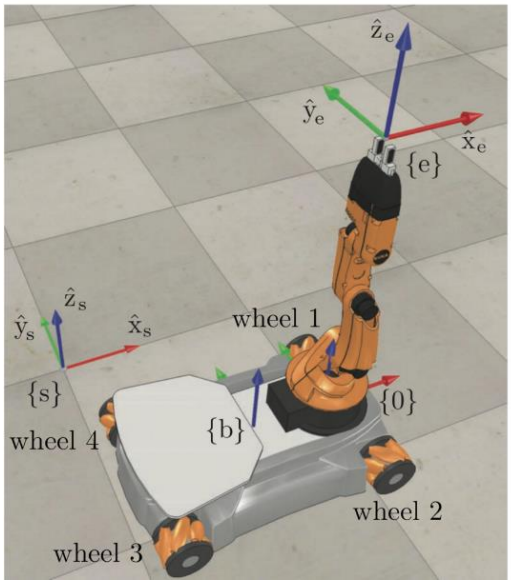


Figure21 : The space frame $\{s\}$ and the frames $\{b\}$, $\{0\}$, and $\{e\}$attached to the mobile manipulator

Both the robotic arm and the mobile base have an impact on the end-effector motion in mobile manipulator systems like the KUKA YouBot. Consequently, both sources of motion must be taken into account in the overall system kinematics. The end-effector's instantaneous spatial velocity, also known as the twist, and the system's generalized velocities are unifiedly mapped by the combined Jacobian matrix.

4.2.3.1. End-Effector Twist Definition

The **end-effector twist**, denoted as $\dot{x}_{EE} \in \mathbb{R}^6$, represents the instantaneous motion of the end-effector in space. It consists of both linear and angular velocity components:

$$\dot{\boldsymbol{x}}_{\boldsymbol{EE}} = \begin{bmatrix} \boldsymbol{V_{EE}} \\ \boldsymbol{\omega_{EE}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v_x} \\ \boldsymbol{v_y} \\ \boldsymbol{v_z} \\ \boldsymbol{\omega_x} \\ \boldsymbol{\omega_y} \\ \boldsymbol{\omega_z} \end{bmatrix}$$

where:

- V_{EE}: translational velocity of the end-effector (in the inertial or spatial frame).
- ω_{EE}: angular velocity of the end-effector frame.

This 6D vector fully characterizes the instantaneous motion of a rigid body in 3D space.

4.2.3.2. Generalized Velocity Vector

The generalized velocity vector of the mobile manipulator comprises both the base and arm joint velocities:

$$\dot{\boldsymbol{q}}_{\boldsymbol{combined}} = \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{base}} \\ \dot{\boldsymbol{q}}_{\boldsymbol{arm}} \end{bmatrix} \in \mathbb{R}^{\boldsymbol{8}}$$

where:

- $\dot{x}_{base} = \left[\dot{x}, \dot{y}, \dot{\theta}\right]^T$ are the planar translational and rotational velocities of the mobile base.
- $\dot{q}_{arm} = [\dot{q}_1, \dot{q}_2, \dots, \dot{q}_5]^T$ are the angular velocities of the 5 revolute joints of the arm.

4.2.3.3. Combined Jacobian Relationship

The combined Jacobian $J_{combined} \in \mathbb{R}^{6\times 8}$ maps the generalized velocity vector to the end-effector twist:

$$\dot{\boldsymbol{x}}_{\boldsymbol{EE}} = \boldsymbol{J_{combined}} \cdot \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{base}} \\ \dot{\boldsymbol{q}}_{\boldsymbol{arm}} \end{bmatrix} = \underbrace{[\boldsymbol{J_{base}} \quad \boldsymbol{J_{arm}}]}_{\boldsymbol{J_{combined}}} \begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{base}} \\ \dot{\boldsymbol{q}}_{\boldsymbol{arm}} \end{bmatrix}$$

where:

$J_{base} \in \mathbb{R}^{6\times 3}$ is the Jacobian relating the mobile base velocities to the end-effector twist.

$J_{arm} \in \mathbb{R}^{6\times 5}$ is the standard manipulator Jacobian derived from the arm's kinematic chain.

4.2.3.4. Interpretation and Relevance

- The **mobile base Jacobian** J_{base} accounts for how the translational and rotational motion of the platform influences the end-effector pose. In holonomic systems like the YouBot, which can independently translate and rotate in the plane, J_{base} is **non-trivial and essential** for full-body motion coordination.
- The **manipulator Jacobian** J_{arm} captures the differential kinematics of the arm alone, and is computed based on the spatial or body twist representation, typically using geometric or analytical methods.
- The **combined Jacobian** allows the derivation of the required base and joint velocities to achieve a desired end-effector trajectory in space. It also forms the foundation for:
    - Velocity-level inverse kinematics,
    - Resolved-rate motion control,
    - Redundancy resolution techniques,
    - Coordinated trajectory tracking.

4.2.3.5. Inverse Kinematics

In order to map the desired motion (position or velocity) of the end-effector to the corresponding velocities or positions of the mobile base and the robotic arm joints, inverse kinematics (IK) is essential. The combined IK for the KUKA YouBot mobile manipulator, which has a 5-DOF arm and a 3-DOF holonomic base, entails solving for eight generalized variables:

$$\begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{base}} \\ \dot{\boldsymbol{q}}_{\boldsymbol{arm}} \end{bmatrix} \in \mathbb{R}^{\mathbf{8}}$$

Velocity-Level Inverse Kinematics:

Given a desired end-effector spatial twist (a 6D velocity vector) denoted as

$$\dot{\boldsymbol{x}}_{\boldsymbol{EE}} \in \mathbb{R}^{\mathbf{6}}$$

the objective is to compute the corresponding generalized velocity vector of the mobile manipulator:

$$\begin{bmatrix} \dot{\boldsymbol{x}}_{\boldsymbol{base}} \\ \dot{\boldsymbol{q}}_{\boldsymbol{arm}} \end{bmatrix} = \boldsymbol{J}^{\dagger}_{\boldsymbol{combined}} \cdot \dot{\boldsymbol{x}}_{\boldsymbol{EE}}.$$

where:

- $J_{combined}^{\dagger} \in \mathbb{R}^{8\times6}$ is the **Moore-Penrose pseudoinverse** of the combined Jacobian,
- $J_{combined} = [J_{base} \quad J_{arm}] \in \mathbb{R}^{6\times8}$,
- $\dot{x}_{EE}$ represents the desired spatial velocity (linear and angular components) of the end-effector, expressed either in the base frame or space frame, depending on the Jacobian's definition.

The pseudoinverse is used to obtain a generalized solution when the system is either overactuated or underactuated. It ensures a consistent mapping from task-space velocities to actuator velocities even when the Jacobian is not square or invertible. In cases of near-singular configurations, **regularization techniques** (e.g., damping) can be applied to enhance numerical stability [18]

4.2.3.6. Workspace Dexterity and Manipulability Analysis:

To further characterize the mobile manipulator's capability to generate motion or force in arbitrary directions, we evaluate its manipulability. Manipulability, as introduced by Yoshikawa, quantifies how "dexterous" a robot is at a given configuration.

The manipulability index is defined as:

$$\boldsymbol{\omega(q) = \sqrt{\det\left(J(q)J^{T}(q)\right)}}$$

where $J(q)$ is the Jacobian matrix at configuration q. A **higher $\boldsymbol{\omega}$** indicates greater freedom of motion and better directional control, while $\omega = 0$ corresponds to a singular configuration.

In the case of the KUKA YouBot, the combined Jacobian $J_{combined} \in \mathbb{R}^{6\times8}$, which includes contributions from both the mobile base and manipulator joints, is used to compute the overall manipulability of the system.

This metric serves as a **practical indicator** in both:

- **Trajectory planning**: to avoid low-manipulability regions of the workspace,
- **Control design**: to anticipate singular behaviors and enable smoother inverse kinematics solutions.

Optionally, manipulability can also be visualized over the workspace to identify **regions of high dexterity**, guiding object placement and base positioning during pick-and-place tasks [32].

4.2.3.7. Handling Near-Singular Configurations: Regularization Techniques

In velocity-level inverse kinematics, we compute the joint and base velocities using the **Moore-Penrose pseudoinverse** of the combined Jacobian:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}}_{\boldsymbol{EE}}$$

However, if the Jacobian $J \in \mathbb{R}^{6\times 8}$ becomes **ill-conditioned or near-singular**—for example, when its determinant is close to zero or its smallest singular values approach zero—the standard pseudoinverse becomes **numerically unstable**. This instability can cause **very large or erratic joint and base velocities**, even in response to small desired end-effector velocities.

To ensure robust control in such situations, **regularization techniques** are introduced to stabilize the computation. The most widely used method is the **Damped Least Squares (DLS)** pseudoinverse, also known as **Tikhonov regularization** or the **Levenberg-Marquardt method**.

Damped Least Squares (DLS) Pseudoinverse

The damped pseudoinverse modifies the standard formula to:

$$\boldsymbol{J}_{\boldsymbol{\lambda}}^{\dagger} = \boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{J}\boldsymbol{J}^{\boldsymbol{T}} + \boldsymbol{\lambda}^{\boldsymbol{2}}\boldsymbol{I})^{-\boldsymbol{1}}$$

where:

- $\lambda > 0$ is a small damping factor
- I is the identity matrix of appropriate dimension.

This formulation introduces a **stabilizing term** $\lambda^2 I$, which prevents the inversion process from blowing up due to small singular values.

This technique:

- **Suppresses large, unrealistic joint velocities** near singularities,
- **Improves numerical conditioning**, especially during quick directional changes,
- **Smooths controller behavior**, avoiding sharp velocity spikes.

when:

- $\lambda = 0$:

You recover the standard pseudoinverse $J^{\dagger}$,with no regularization.

- $\lambda > 0$:

You gain **robustness and stability**, at the cost of a small, controlled deviation from the exact solution.

4.2.3.8. Choosing the Damping Factor λ

- A **small fixed value** (e.g., λ=0.01) often suffices for typical tasks.
- For more sensitive systems, **adaptive damping** may be used, based on the **Jacobian's condition number** or **smallest singular value**:

$$\boldsymbol{\lambda = \frac{\epsilon}{\sigma_{min} + \epsilon}}$$

where:

  - ϵ is a small positive constant,
  - σ_{min} is the smallest singular value of J.

Regularization is a **critical necessity**, not an optional enhancement, in practical control of hybrid and redundant robotic systems such as the **KUKA YouBot**. The integration of a holonomic mobile base with a 5-DOF robotic arm results in an **8-DOF kinematic system**, which inherently increases the likelihood of encountering **near-singular configurations** during motion.

In such scenarios, the Jacobian matrix becomes ill-conditioned, leading to unstable or unbounded joint and base velocity commands when using standard inverse kinematics methods.

By incorporating **damped pseudoinverse regularization** into the velocity-level inverse kinematics:

- **Numerical stability** is preserved even in configurations close to singularity,
- **Actuators are safeguarded** against infeasible or unsafe velocity commands,
- **Smooth and reliable task execution** is maintained, which is essential for precision operations like **pick-and-place**, where even small deviations can lead to failure.

Ultimately, regularization ensures that the robot remains **controllable, safe, and predictable** across its full workspace, enabling more robust and resilient control strategies. [33]

4.2.3.9. Proposed Strategy of Work:

4.2.3.9.1. Generating the Desired Trajectory:

This work uses a structured reference trajectory made up of eight consecutive segments to specify the desired motion of the robot's end-effector. Every segment has a specific function in carrying out an entire pick-and-place task, guaranteeing accurate and seamless object manipulation. The following are the trajectory segments:

1. Initial to Standoff Configuration: In order to prepare for grasping, the end-effector shifts from its starting position to a "standoff" position a few centimeters above the target object.
2. Approach to Grasp: From the standoff to the grasp configuration, the end-effector translates vertically downward.
3. Gripper Closure: While keeping the end-effector in a fixed position, the gripper closes on the object.
4. Retreat to Standoff: The end-effector returns to the standoff position above the object it is grasping by moving vertically upward.
5. Travel to the Position Standoff: By coordinating the movements of the manipulator arm and the mobile base, the end-effector moves from the pickup standoff to a standoff position above the placement location.
6. Descent to Placement: The end-effector descends to the object's ultimate placement position.
7. Gripper Opening: Once more maintaining a fixed pose, the gripper opens to release the object.
8. Retreat to Standoff: To finish the manipulation, the end-effector retreats vertically to the last standoff configuration.

To guarantee seamless transitions and reliable task execution, these segments are meticulously designed utilizing the concepts of trajectory generation, time-scaling, and kinematic control. Interestingly, gripper actuation segments 3 and 7 simulate grasping and releasing, respectively, by toggling the gripper state while maintaining constant end-effector poses. In order to provide smooth velocity and acceleration profiles, vertical translation segments 2, 4, 6, and 8 use short, linear vertical motions that are planned with polynomial time scaling. Either constant screw motions or decoupled Cartesian translations and rotations, time-parameterized by third- or fifth-order polynomials, are needed for the coordinated motion of the mobile base and arm in the longer transition segments (1 and 5), which are planned through interpolation in SE (3) space [34].

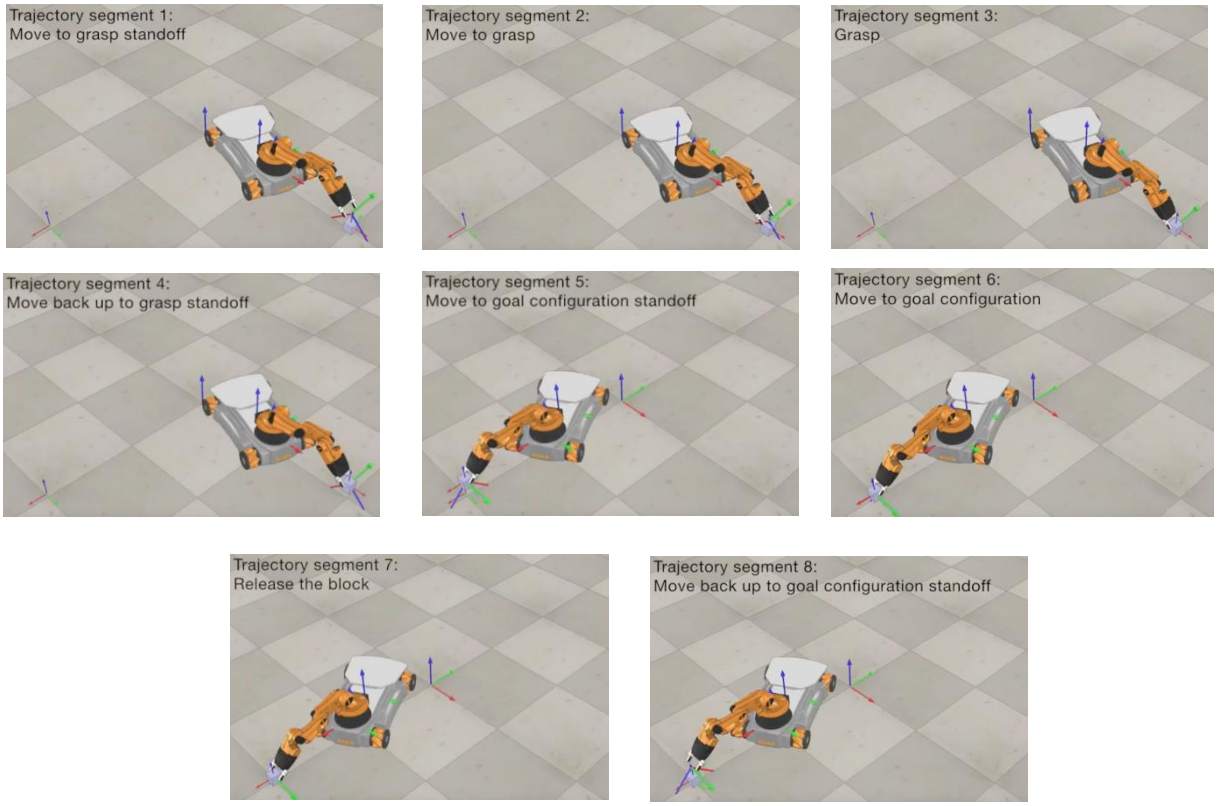


Figure 22: The Trajectory Segments

4.2.3.9.2. Polynomial Time Scaling for Smooth Trajectories:

Polynomial time scaling functions are used to produce smooth trajectories that start and finish at rest, that is, with zero acceleration and velocity. By avoiding sudden jerks or discontinuities that could harm the hardware or impair control performance, this guarantees practical and stable robot motion.

1. Cubic (Third-Order) Time Scaling:

The cubic time-scaling function is defined as:

$$s(t) = 3(\frac{t}{T})^2 - 2(\frac{t}{T})^3$$

with boundary conditions ensuring start and end at rest:

$$s(0) = 0, \quad \dot{s}(0) = 0, \quad s(T) = 1, \quad \dot{s}(T) = 0$$

The corresponding velocity and acceleration profiles are:

$$\dot{s}(t) = \frac{6t}{T^2} - \frac{6t^2}{T^3}, \quad \ddot{s}(t) = \frac{6}{T^2} - \frac{12t}{T^3}$$

This scaling is simple to implement and guarantees zero velocity at the boundaries. However, it produces non-zero acceleration at the endpoints, which implies infinite jerk (the rate of change of acceleration), potentially causing wear or instability in sensitive actuators.

2. Quintic (Fifth-Order) Time Scaling:

For smoother motion, especially when minimizing jerk is critical, the quintic polynomial time scaling is employed:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

where coefficients a_i are determined by imposing six boundary conditions:

$$s(0) = 0, \quad \dot{s}(0) = 0, \quad \ddot{s}(0) = 0, \quad s(T) = 1, \quad \dot{s}(T) = 0, \quad \ddot{s}(T) = 0$$

The closed-form solution reduces to:

$$s(t) = 10\left(\frac{t}{T}\right)^3 - 15\left(\frac{t}{T}\right)^4 + 6\left(\frac{t}{T}\right)^5$$

with velocity and acceleration given by:

$$\dot{s}(t) = \frac{30t^2}{T^3} - \frac{60t^3}{T^4} + \frac{30t^4}{T^5}, \quad \dot{s}(t) = \frac{60t}{T^3} - \frac{180t^2}{T^4} + \frac{120t^3}{T^5}$$

This polynomial ensures zero velocity and acceleration at both start and end, producing the smoothest trajectory and effectively eliminating jerks.

For high-speed, precise, or heavy-load manipulation tasks where smoothness and less mechanical stress are essential, quintic time scaling is recommended. It guarantees better trajectory tracking performance and actuator longevity despite having a marginally higher computational complexity than cubic scaling. [18]

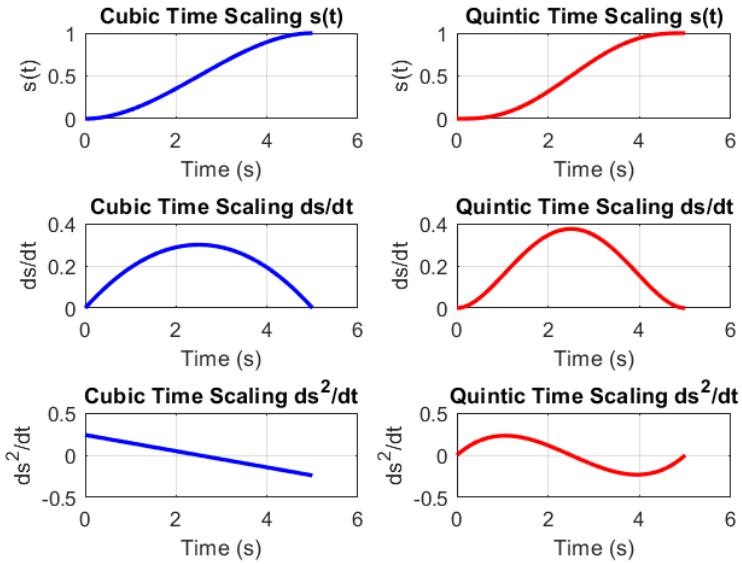


Figure 23: Time Scaling Profiles: Cubic vs. Quintic

For both cubic and quintic polynomial trajectories over a fixed time interval, the plots show the time scaling functions $s(t)$, their first derivatives ds/dt (velocity profiles), and their second derivatives d^2s/dt^2 (acceleration profiles). The progression along a trajectory from beginning to end is parameterized by the time scaling function $s(t)$. Although smooth velocity with zero velocity at the beginning and end is ensured by the cubic time scaling, abrupt jerks are caused by discontinuous acceleration. The quintic time scaling, on the other hand, creates a smoother and more physically realistic motion profile by starting and ending with zero velocity and zero acceleration, which ensures smoothness in position, velocity, and acceleration. Because of its smoothness, quintic scaling is preferred for safe and accurate motion in robotic systems by lowering mechanical stress and vibrations.

4.2.3.9.3. Trajectory Interpolation in SE (3): Cartesian vs. Screw Motion

One of two methods—Cartesian interpolation or Screw (twist-based) interpolation—can be used to perform the geometric interpolation between start and goal poses in SE (3) after a smooth time scaling function $s(t)$ has been defined. Screw trajectories use the exponential map of a spatial twist to create a continuous transformation, utilizing the mathematical structure of rigid body motion:

$$X(s) = X_{start} \cdot \exp\left(\log\left(X_{start}^{-1} X_{end}\right) \cdot s\right)$$

This technique ensures smooth, physically consistent motion along the shortest path in the manifold of SE (3) by capturing coupled translation and rotation along a single screw axis. Because it prevents an artificial division between position and orientation, it is especially advantageous for motions with significant or simultaneous rotational and translational components.

Alternatively, **Cartesian interpolation** separates translation and rotation, computing them independently as:

$$p(s) = p_{start} + s(p_{end} - p_{start})$$

$$R(s) = R_{start} \cdot \exp\left(\log\left(R_{start}^{-1} R_{end}\right) \cdot s\right)$$

This method is computationally efficient and frequently adequate for structured tasks where rotations are moderate or restricted, such as vertical translations or transitions between known grasping and standoff configurations, even though it might not strictly follow a screw path. This implementation balances simplicity, efficiency, and trajectory smoothness by purposefully using Cartesian interpolation in these situations. However, because of its faithfulness to rigid-body motion theory, screw-based interpolation continues to be the recommended option for more complex or dynamic motions. [18]

The figure, which combines Cartesian and Screw motion segments, shows the end-effector trajectory of the KUKA YouBot mobile manipulator carrying out a pick-and-place task. While red segments indicate screw trajectories, blue segments represent Cartesian trajectories. Different motion interpolation techniques used during the task are distinguished by these trajectory types. Every ten steps, the end-effector frame is displayed with RGB arrows (red for the X-axis, green for the Y-axis, and blue for the Z-axis) to show how the pose changes as the path progresses. All things considered, the plot functions as a visual representation of motion planning techniques in 3D space as well as a confirmation of trajectory generation.

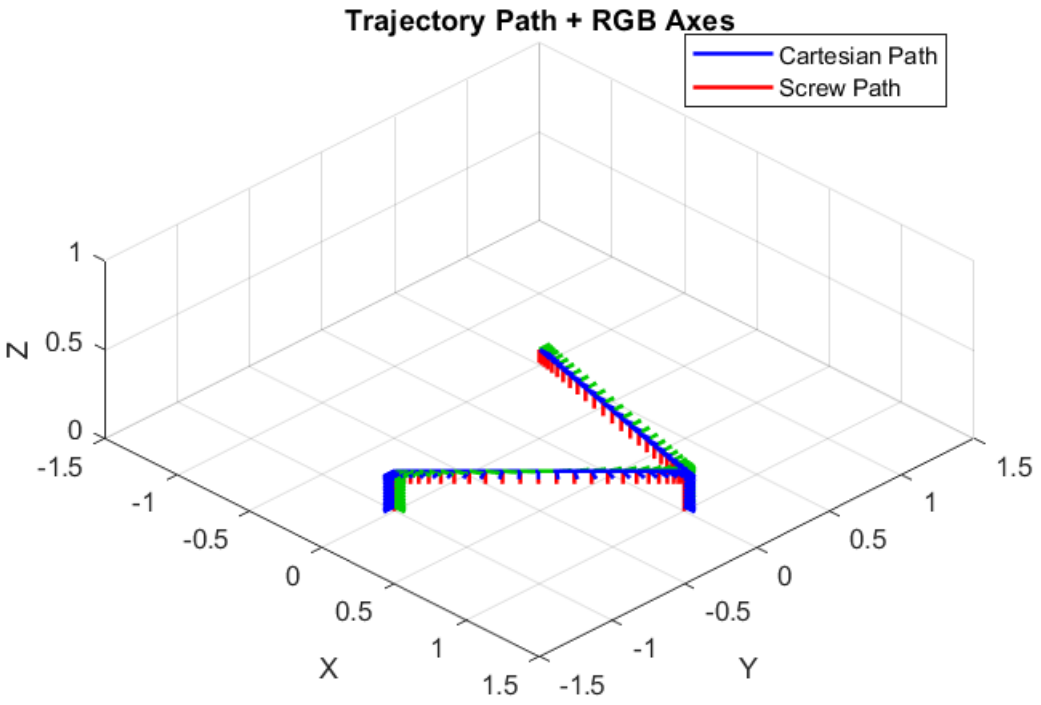


Figure 24: Pick-and-Place Trajectory: Cartesian and Screw Motion Segments

4.2.3.9.4. Application to End-Effector Trajectory Generation

Each segment's trajectory is produced by interpolating between start and goal poses in SE (3) over the designated time interval using the chosen polynomial time scaling, usually the quintic. The end-effector pose at each timestep is described by a sequence of smooth transformation matrices $T_{se}(t)$.

A corresponding pose vector $X(t)$ that represents position and orientation is taken from each pose matrix. The desired end-effector twist $V_{ee}(t)$ is calculated by differentiating successive poses. The inverse kinematics and control layers use this twist as the motion input, allowing the manipulator arm and robot base to move in unison along the predetermined path.

4.2.3.9.5. End-Effector Pose and Twist Extraction

After generating the reference trajectory as a time-parametrized sequence of transformation matrices $T_{se}(t)$, each matrix encapsulates the end-effector's position and orientation in the space frame and fully describes the desired pose at a given instant. To be more precise, every transformation matrix $T_{se}(t)$ can be broken down as follows:

$$T_{se}(t) = \begin{bmatrix} R(t) & p(t) \\ 0 & 1 \end{bmatrix}$$

where $R(t) \in SO(3)$ is the rotation matrix and $p(t) \in \mathbb{R}^3$is the position vector.

The **spatial velocity** of the end-effector—called the **body-frame twist** and denoted $V_{ee}(t) \in \mathbb{R}^6$—represents its instantaneous angular and linear velocities, expressed in its own frame. This twist can be computed using two main methods:

- **Discrete Approximation Using Lie Group Theory:**

When the trajectory is sampled at discrete time steps, the twist is approximated from two successive poses using:

$$[V_{ee}]^\wedge \approx \frac{1}{\Delta t} \log\left(T_{se}^{-1}(t) \cdot T_{se}(t + \Delta t)\right)$$

Where:

- $\log(\cdot)$ is the matrix logarithm, mapping an element of SE (3) to its associated Lie algebra se (3), which represents twists.
- The operator $(\cdot)^\wedge$ maps a 6D twist vector into its 4×4 matrix form.
- The result is a matrix representation of the twist describing the relative motion between the two configurations.

This method generalizes finite differencing to curved spaces like SE (3), offering a geometrically consistent way to approximate motion between poses.

- **Analytical Expression (When Derivatives Are Known)**

If the time derivative $\dot{T}_{se}(t)$ is available (e.g., in symbolic computation or simulation), the twist can be computed exactly as:

$$[V_{ee}(t)]^\wedge = T_{se}^{-1}(t) \cdot \dot{T}_{se}(t)$$

This expression is derived from the differential kinematics of rigid bodies and yields the true body-frame twist without relying on discrete approximations.

Recovering the 6D Twist Vector

In both methods, the 6D twist vector $V_{ee}(t)$—comprising angular and linear velocity components—is obtained from its matrix form by applying the vee **operator**, which is the inverse of $(\cdot)^{\wedge}$:

$$V_{ee}(t) = [V_{ee}(t)]^{\vee}$$

The twist vector can be explicitly expressed as:

$$V_{ee}(t) = \begin{bmatrix} \omega(t) \\ v(t) \end{bmatrix}$$

where $\omega(t) \in \mathbb{R}^3$ is the angular velocity vector, and $v(t) \in \mathbb{R}^3$ is the linear velocity vector. [18]

In practical implementations, this operation is handled by utility functions such as se3ToVec () in MATLAB, which convert the twist matrix into a standard 6D column vector.

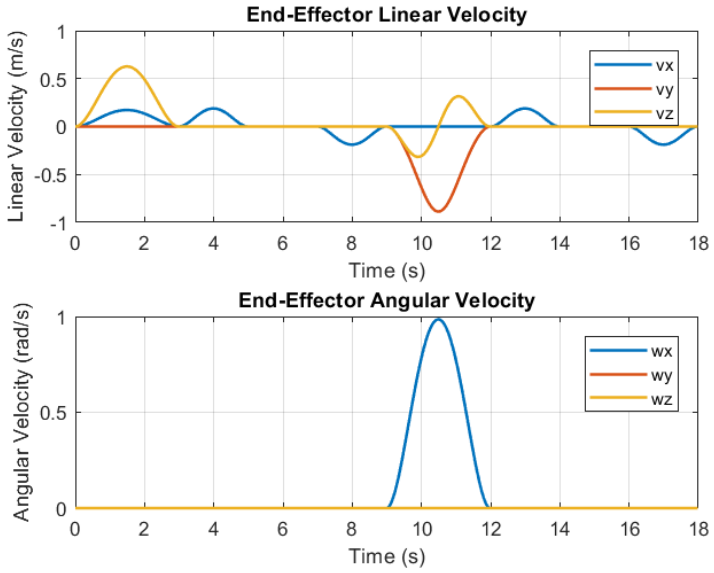


Figure25 : Time Evolution of the End-Effector Twist in the body Frame

Eight-time segments make up the end-effector's motion trajectory, each of which reflects unique velocity characteristics in relation to the body frame and is in line with the pick-and-place task's physical requirements. The end-effector moves from its starting position to the first stand-off configuration in the first segment. It shows non-zero velocities in the x- and z-axes of the body frame ($v_x > 0, v_z = 0$), which means that it moves forward and upward in its local frame to reach the approach position safely. The second segment moves straight ahead in its local frame to

reach the pre-grasp pose, continuing with velocity only along the end-effector's x-axis.

All velocity components are zero in the third segment because the gripper closes around the object while the end-effector remains stationary. The velocity along the local x-axis turns negative ($v_x < 0$) in the fourth segment, signifying a retreat along the same approach direction. This guarantees that the end-effector returns precisely along the path it entered. As the robot moves from the first stand-off to the second stand-off above the goal in the fifth segment, both rotation about the -axis ($\omega_x \neq 0$) and velocity components along the body y- and z-axes show a combined motion of lifting, shifting laterally, and reorienting the gripper to match the cube's goal orientation. The velocity profile defined in the end-effector's frame is in good agreement with the generated trajectory and exhibits smooth quintic (fifth-order polynomial) time-scaling, which guarantees physical feasibility, smooth accelerations, and accurate execution of the manipulation task. The sixth segment reflects the second segment's velocity pattern, as the end-effector advances again in its x-axis direction to reach the goal pose, the seventh segment remains stationary again as the gripper opens to release the object, and the eighth segment shows an upward motion, lifting away from the placement pose.

4.2.3.9.6. Joint and Base Velocity Extraction from the Combined Jacobian

Given the end-effector spatial twist $V_{ee}(t) \in \mathbb{R}^6$, representing the linear and angular velocities of the end-effector, the joint velocities $\dot{q} \in \mathbb{R}^n$ of the manipulator arm and the base velocities $\dot{u} \in \mathbb{R}^m$ of the mobile platform can be computed using the combined Jacobian matrix $J_{combined} \in \mathbb{R}^{6\times(n+m)}$.

The combined Jacobian relates the joint and base velocities to the end-effector twist as:

$$V_{ee}(t) = J_{combined} \begin{bmatrix} \dot{u}(t) \\ \dot{q}(t) \end{bmatrix}$$

where:

- $\dot{u}(t) \in \mathbb{R}^m$ represents the velocity vector of the mobile base (e.g., linear and angular velocities depending on the base kinematics),
- $\dot{q}(t) \in \mathbb{R}^n$ represents the manipulator joint velocities,
- $J_{combined}$ is composed by horizontally stacking the base Jacobian $J_b \in \mathbb{R}^{6\times m}$ and the arm Jacobian $J_a \in \mathbb{R}^{6\times n}$:

$$J_{combined} = [J_b \quad J_a]$$

To solve for the velocity inputs $\begin{bmatrix}\dot{u}\\ \dot{q}\end{bmatrix}$, the inverse kinemsatics at the velocity level is performed by computing the pseudoinverse of $J_{combined}$:

$$\begin{bmatrix}\dot{u}(t)\\ \dot{q}(t)\end{bmatrix} = J^{\dagger}_{combined} V_{ee}(t)$$

where $J^{\dagger}_{combined}$ denotes the Moore-Penrose pseudoinverse, used to handle redundant or non-square Jacobians and to obtain a least-squares solution.

This calculation makes it possible to precisely and smoothly execute complex trajectories by coordinating control of the manipulator joints and mobile base to reach the intended end-effector velocities. [18]

4.2.3.9.7. Manipulability Analysis and Singularity Handling

It is crucial to assess the mobile manipulator's manipulability in order to guarantee steady and agile operation throughout the trajectory. The manipulability index measures how well the robot can move in different directions with its current setup and aids in determining how close it is to kinematic singularities, which occur when the Jacobian loses rank and some motions become unfeasible or poorly conditioned.

For the combined system, the **Yoshikawa manipulability index** is computed from the combined Jacobian $J_{combined}(t)$ as:

$$\mu(t) = \sqrt{\det\left(J_{combined}(t) J^{T}_{combined}(t)\right)}$$

While values near zero indicate the robot is close to a singularity, which can lead to abrupt or unstable velocity commands, a high value of $\mu(t)$ indicates a well-conditioned configuration. [32]

The damped least-squares inverse is used in place of the Moore-Penrose pseudoinverse of the Jacobian to robustly handle such circumstances, particularly during inverse velocity kinematics:

$$\boldsymbol{J^{\dagger}_{combined} = J^{T}_{combined}\left(J_{combined} J^{T}_{combined} + \lambda^{2} I\right)^{-1}}$$

where $\lambda > 0$ is a small damping factor. This regularization stabilizes the inverse solution near singularities and prevents large or erratic joint and base velocities.

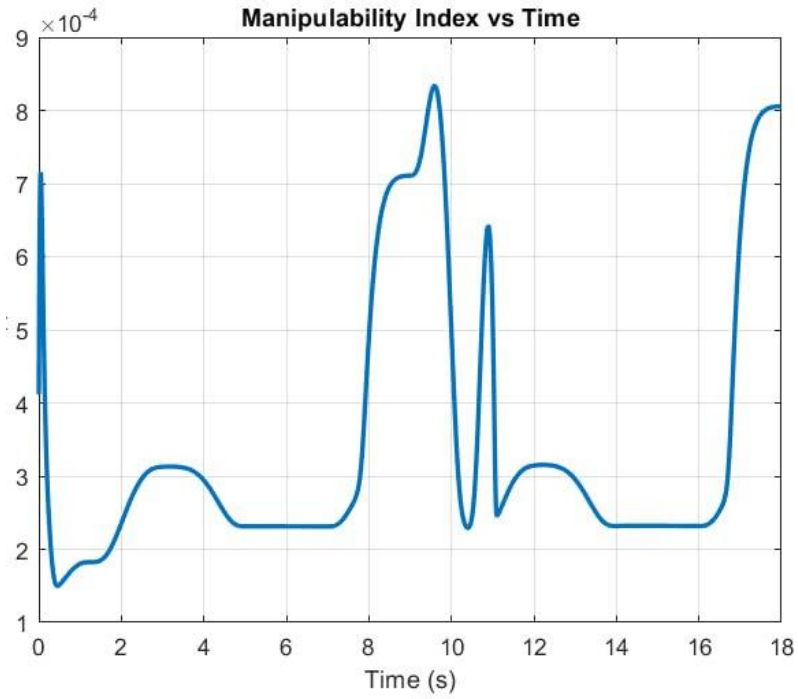


Figure 26: Manipulability Index over Time for the Combined System Jacobian

The manipulability index $\mu(t)$ was plotted over time along the executed trajectory to provide additional evidence for the analysis. The evolution of manipulability is clearly depicted in this plot, which also identifies any critical singular regions and supports the application of regularized inversion techniques during those sections. The analysis acts as a diagnostic tool to confirm that the generated trajectory and velocity control are safe and effective. [33]

4.2.3.9.8. Velocity Control for Trajectory Tracking

After the inverse velocity kinematics have been used to calculate the desired velocity commands, $\dot{u}_{des}(t)$ and $\dot{q}_{dest}(t)$, a control strategy that motivates the manipulator arm and mobile base to follow these reference velocities over time must be put into place. This layer makes sure that the system's actual motion precisely follows the intended end-effector trajectory.

In this implementation, observers estimate the complete state from a limited set of measurements, and state-space controllers created via pole placement control both the manipulator arm and the mobile base. When compared to traditional PID

schemes, this contemporary control architecture provides superior dynamic performance and robustness.

- **Mobile Base Control**: A linearized base dynamics model is used to derive the control law for a full-state feedback controller. In order to accurately track the desired base velocity $\dot{u}_{des}(t)$, which usually includes $[v_x, v_y, \omega_z]$, an observer reconstructs the entire velocity state from the available sensor data (such as wheel encoders or odometry).
- **Arm Control**: The dynamic model of each joint is used to determine the state feedback law that governs it. In order for the controller to enforce the intended joint velocity profile $\dot{q}_{des}(t)$, which is obtained from the trajectory planner, observers estimate the joint velocities from position measurements (e.g., encoders).

In addition to compensating for unmodeled dynamics and measurement noise using the observer-based estimation framework, this control layer guarantees steady, smooth trajectory tracking throughout the entire mobile manipulator system.

5. Dynamic Modeling

5.1. Dynamic Modeling of Manipulator Robot

This dynamic study focuses on the KUKA youBot's 5-degree-of-freedom (DOF) serial robotic arm.

The following assumptions are applied:

- The arm consists of five rigid-body links connected via revolute joints.
- Each link has a known mass, center of mass, and moment of inertia.
- All joints are actuated and rotary.
- No damping, friction, or joint compliance is considered in this base model.
- The gravitational field acts in the negative Z-direction of the global (base) frame.
- The generalized coordinates $(\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3}, \boldsymbol{q_4}, \boldsymbol{q_5})$ represent the five revolute joint angles.
- Denavit–Hartenberg (DH) conventions are used for the kinematic structure.

Symbol	*Definition*
q_i	Generalized coordinate (joint angle) of joint $i, i = 1 \ldots 5$
$q, \dot{q}, \ddot{q}$	Vectors of joint positions, velocities, and accelerations
m_i	Mass of link i
r_{C_i}	Position of center of mass of link i, expressed in base frame
v_{C_i}	Linear velocity of the center of mass of link i
ω_i	Angular velocity of link i
I_i	Inertia tensor of link i about its center of mass
J_{v_i}	Jacobian mapping joint velocities to linear velocity of C_i
J_{ω_i}	Jacobian mapping joint velocities to angular velocity of link i
T_i	Kinetic energy of link i
T	Total kinetic energy of the robot
U	Total potential energy due to gravity
$\mathcal{L}$	Lagrangian function, $\mathcal{L} = T - U$
τ_i	Applied torque at joint i
τ	Vector of joint torques
$M(q)$	Inertia matrix of the manipulator
$C(q, \dot{q})$	Coriolis and centrifugal matrix
$G(q)$	Gravity force vector

Table 4: Symbols and Definitions for robotic arm dynamic modeling

Kinetic Energy

The kinetic energy of the robotic arm comprises both translational and rotational contributions from each of its five links. For a single link, the kinetic energy is calculated as follows:

$$[\boldsymbol{T_i} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{m_i}|\boldsymbol{v_{C_i}}|^{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\omega_i^T I_i \omega_i}]$$

where $\boldsymbol{m_i}$ is the mass of link $\boldsymbol{i}$, $\boldsymbol{v_{C_i}}$ is the linear velocity of its center of mass, $\boldsymbol{\omega_i}$ is the angular velocity, and $\boldsymbol{I_i}$ is the inertia matrix. These velocities are expressed in terms of the joint velocities via Jacobian matrices:

$$[\boldsymbol{v_{C_i} = J_{v_i}(q)\dot{q}}, \quad \boldsymbol{\omega_i = J_{\omega_i}(q)\dot{q}}]$$

The kinetic energy of each individual link can be expressed as a function of the joint velocities and the link's Jacobians. Specifically, for the $\boldsymbol{i^{th}}$ link, the kinetic energy is given by:

$$[\boldsymbol{T_i = \frac{1}{2}\dot{q}^T\left[m_i J_{v_i}^T J_{v_i} + J_{\omega_i}^T I_i J_{\omega_i}\right]\dot{q}}]$$

This formulation accounts for both the translational and rotational motion of each link. To obtain the total kinetic energy of the robotic arm, the individual contributions from all five links are summed:

$$[\boldsymbol{T = \sum_{i=1}^{5} T_i = \frac{1}{2}\dot{q}^T M(q)\dot{q}}]$$

The matrix $\boldsymbol{M(q)}$ is known as the configuration-dependent inertia matrix and encapsulates the mass distribution and geometry of the entire manipulator. It is computed by summing the inertial contributions of each link as follows:

$$[\boldsymbol{M(q) = \sum_{i=1}^{5}\left(m_i J_{v_i}^T J_{v_i} + J_{\omega_i}^T I_i J_{\omega_i}\right)}]$$

Potential Energy

The potential energy of the robotic manipulator arises from gravity acting on each link's mass. For each link, the potential energy depends on the vertical position (height) of the center of mass in the gravitational field:

$$[\boldsymbol{U_i = m_i g z_{C_i}(q)}]$$

where $\boldsymbol{g}$ is the gravitational acceleration and $\boldsymbol{z_{C_i}}$ is the height of the center of mass of link $\boldsymbol{i}$. The total potential energy of the arm is obtained by summing the contributions from all links:

$$[\boldsymbol{U = \sum_{i=1}^{5} m_i g\, z_{C_i}(q)}]$$

Lagrangian Function

The Lagrangian function represents the difference between the kinetic and potential energy of the system. It forms the foundation for deriving the equations of motion:

$$[\mathcal{L}(q, \dot{q}) = T(q, \dot{q}) - U(q)]$$

Substituting the expressions for the system's total kinetic and potential energies into the Lagrangian definition yields the following expanded form:

$$[\mathcal{L} = \frac{1}{2}\dot{q}^T M(q)\dot{q} - \sum_{i=1}^{5} m_i g\, z_{C_i}(q)]$$

Equations of Motion

Using the Lagrange formulation, the equations of motion for each joint can be derived by applying the Euler-Lagrange equation:

$$[\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_\iota}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i]$$

When expressed in matrix form, the dynamics of the 5-DOF robotic arm can be written as:

$$[M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau]$$

Inertia Matrix

As previously defined:

$$[M(q) = \sum_{i=1}^{5}\left[m_i J_{v_i}^T J_{v_i} + J_{\omega_i}^T I_i J_{\omega_i}\right]]$$

Gravity Vector

The gravity vector $G(q)$ accounts for the torques at each joint caused by gravitational forces acting on the links' masses. It is derived by taking the partial derivatives of the potential energy function $U(q)$ with respect to each generalized coordinate q_i:

$$G_i(q) = \frac{\partial U}{\partial q_i}$$

So:

$$\boldsymbol{G(q)} = \left[\frac{\boldsymbol{\partial U}}{\boldsymbol{\partial q_1}} \quad \frac{\boldsymbol{\partial U}}{\boldsymbol{\partial q_2}} \quad \frac{\boldsymbol{\partial U}}{\boldsymbol{\partial q_3}} \quad \frac{\boldsymbol{\partial U}}{\boldsymbol{\partial q_4}} \quad \frac{\boldsymbol{\partial U}}{\boldsymbol{\partial q_5}}\right]^{\boldsymbol{T}}$$

This vector expresses how changes in the configuration of the arm affect the potential energy, and therefore, the torque required to counteract gravity.

Coriolis and Centrifugal Matrix

The Coriolis and centrifugal forces are represented by the matrix $\boldsymbol{C(q,\dot{q})}$, which captures the nonlinear velocity-dependent terms in the dynamics. Each element $\boldsymbol{C_{ij}}$ is computed using the Christoffel symbols of the first kind:

$$\left[\boldsymbol{C_{ij}} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{5}} \frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\frac{\boldsymbol{\partial M_{ij}}}{\boldsymbol{\partial q_k}} + \frac{\boldsymbol{\partial M_{ik}}}{\boldsymbol{\partial q_j}} - \frac{\boldsymbol{\partial M_{jk}}}{\boldsymbol{\partial q_i}}\right)\boldsymbol{\dot{q}_k}\right]$$

This matrix ensures the model accurately reflects the coupling effects and inertial interactions between joints during motion.

Final Dynamic Model

Combining the inertia matrix $\boldsymbol{M(q)}$, the Coriolis and centrifugal matrix $\boldsymbol{C(q,\dot{q})}$, and the gravity vector $\boldsymbol{G(q)}$, we obtain the complete form of the manipulator's equations of motion:

$$\left[\boxed{\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau}}\right]$$

This equation governs the dynamic behavior of the robotic arm and serves as the basis for simulation and control design [18].

5.2. Dynamic Modeling of a Mobile Robot Platform

The equations of motion of a wheeled mobile robot operating on a planar surface are derived using the Lagrangian formalism. Translational and rotational dynamics are accounted for, and the model is structured for platforms equipped with omnidirectional or Mecanum wheels.

Generalized Coordinates

Let the configuration of the robot be given by the vector:

$$\boldsymbol{q} = [\boldsymbol{x} \quad \boldsymbol{y} \quad \boldsymbol{\theta}]^{\boldsymbol{T}}$$

Where:

- $\boldsymbol{x, y}$: represent the position of the robot's center in the inertial (world) frame.
- $\boldsymbol{\theta}$: the robot's orientation about the vertical axis.

The corresponding time derivatives are:

$$\dot{\boldsymbol{q}} = [\dot{\boldsymbol{x}} \quad \dot{\boldsymbol{y}} \quad \dot{\boldsymbol{\theta}}]^{\boldsymbol{T}}$$

Velocity Transformation to Body Frame

Let the velocity in the robot's body-fixed frame be:

$$\boldsymbol{V_b} = [\boldsymbol{v_{bx}} \quad \boldsymbol{v_{by}} \quad \boldsymbol{\omega_{bz}}]^{\boldsymbol{T}}$$

The relation between $\dot{\boldsymbol{q}}$ and $\boldsymbol{V_b}$ is given by:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{y}} \end{bmatrix} = \mathbf{R}(\boldsymbol{\theta}) \begin{bmatrix} \boldsymbol{v_{bx}} \\ \boldsymbol{v_{by}} \end{bmatrix}, \boldsymbol{with\ \omega_{bz} = \dot{\theta}}$$

Where $\mathrm{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

Kinetic Energy

The total kinetic energy **T** of the robot body, modeled as a rigid body is expressed by:

$$\boldsymbol{T} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{m}(\dot{\boldsymbol{x}}^{\mathbf{2}} + \dot{\boldsymbol{y}}^{\mathbf{2}}) + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{I_z}\dot{\boldsymbol{\theta}}^{\mathbf{2}}$$

Substituting $\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}}$ in terms of $\boldsymbol{v_{bx}}, \boldsymbol{v_{by}}$:

$$\boldsymbol{T} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{m}\left(\boldsymbol{v_{bx}}^{\mathbf{2}} + \boldsymbol{v_{by}}^{\mathbf{2}}\right) + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{I_z}\boldsymbol{\omega_{bz}}^{\mathbf{2}}$$

Lagrangian Function

The Lagrangian $\boldsymbol{\mathcal{L}}$ is defined as:

$$\boldsymbol{\mathcal{L}(q, \dot{q}) = T - V = T}$$

(since potential energy $\boldsymbol{V = 0}$ on a flat horizontal plane)

Lagrange's Equations

Lagrange's equations for each generalized coordinate q_i are:

$$\frac{\boldsymbol{d}}{\boldsymbol{dt}}\left(\frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial \dot{q}_i}}\right) - \frac{\boldsymbol{\partial \mathcal{L}}}{\boldsymbol{\partial q_i}} = \boldsymbol{Q_i}$$

Where:

- $\boldsymbol{Q_i}$: generalized non-conservative force associated with $\boldsymbol{q_i}$ (e.g., due to wheel torques).

In this case:

- $\boldsymbol{\partial \mathcal{L}/\partial q_i = 0}$(no explicit configuration dependency),

- The derivatives are only with respect to $\dot{\boldsymbol{q}}$, which are linked to body-frame velocities.

The system in matrix form using the **inertia matrix** $\mathbf{M}(\boldsymbol{q})$is given by:

$$\mathbf{M}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{q}} = \boldsymbol{Q}$$

But since the system is usually controlled and measured in the body frame, it is more convenient to express the dynamics using $\boldsymbol{V_b}$.

Dynamics in Body Frame:

We define the inertia matrix in the body frame:

$$\mathbf{M} = \begin{bmatrix} \boldsymbol{m} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{m} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I_z} \end{bmatrix}$$

Assuming no Coriolis terms ($\boldsymbol{C} = \mathbf{0}$) and a linear viscous friction model:

$$\mathbf{F}(\boldsymbol{V_b}) = \begin{bmatrix} \boldsymbol{f_x}.\boldsymbol{v_{bx}} \\ \boldsymbol{f_y}.\boldsymbol{v_{by}} \\ \boldsymbol{f_\theta}.\boldsymbol{\omega_{bz}} \end{bmatrix}$$

Where:

- $\boldsymbol{f_x}, \boldsymbol{f_y}$: translational friction coefficients,
- $\boldsymbol{f_\theta}$: rotational damping coefficient.

The final dynamic model in the body frame becomes:

$$\mathbf{M} \cdot \dot{\boldsymbol{V}}_{\boldsymbol{b}} + \mathbf{F}(\boldsymbol{V_b}) = \boldsymbol{J}^{\boldsymbol{T}} \cdot \boldsymbol{\tau}$$

Where:

- $\boldsymbol{\tau} \in \boldsymbol{R}^{4\times1}$: torques applied by the four-wheel motors,
- $\boldsymbol{J}^{\boldsymbol{T}}$ transpose of the Jacobian matrix mapping wheel torques into generalized forces.

Final Dynamic Equation (Body Frame)

$$\begin{bmatrix} \boldsymbol{m} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{m} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I_z} \end{bmatrix} \begin{bmatrix} \boldsymbol{v_{bx}} \\ \boldsymbol{v_{by}} \\ .\boldsymbol{\omega_{bz}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{f_x}.\boldsymbol{v_{bx}} \\ \boldsymbol{f_y}.\boldsymbol{v_{by}} \\ \boldsymbol{f_\theta}.\boldsymbol{\omega_{bz}} \end{bmatrix} = \boldsymbol{J}^{\boldsymbol{T}} \cdot \boldsymbol{\tau} \qquad [35]$$

6. Control Strategy

6.1. Arm control

6.1.1. State-Space Formulation of Joint-Space Dynamics

In the design of model-based controllers such as pole placement with state estimation, it is customary to reformulate the second-order nonlinear joint-space dynamics of a robotic manipulator into a first-order state-space representation.

The simplified joint-space dynamics of an n-DOF robotic arm, neglecting Coriolis, centrifugal, gravity, and frictional terms, can be expressed as:

$$\boldsymbol{M(\theta)\ddot{\theta} = \tau}$$

where:

- $\theta \in \mathbb{R}^n$ is the vector of joint positions,
- $\ddot{\theta} \in \mathbb{R}^n$ is the vector of joint accelerations,
- $\tau \in \mathbb{R}^n$ is the control input (joint torques),
- $M(\theta) \in \mathbb{R}^{n \times n}$ is the positive-definite inertia matrix.

To express this system in first-order form, we define the state vector:

$$\boldsymbol{x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^{2n}}$$

Taking the time derivative of the state, we obtain:

$$\boldsymbol{\dot{x} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ M^{-1}(\theta)\tau \end{bmatrix}}$$

the vicinity of a nominal configuration θ_0, and under the assumption of a constant inertia matrix $M \approx M(\theta)$, the system can be approximated as linear time-invariant (LTI). The corresponding state-space form is then:

$$\boldsymbol{\dot{x} = Ax + Bu}$$

with system matrices:

$$\boldsymbol{A = \begin{bmatrix} 0_{n\times n} & I_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix}, \quad B = \begin{bmatrix} 0_{n\times n} \\ M^{-1} \end{bmatrix}}$$

In this case, the identity submatrix in the upper-right block of matrix A reflects the intrinsic coupling between joint positions and velocities through its non-zero structure. This encapsulates the kinematic relation $\dot{\theta} = \dot{\theta}$, meaning that velocity is the time derivative of position. The fact that the lower block of A stays at zero suggests that control inputs, not just state feedback, have a direct impact on accelerations through the inertia matrix.

Therefore, in order to accurately represent the evolution of the system in terms of its state variables, the first-order state-space formulation must include a non-zero system matrix A, even though the physical system displays purely second-order dynamics. [31]

6.1.2. Observer-Based State Feedback via Pole Placement

In many practical scenarios, the full state vector x is not directly measurable. To address this, a Luenberger observer is employed to estimate the full state:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A\hat{x}} + \boldsymbol{Bu} + \boldsymbol{L}(\boldsymbol{y} - \boldsymbol{C\hat{x}})$$

Where:

- $\hat{x}$ is the estimated state,
- $y = Cx$ is the measurement (often partial),
- L is the observer gain matrix chosen such that the estimation error converges to zero exponentially.

The controller then uses the estimated state in the **full-state feedback law**:

$$\boldsymbol{u} = -\boldsymbol{K}(\hat{\boldsymbol{x}} - \boldsymbol{x_{ref}})$$

Where:

- x_{ref} is the desired trajectory (position and velocity),
- K is a gain matrix computed via **pole placement** to assign desired closed-loop poles.

This architecture ensures:

- **Stability** and desired dynamics of the closed-loop system via K,
- **Accurate state estimation** via fast observer dynamics governed by L. [36]

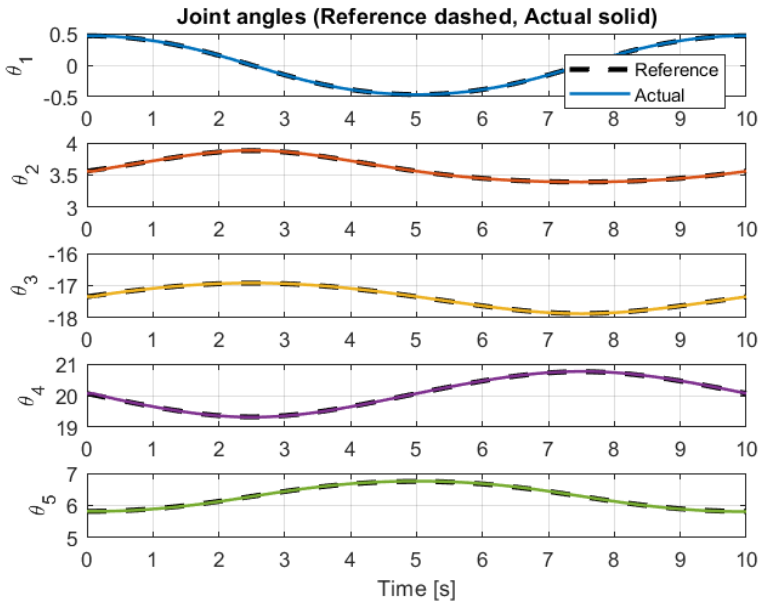


Figure 27: Joint Angles (Reference dashed, Actual solid)

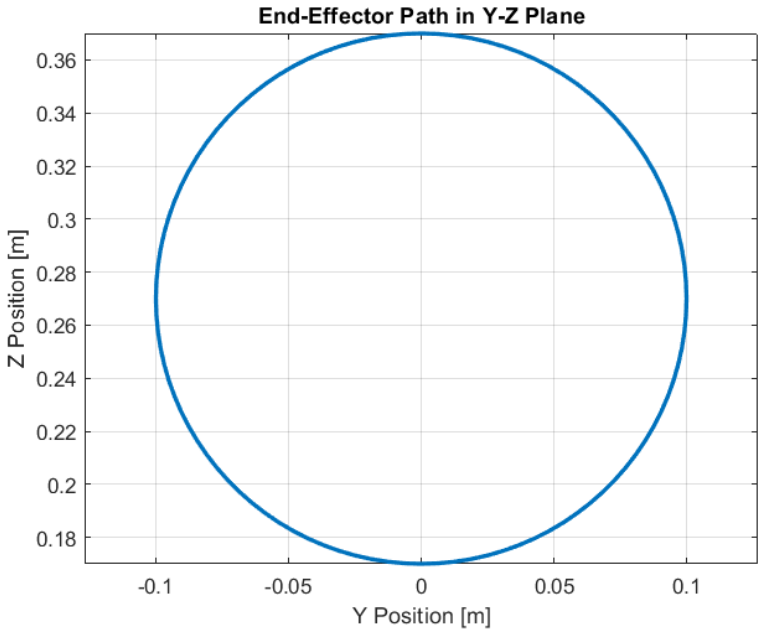


Figure 28: End-Effector Path in Y-Z Plane

The design of the controller and observer gains via pole placement critically influences the dynamic behavior of the robotic arm's closed-loop system. The closed-loop poles, represented as complex conjugate pairs $\lambda_i = \sigma_i \mp j\omega_i$, determine the speed and quality of the system response.

The settling time t_s — the time taken for the system to reach and remain within a small tolerance band around the desired trajectory — is inversely related to the magnitude of the real part of the poles:

$$t_s \approx \frac{4}{|\sigma_i|}$$

here $\sigma_i < 0$ is the real part of the pole. More negative σ_i values yield faster convergence.

The damping ratio ζ, defined as

$$\zeta = \frac{-\sigma_i}{\sqrt{\sigma_i^2 + \omega_i^2}}$$

quantifies the relative contribution of the pole's real and imaginary parts and directly affects oscillations and overshoot.

Maximum overshoot M_p in the system's transient response is linked to the damping ratio by

$$M_p = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}}$$

which decreases as ζ approaches 1, indicating reduced oscillations and a more critically damped system.

In order to ensure faster estimation error convergence, the observer design purposefully places its poles farther to the left in the complex plane than the controller poles. Without sacrificing the closed-loop system's stability, this separation improves state estimation's responsiveness and accuracy. The selection of poles was primarily guided by these mathematical relations, which ensured

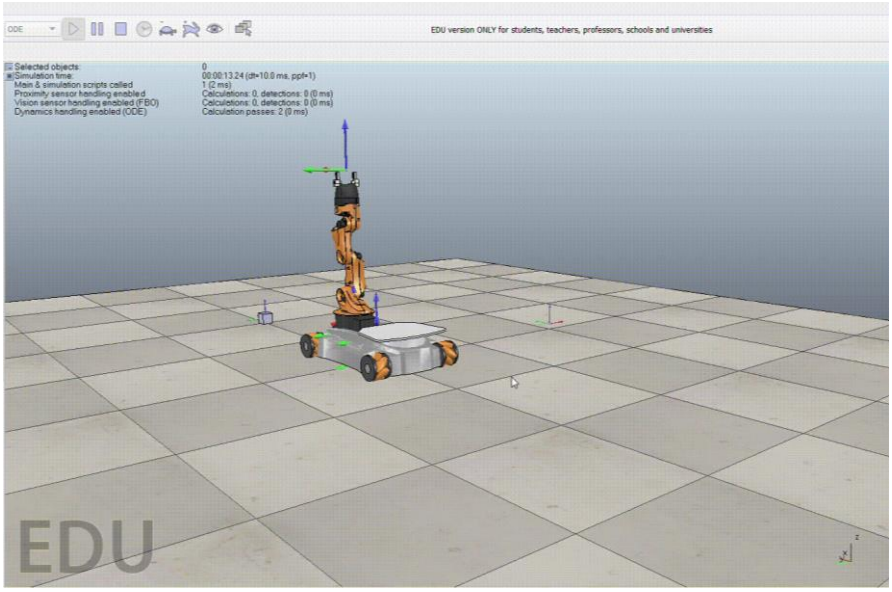


Figure 29: Simulation of Circular End-Effector Trajectory in the Y-Z Plane for the Robotic Arm Using Pole Placement and Observer-Based Control

smooth and accurate arm trajectory tracking by achieving a desirable trade-off between minimal overshoot, fast settling time, and robust state estimation. [37]

6.2. Mobile Base Control

6.2.1. Nonlinear Dynamic Model of a Mobile Base

The full dynamics of a mobile base can be written using Newton–Euler equations as:

$$\boldsymbol{M_b(\xi)\ddot{\xi} + C\left(\xi, \dot{\xi}\right)\dot{\xi} + G(\xi) = J_b^T(\xi)u}$$

Where:

- $\xi = [x, y, \phi]^T$ is the configuration (planar position and orientation),
- $\dot{\xi} = \left[\dot{x}, \dot{y}, \dot{\phi}\right]^T$ is the body velocity,
- M_b: inertia matrix,
- C: Coriolis and centripetal effects,
- G: gravitational and external force terms (zero for planar base),
- J_b^T: Jacobian mapping from wheel inputs to body dynamics,
- u: control input (wheel torques or velocities).

6.2.2. Simplifying Assumptions

To derive a linear approximation, we assume:

- The system operates near a nominal point with **small velocities**,
- Coriolis and gravity terms are negligible: $C(\xi, \dot{\xi}) \approx 0, G(\xi) \approx 0$,
- The inertia matrix is locally **constant**: $M_b(\xi) \approx M_b$,
- Control inputs are **wheel velocities**, mapped through the Jacobian.

Under these conditions, the simplified dynamics reduce to:

$$\boldsymbol{M_b\ddot{\xi} = J_b^T u}$$

6.2.3. Define the State Vector

We define the 6-dimensional state vector as:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\xi} \\ \dot{\boldsymbol{\xi}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{\phi} \\ \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix}$$

Then, the time derivative is: [38]

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{\xi}} \\ \ddot{\boldsymbol{\xi}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{\phi}} \\ \boldsymbol{M_b^{-1} J_b^T u} \end{bmatrix}$$

6.2.4. Linear State-Space Form

We now write the standard first-order state-space representation:

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu}$$

Where the system matrices are:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0_{3\times3}} & \boldsymbol{I_{3\times3}} \\ \boldsymbol{0_{3\times3}} & \boldsymbol{0_{3\times3}} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0_{3\times4}} \\ \boldsymbol{M_b^{-1} J_b^T} \end{bmatrix}$$

Control Strategy Explanation:

In order to achieve stable and precise tracking of a desired path, the control system for the mobile base uses a Luenberger observer in conjunction with a state-space feedback controller that is designed via pole placement.

6.2.5. System Modeling

The mobile base dynamics are modeled in a 6-dimensional state space, with the state vector:

$$\boldsymbol{x} = [\boldsymbol{x\ y\ \theta\ v_x\ v_y\ \omega_z}]^{\boldsymbol{T}}$$

Where (x, y) represent the planar position, θ the orientation, and (v_x, v_y, ω_z) the corresponding linear and angular velocities in the body frame.

The system dynamics matrices A and B are constructed as:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0_{3\times3}} & \boldsymbol{I_{3\times3}} \\ \boldsymbol{0_{3\times3}} & \boldsymbol{0_{3\times3}} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0_{3\times4}} \\ \boldsymbol{M_b^{-1} J_b^T} \end{bmatrix}$$

Where:

- M_b is the inertia of the base.
- J_b is the Jacobian relating body velocities to individual wheel velocities, accounting for the Mecanum wheel configuration.

- The system is fully observed, with the output matrix $C = I_6$, meaning all states are assumed measurable. [31]

6.2.6. Control Design

1. **Pole Placement for Controller Gain K:**
   The controller gain matrix K is computed to place the closed-loop poles of the system at desired locations in the complex plane. These poles are selected to ensure:
   - Stability (all poles in the left half-plane),
   - Adequate settling time,
   - Well-damped response with limited overshoot.
2. **Observer Gain L:**
   A Luenberger observer is designed to estimate the states using the measured output y and the control input u:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A\hat{x} + Bu + L(y - C\hat{x})}$$

The observer poles are chosen to be faster than the controller poles to guarantee rapid convergence of the estimated state $\hat{x}$ to the true state x.

6.2.7. Reference Trajectory Tracking

The desired reference trajectory x_{ref} is obtained from the desired wheel angular velocities $\dot{\theta}_{wheels,ref}$ using the inverse Jacobian relationship:

$$\boldsymbol{v_{ref} = J_b^{-1}\dot{\theta}_{wheels,ref} \implies x_{ref}(t) = \begin{bmatrix} x_{ref}(t) \\ y_{ref}(t) \\ \theta_{ref}(t) \\ v_{x,ref(t)} \\ v_{y,ref}(t) \\ \omega_{z,ref}(t) \end{bmatrix}}$$

Where $x_{ref}(t)$ and $y_{ref}(t)$ represent the desired position, $\theta_{ref}(t)$ the desired orientation, and the velocity terms are derived from inverse kinematics or differentiation.

The control input u (wheel velocity commands) is computed via full-state feedback from the estimated state $\hat{x}$:

$$\boldsymbol{u = -K(\hat{x} - x_{ref})}$$

ensuring the base accurately tracks the desired motion. [36]

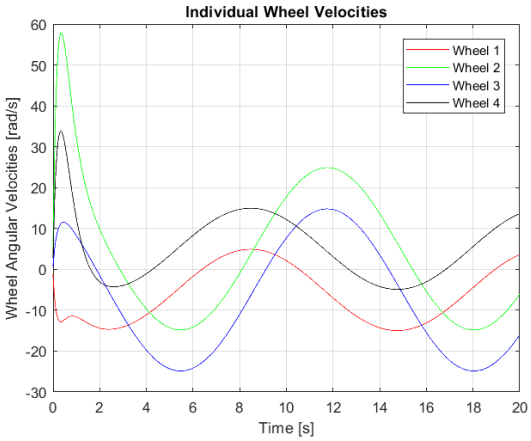


Figure 31: Individual Wheel Velocities

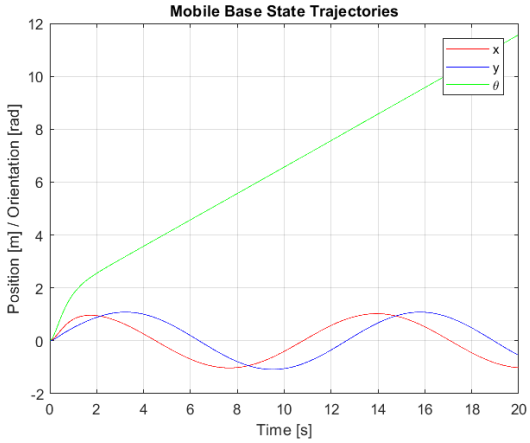


Figure 30: Mobile Base State Trajectories

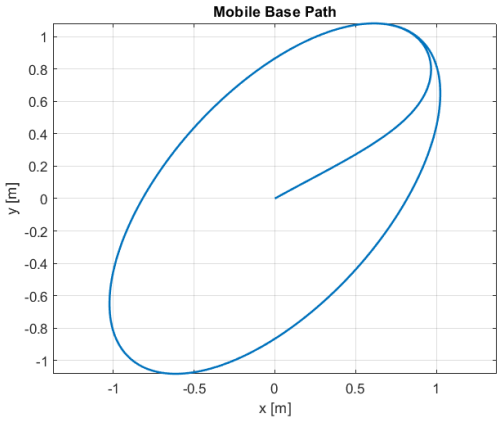


Figure 33: Mobile Base Path

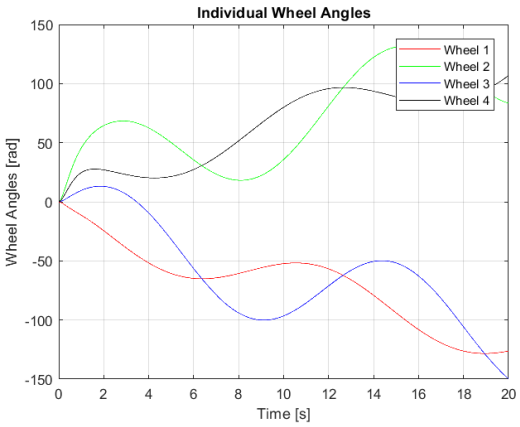


Figure 32: Individual Wheel Angles

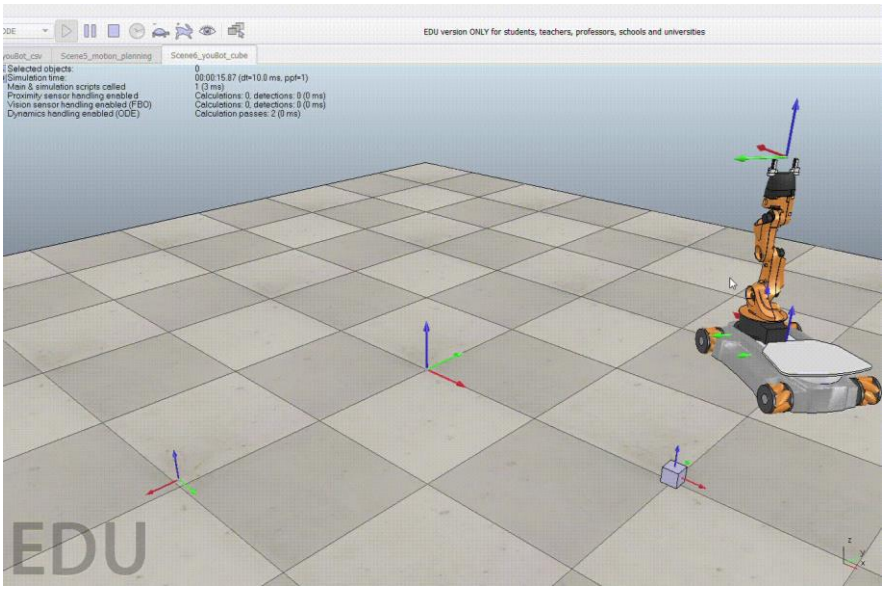


Figure 34: Simulation of Circular Trajectory Tracking for the Mobile Base Using Pole Placement and Observer-Based Control

7. Simulation

7.1. Simulation using CoppeliaSim and MATLAB

CoppeliaSim is used in this project as a dynamic simulator to validate an end-to-end robotic planning and control pipeline on the youBot mobile manipulator, a hybrid robot consisting of a four Mecanum wheels mobile base and a 5-joint serial manipulator. The robot must carry a cube from a known initial state to a specified goal state. To do this, MATLAB-oriented software generates a CSV trajectory file indicating the robot configuration over time, and this is played out in the physics-enabled CoppeliaSim environment.

One line of the CSV file is produced for each 0.01 second (10 millisecond) timestep with 13 comma-separated values:

```
chassis phi, chassis x, chassis y, J1, J2, J3, J4, J5,
W1, W2, W3, W4, gripper state
```

Where:

- **Chassis phi, x, y**: base planar orientation and position.
- **J1 to J5**: joint angles of the robotic arm.
- **W1 to W4**: angular positions of the four Mecanum wheels.
- **Gripper state**: 0 for open, and 1 for closed.

7.1.1. Constraints in Trajectory Generation

The MATLAB-generated trajectory is not arbitrary; it must comply with a well-defined set of constraints. Thus, the motion planning process can be conceived as a **Constraint Satisfaction Problem (CSP)**. The principal classes of constraints are:

1. **Temporal Constraints**

   The gripper takes 0.625 seconds (63 steps) to fully open or close. This is why, during the initiation of a grasp or release, the gripper's state must remain unchanged for at least 63 consecutive steps to allow the action to complete.

2. **Geometric and Kinematic Constraints**

   The chassis must follow the kinematic model of a Mecanum wheel base supporting three degrees of freedom: longitudinal (forward/backward), lateral (sideways), and in-place rotation (θ). The planner must convert the desired chassis motion to equivalent wheel speeds and adjust the wheel angles accordingly. The robotic arm must respect joint limits and avoid self-

collision or kinematic singularities while planning object grasping and placement motion.

3. **Physical Interaction Constraints**

   The gripper must be properly placed and oriented relative to the cube prior to closing in order to successfully establish the grasp. All motion must be dynamically feasible—smooth and continuous—to prevent the cube from sliding or falling due to jerky acceleration. The paths must be jerk-free and non-discontinuous motion that would provide unrealistic response from the physics engine.

4. **Task Constraints**

   The robot must execute an eight-segment planned trajectory sequence:

   1. Transition to a standoff pose above the block.
   2. Drop vertically to the grasp pose.
   3. Close the gripper and hold (0.625 seconds).
   4. Return to the standoff pose.
   5. Transition to a standoff pose above the goal.
   6. Drop to the placement pose.
   7. Open the gripper and hold (0.625 seconds).
   8. Return to the final standoff pose.

5. **Initial Condition Constraints**

   The robot's starting configuration may differ from the initial reference pose. Therefore, the control system must correct for any initial tracking error before entering the grasping phase. These constraints are collectively enforced using a planning and control pipeline written in MATLAB, employing time-scaling methods and trajectory generation strategies (e.g., Cartesian interpolation, screw motion) to create reference paths. These reference paths are then tracked using a simulated control loop.

7.1.2. CoppeliaSim Integration

The produced CSV path is emulated in Scene6 of CoppeliaSim using the native physics engine ODE, which can support realistic physics simulation like friction, contact dynamics, and inertia. The simulation phase is not only used to verify the geometric correctness of the planned motion but also verify the physical correctness of robot–object interaction—whether the cube is grasped, translated, and set down correctly. By integrating symbolic planning with dynamic simulation, the proposed pipeline bridges the gap between low-level motion control and high-level task description and obtains a fully automated and physically realizability-assured mobile manipulation solution. [34]

7.2. Design and ROS-Based Visualization of a Mobile Manipulator Using RViz and Xacro in ROS1 Noetic

The design and simulation processes were carried out using multiple tools. The mechanical structure was modeled using **CATIA V5**, while the control and simulation environment was implemented using **ROS (Robot Operating System)**. We used **RViz** for visualizing the robot's state and motion during task execution.

7.2.1. Robotic Arm Design

The robotic arm in this project was designed as a 5-DOF manipulator, consisting of five revolute links and an end-effector. The structure is defined using Xacro files to allow for modular and scalable robot descriptions.

Two main Xacro files were used:

- **variables.xacro**: Contains the standard specifications and dimensions of each component in the arm. This file stores constants such as link lengths, joint limits, and visual/collision parameters, making the model easy to maintain and update.
- **arm.xacro**: Defines the full arm assembly including all links and joints, and concludes with the end-effector, which is used for interacting with objects in the environment.

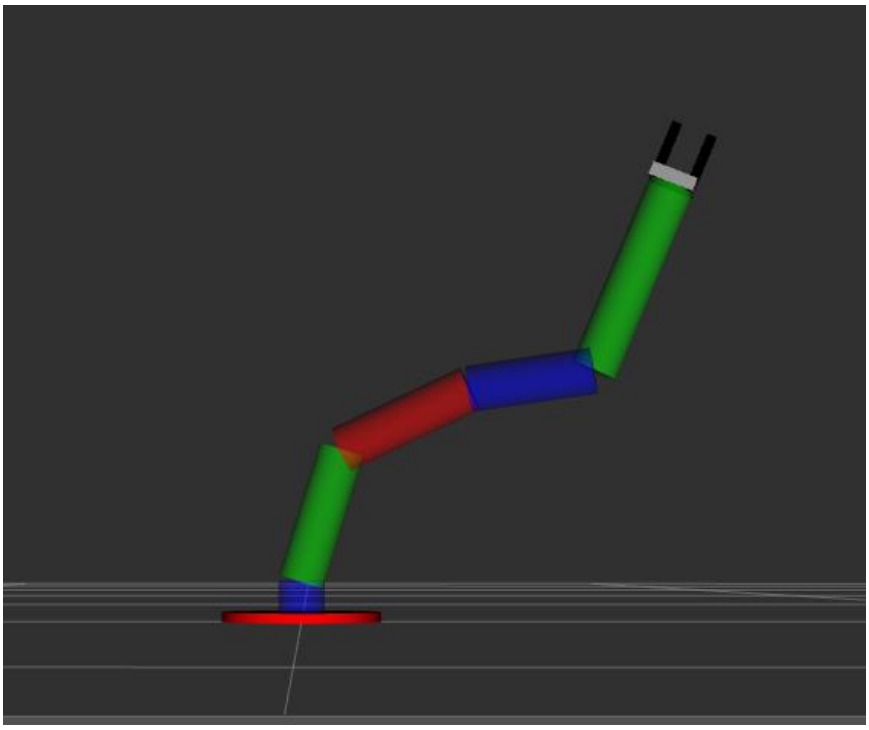


Figure 35: Rendered image of the full 5-DOF robotic arm (from RViz)

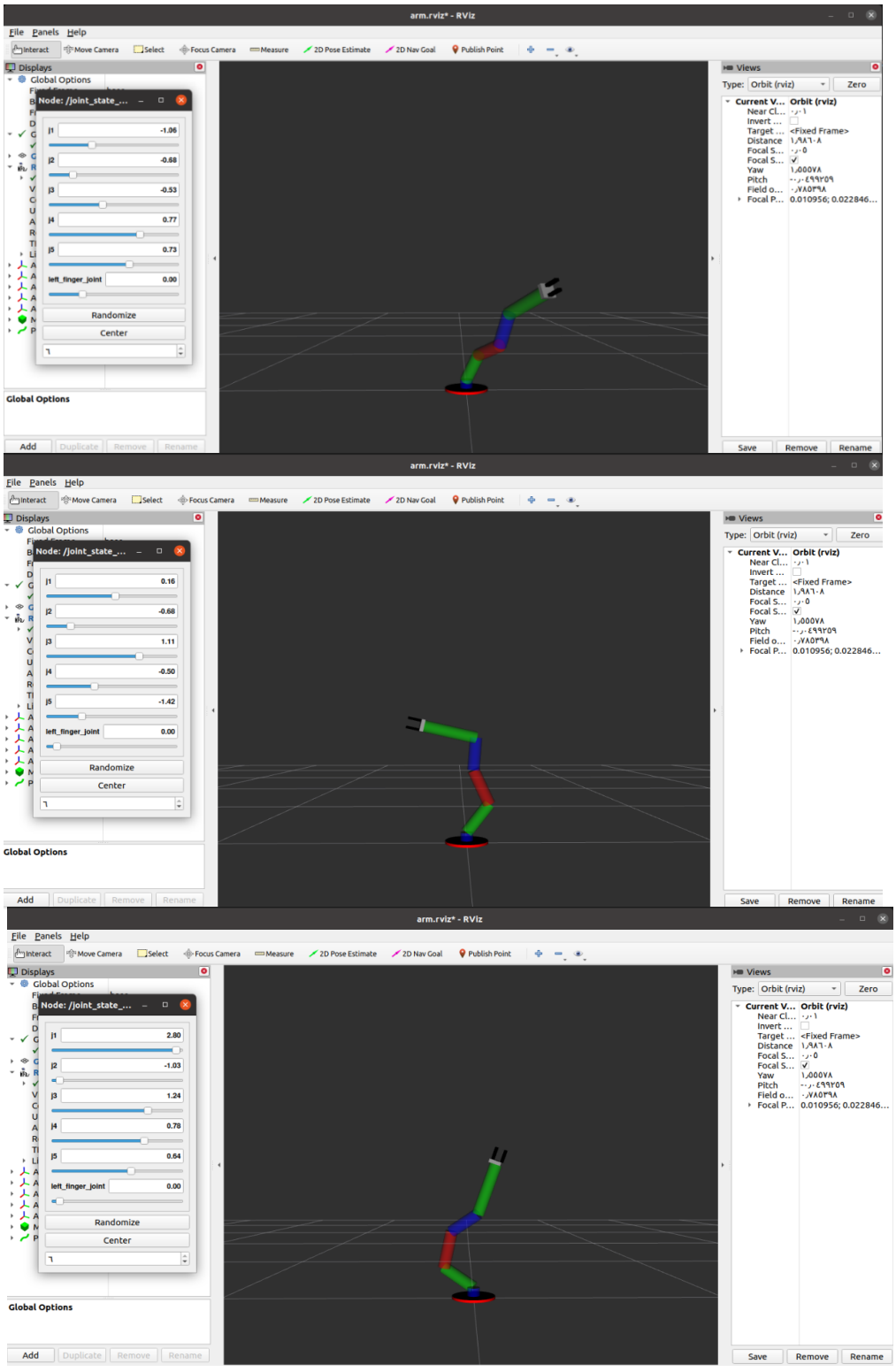


Figure 36: Multiple screenshots showing different joint configurations using joint_state_publisher_gui

To visualize and monitor the arm's motion, a joint_state_publisher_gui was used, enabling manual control of the joints for testing and demonstration. This allows real-time feedback through RViz as the arm moves through different configurations.

7.2.1.1. End Effector Pose Tracking

To track the end-effector's position in space, a ROS node named **end_effector_pose.py** was developed. This node uses the **TF (transform)** system in ROS to determine the position of the **gripper base (/gripper_base)** relative to the robot's **main base (/base)** frame.

The TF system maintains a tree of coordinate frames and allows real-time transformation lookup between them. In our case, the node uses tf.TransformListener() to **query the latest transformation between the base and the gripper**, extracting the translation component (x, y, z) which represents the current 3D position of the end-effector in space.

This position is then published to the /end_pose topic using a geometry_msgs/Point message. The node also logs the values for monitoring or debugging purposes.

Main functionalities:

- Subscribes to the TF tree (listens to transformations in real-time)
- Looks up the transform from /base to /gripper_base
- Extracts the position (x, y, z) and publishes it to /end_pose
- Continuously logs the gripper's position for visualization and debugging

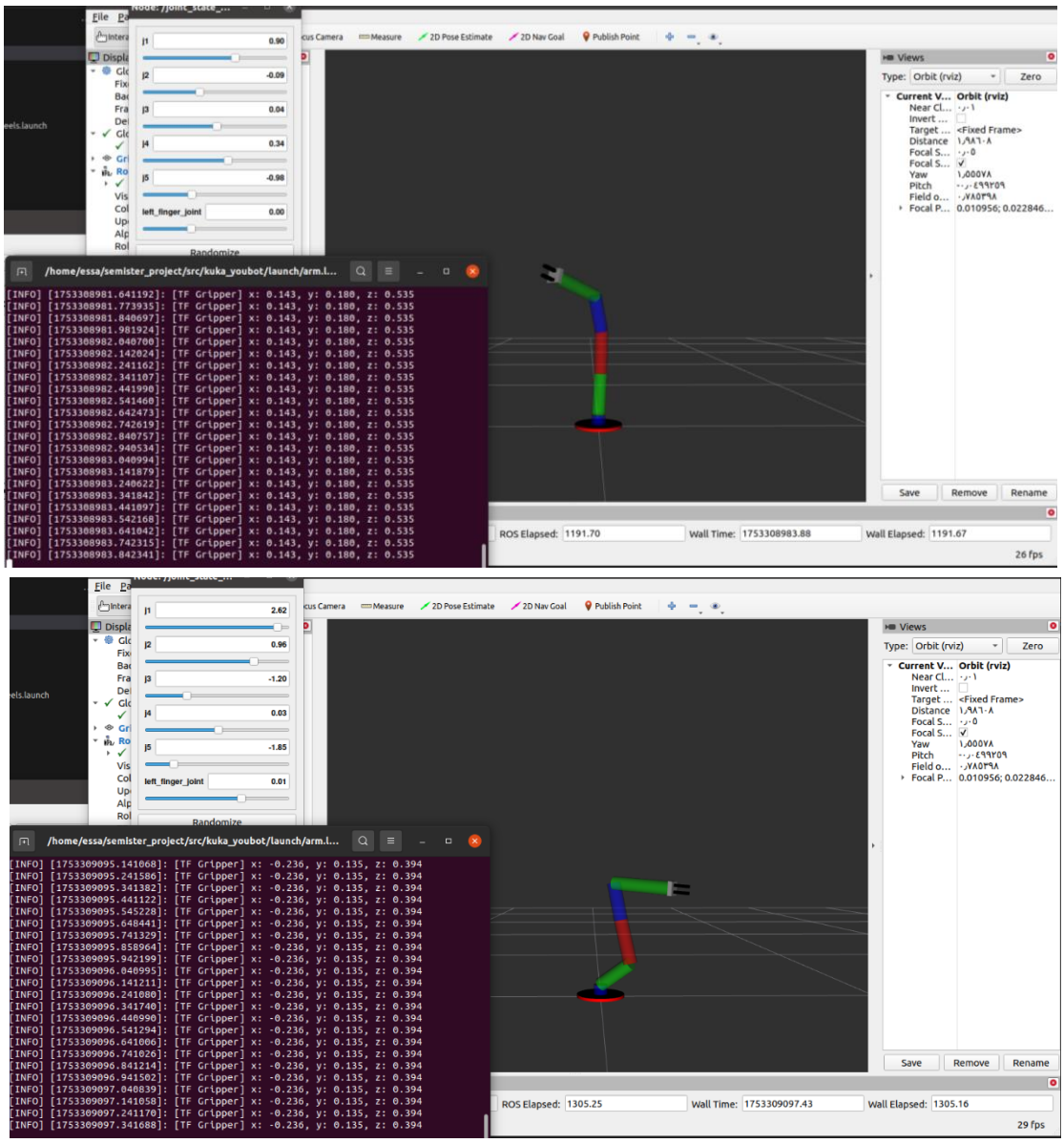


Figure 37: Image of the robotic arm with TF coordinate frames visible (e.g., /base, /gripper_base)

7.2.1.2. End Effector Marker Visualization

Another node, **marker.py**, subscribes to the /end_pose topic and visualizes the end-effector's position using an RViz marker (a sphere). This helps in identifying the exact pose of the gripper in 3D space during motion, providing visual feedback for trajectory validation.

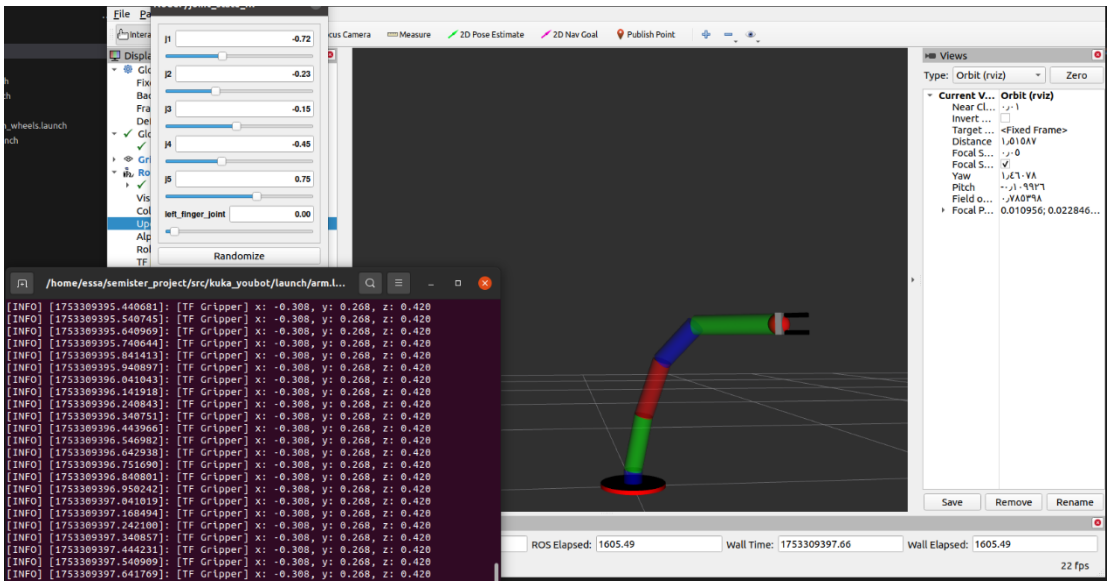


Figure 38: RViz screenshot showing the marker (sphere) at the end-effector position

7.2.1.3. Trajectory Visualization

To visualize the trajectory of the end-effector during motion, a dedicated ROS node named **path.py** was implemented. This node creates a continuous visual trail in **RViz**, showing the positions that the gripper has passed through over time. This is especially useful when validating both **planned paths** and **manually guided movements**, as it provides immediate visual feedback in task space.

The node works by subscribing to the **/end_pose** topic, which carries the live 3D coordinates of the end-effector (published by **end_effector_pose.py**). Every time a new point is received, the node:

1. Creates a new **PoseStamped** message representing that position.
2. Adds it to a Path message (from **nav_msgs/Path**).
3. Publishes the updated path to the topic **/end_effector_path**.

Over time, the published path grows into a **trajectory line**, which RViz can render as a **line strip** showing the full movement history of the gripper.

How the node works in detail:

- **self.pose_callback()** is triggered every time a new **Point** message arrives.
- It wraps the point into a **PoseStamped** message (needed by RViz and **Path**).
- The new pose is appended to a **Path** object that accumulates all positions.
- The path is continuously published, creating a live trail in RViz.

7.2.1.4. Node Communication Structure (rqt_graph)

To better understand the communication between ROS nodes and the data flow within the system, the **rqt_graph** tool was used. It visually represents the connections between the nodes such as:

- **/end_effector_tf_reader** subscribing to TF and publishing to **/end_pose**
- **/marker** and **/path** subscribing to **/end_pose** and publishing RViz markers
- **/joint_state_publisher_gui** publishing joint states

This graph helps debug and validate the overall architecture of the robotic arm system.

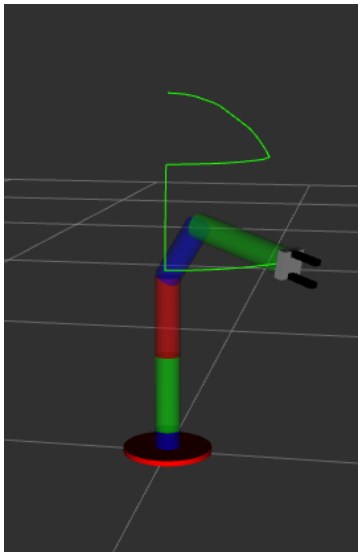


Figure 39: RViz screenshot showing the trajectory path of the end-effector.

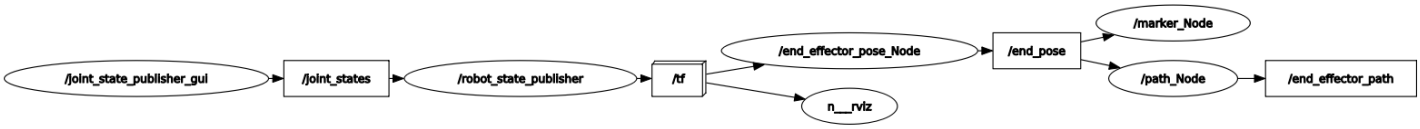


Figure 40: rqt_graph showing communication between end_effector_pose.py, marker, path, and joint_state_publisher

7.2.2. Mobile Base Design

The mobile base of the robot was designed using **Mecanum wheels**, enabling omnidirectional movement — forward, sideways, diagonal, and rotational — which provides high maneuverability in static environments. This feature is essential for executing accurate and flexible pick-and-place tasks.

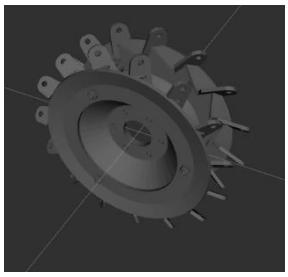


Figure 41: Rendered view of a single Mecanum wheel (from CATIA)

The base was structured using modular **Xacro** files as follows:

- **variables.xacro**: Defines all shared physical parameters such as wheel radius, chassis dimensions, and relative positions. These values ensure consistency across the different robot components.

- **single_wheel.xacro**: Describes the geometry and physical properties of a single Mecanum wheel, without its rollers. The roller geometry is generated automatically when the wheel is instantiated, making the wheel ready for visual and collision simulation.

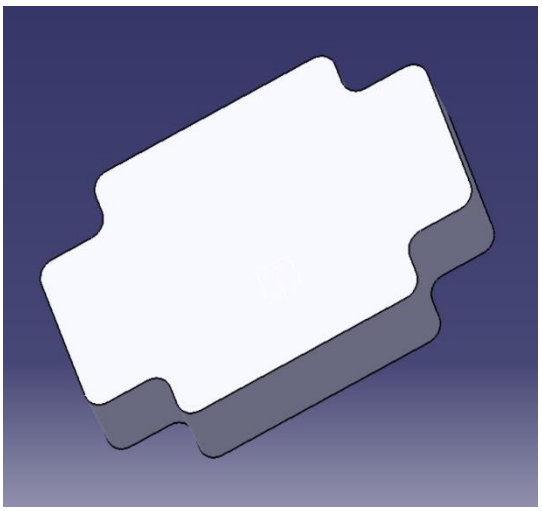


Figure 42: Base chassis alone — from CATIA

- **base.xacro**: This is the central file responsible for assembling the mobile base. It includes the chassis body and instantiates the four Mecanum wheels at their correct positions. The frame was originally designed using **CATIA V5**, then translated into a URDF-compatible format using this Xacro file. This allowed for seamless simulation and integration in ROS.

Finally, the complete mobile base was loaded into RViz and prepared for integration with the robotic arm.

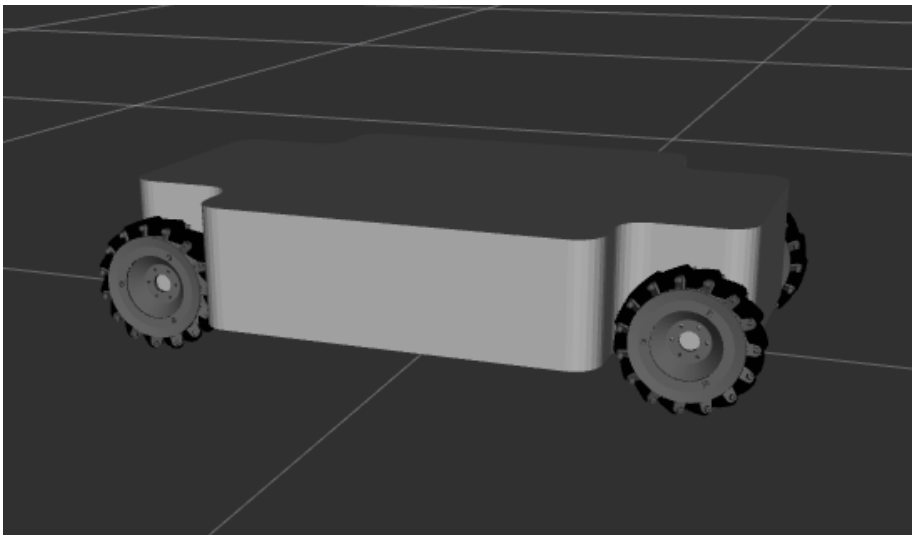


Figure 45: Full base image showing chassis + four Mecanum wheels (from RViz)

7.2.2.1. Node Communication Structure (rqt_graph)

To understand how the mobile base components communicate within the ROS environment, we used **rqt_graph** to visualize the system's structure. This tool displays the interaction between ROS nodes, topics, and services.

In this part of the system, **rqt_graph** shows:

- The **robot_state_publisher** broadcasting joint and TF transforms.
- The **joint_state_publisher_gui** providing joint values (in case manual testing of wheel joints was done).
- Any additional topics related to **wheel controllers** or TF frames for the base.

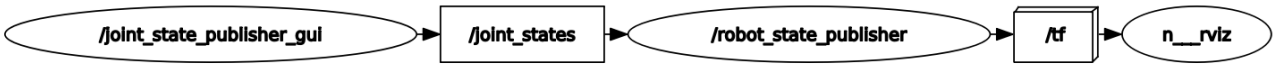


Figure 48: rqt_graph showing only the nodes and topics related to the mobile base

7.2.3. Integration of Base and Arm

After designing both the robotic arm and the mobile base separately, the two components were integrated into a single unified robot using ROS Xacro. The arm was **mounted onto the mobile base** by defining its position relative to the base frame using a fixed transform.

This was achieved through the following coordinate definition:

xyz="0 ${shifted_base_to_arm} ${base_box_height + base_clearance}"

This configuration aligns the bottom of the arm with the top of the chassis, ensuring correct positioning both visually and physically in the simulation environment. The transform accounts for vertical spacing and mechanical alignment.

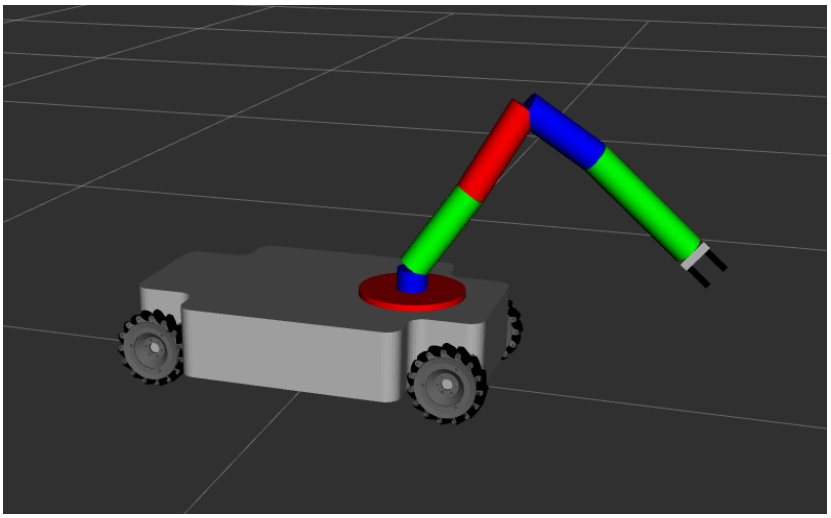


Figure 49: full robot assembled

All previously developed nodes (for joint publishing, end-effector tracking, marker visualization, and trajectory plotting) were combined and launched together in this integrated setup. This enabled full system functionality, allowing the arm to move while mounted on the base, and still compute its end-effector position relative to the base frame in real time.

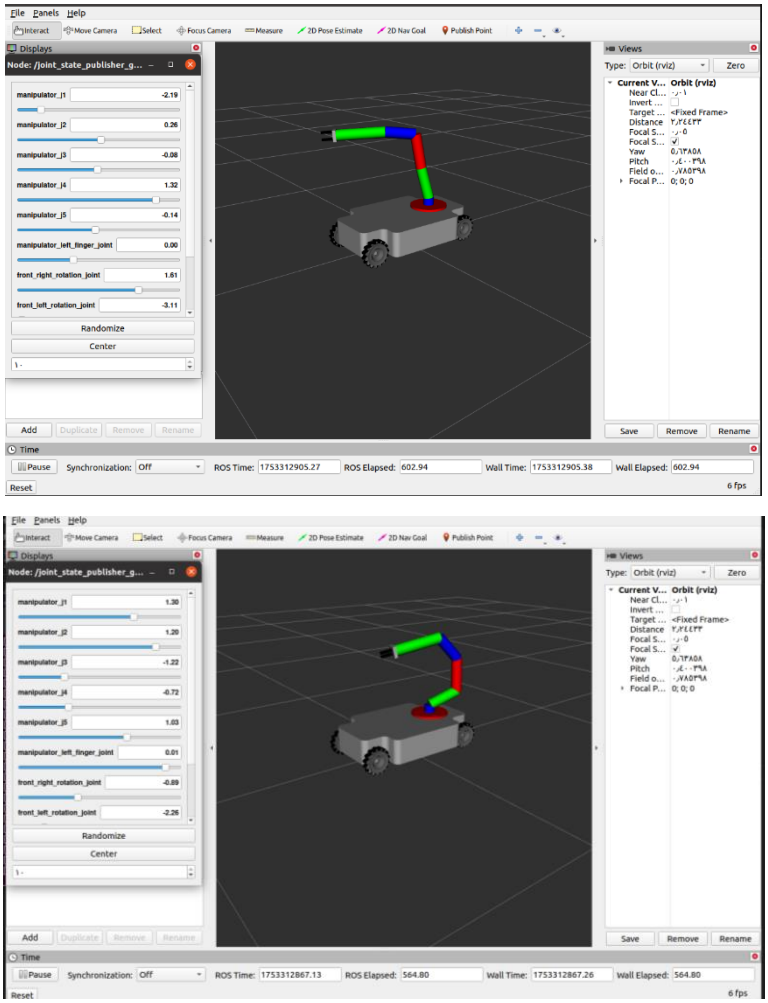


Figure 52: images showing the arm moving while mounted on the base using the joint_state_publisher_gui

As the arm moves, the same TF-based node continues to track the end-effector's 3D position relative to the base. This means that even when integrated, the system accurately computes the pose of the gripper in task space and updates the RViz marker and path visualization accordingly.

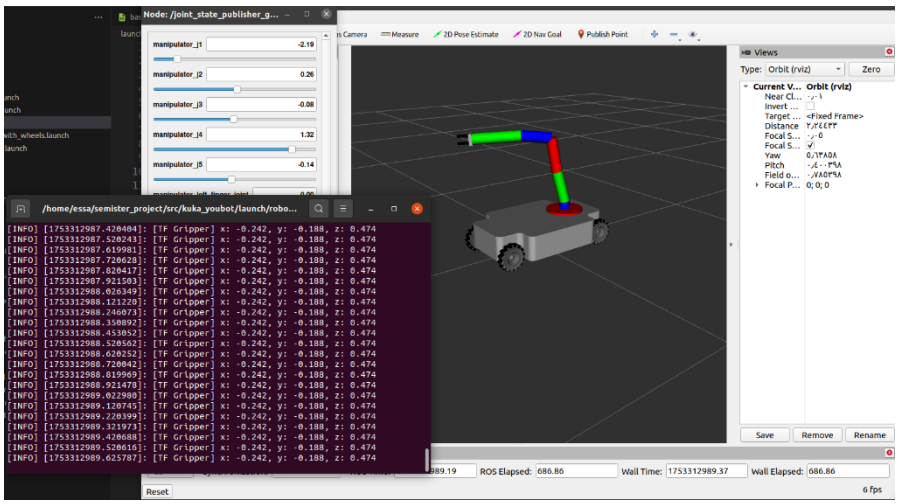


Figure 53: RViz screenshot showing the end-effector position while mounted on the base

7.2.4. System Launch and Configuration

To efficiently initialize the entire robot system with all its functional components, we created a unified **ROS launch file** that brings up the complete simulation with a single command. This file handles the robot's description, visualization, joint control, and all custom ROS nodes.

Below is an overview of the key elements in the launch file:

- **robot_description parameter**: Loads the robot's full URDF model from **robot.xacro**, which combines both the arm and the mobile base.
- **robot_state_publisher**: Publishes the TF transforms between all links of the robot, based on the current joint states.
- **joint_state_publisher_gui**: Provides a graphical interface to manually control the joint positions for testing the arm's movement.
- **rviz**: Launches the RViz environment with a predefined configuration file (**robot2.rviz**), which includes visualizations such as the robot model, TF frames, markers, and trajectory.
- **Custom Nodes**:
  - **end_effector_pose.py**: Computes and publishes the end-effector's position using TF.
  - **marker.py**: Visualizes the gripper position as a sphere in RViz.
  - **path.py**: Continuously updates the trajectory path of the end-effector.
- **rqt_graph**: Automatically launches the graphical tool to visualize all node-topic interactions, providing a live overview of the system structure.

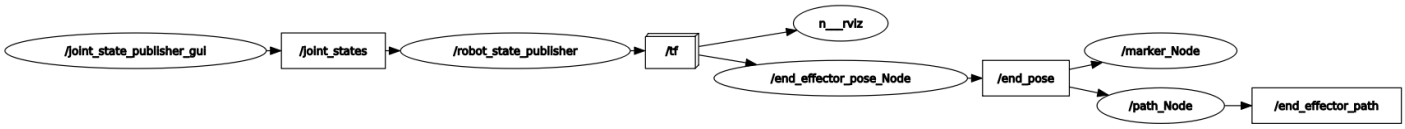


Figure 54: rqt graph for full system

This centralized launch configuration significantly reduces the setup time and ensures consistent integration between all components. It also facilitates testing, debugging, and presentation of the mobile manipulator system as a cohesive whole.

8. Conclusions, Results, and Future Work

8.1. Conclusion

The project demonstrated successful development of a complete simulation platform for a mobile manipulation system using the KUKA youBot. Merging a 5-DOF robotic arm with a Mecanum-wheeled mobile base, the system was able to perform precise pick-and-place movements in cluttered scenarios. The use of an observer-based state feedback controller via pole placement guaranteed stable dynamic response and accurate tracking of trajectories. The project effectively combined kinematic modeling, control design, and simulation to obtain a scalable and well-coordinated robot architecture.

Having two independent simulation environments—CoppeliaSim for dynamic interaction and ROS1 Noetic with RViz for geometric model verification—allowed both functional and structural correctness of the platform to be verified. This dual approach increased confidence in control strategies and robot configuration implemented. The integration of CATIA V5 for precise CAD modeling contributed to the physical realism of the simulated robot, strengthening the accuracy of URDF-based visualization and future integration with ROS-based motion planning tools.

8.2. Results

- The simulation conducted using the Scene6_youBot_Cube in CoppeliaSim demonstrated the feasibility of both the mobile base and the manipulator being unified under a coordinated motion paradigm. The robot was successfully able to execute pick-and-place routines that incorporated simultaneous motion between the chassis and the 5-DOF manipulator. The motion was driven using pre-calculated CSV files with detailed trajectory

information, and the simulated system maintained structural coherence at runtime. There were no joint limit exceedances during the trajectory execution.

- The simulation setup allowed for testing of a dynamic scene where the youBot grasped a physical cube, validating that the gripper can close around the object and conduct object transportation in a controlled setting. The use of the pre-determined scene allowed for implementing control and trajectory logic without having to recreate the robot model manually.
- Apart from this, a static RViz simulation under ROS1 Noetic confirmed the accuracy of the robot's physical geometry, including joints, links, and wheel configuration, to be accurate. Although motion was not simulated under ROS, RViz visualization ensured alignment with the URDF/XACRO model and assumed actual geometry under the CoppeliaSim environment.
- The mechanical structure of the mobile base was originally modeled using CATIA V5, and its dimensions were then transferred into URDF/XACRO format. This ensured geometric consistency between the CAD design and the simulation environments in ROS1 Noetic and CoppeliaSim.

8.3. Future Work

Future growth of the project might concentrate on a number of principal areas:

- Dynamic simulation within ROS: Integrate the robot model into a ROS-Gazebo pipeline for full dynamic simulation within the ROS environment, where it will then open up the possibility for real-time trajectory planning and sensor-based feedback.
- Motion execution within RViz/ROS: Extend the RViz setup beyond static visualization to really perform motion control by using ROS topics and services--for example, by using the joint_trajectory_controller-to make the system more interactive and testable within ROS.
- Sensor Integration: Incorporating external sensors (e.g., cameras or force sensors) for real-time environment perception and feedback-based manipulation could improve robustness in unstructured scenarios.
- Grasp Validation and Metrics: Adding tools to evaluate grasp quality, success rate, and object interaction would provide a quantitative basis for improving control and design strategies.
- Real-World Deployment: The ultimate extension would be deploying the control system on a physical youBot robot, validating simulation findings in real-world applications and closing the sim-to-real gap.

9. References

[1] H. Engemann, S. Du, S. Kallweit, P. Cönen and H. Dawar, "OMNIVIL—An autonomous mobile manipulator for flexible production," *Sensors,* vol. 20, no. 24, p. 7249, 2020.

[2] S. Datta, R. Ray and D. Banerji, "Development of autonomous mobile robot with manipulator for manufacturing environment," *The International Journal of Advanced Manufacturing Technology,* vol. 38, no. 5–6, p. 536–542, 2007.

[3] C. C. Kemp, A. Edsinger, H. M. Clever and B. Matulevich, "The design of Stretch: A compact, lightweight mobile manipulator for indoor human environments," in *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA, 2022.

[4] R. Oftadeh, J. Taghirad, A. Saber and A. Rezaee, "System Integration for Real-time Mobile Manipulation," *International Journal of Advanced Robotic Systems,* vol. 11, no. 1, p. 51, 2014.

[5] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal,* vol. 6, no. 1, p. 4, 2019.

[6] M. Ciocarlie, K. Hsiao, A. Leeper and D. Gossow, "Mobile manipulation through an assistive home robot," in *IEEE*, Vilamora-Algarva, Portugal, 2012.

[7] J. González Huarte and A. Ibarguren, "Visual Servoing Architecture of Mobile Manipulators for Precise Industrial Operations on Moving Objects," *Robotics,* p. 71, 2024.

[8] Z. He, X. Zhang, S. Jones, S. Hauert, D. Zhang and N. F. Lepora, "TacMMs: Tactile Mobile Manipulators for Warehouse Automation," *IEEE Robotics and Automation Letters,* vol. 8, no. 5, p. 4729–4736, October 2023.

[9] R. Holmberg and O. Khatib, "Development and control of a holonomic mobile robot for mobile manipulation tasks," *The International Journal of Robotics Research,* vol. 19, no. 11, p. 1066–1074, November 2000.

[10] J. Xu, Y. Domae, T. Ueshiba, W. Wan and K. Harada, "Planning a Sequence of Base Positions for a Mobile Manipulator to Perform Multiple Pick-and-Place Tasks," Cornell University Library, Ithaca, NY, USA, 2020.

[11] J. Haviland, N. Sünderhauf and P. Corke, "A Holistic Approach to Reactive Mobile Manipulation," *IEEE Robotics and Automation Letters,* vol. 7, no. 2, pp. 3122-3129, 2022.

[12] S. Yan, Z. Zhang, M. Han, Z. Wang, Q. Xie, Z. Li, Z. Li, H. Liu, X. Wang and S.-C. Zhu, "M2Diffuser: Diffusion-based Trajectory Optimization for Mobile Manipulation in 3D Scenes," 15 October 2024. [Online]. Available: https://arxiv.org/abs/2410.11402.

[13] J. Wu, W. Chong, R. Holmberg, A. Prasad, Y. Gao, O. S. S. Khatib, S. Rusinkiewicz and J. Bohg, "TidyBot++: An Open-Source Holonomic Mobile Manipulator for Robot Learning," in *Proceedings of The 8th Conference on Robot Learning (CoRL 2024)*, Munich, 2025.

[14] K. He and e. al., "On-The-Go Robot-to-Human Handovers with a Mobile Manipulator," Cornell University Library, Ithaca, NY, USA, 2022.

[15] N. Ghodsian, K. Benfriha, A. Olabi, V. Gopinath and A. Arnou, "Mobile Manipulators in Industry 4.0: A Review of Developments for Industrial Applications," *Sensors,* vol. 23, p. 8026, 2023.

[16] R. Schneider, D. Honerkamp, T. Welschehold and A. Valada, "Task-Driven Co-Design of Mobile Manipulators," 21 December 2024. [Online]. Available: https://arxiv.org/abs/2412.16635.

[17] KUKA Laboratories GmbH, "KUKA youBot – Research & Application Development in Mobile Robotics," KUKA Laboratories GmbH, Datasheet, 2014.

[18] K. M. Lynch and F. C. Park, Modern Robotics: Mechanics, Planning, and Control, Cambridge: Cambridge University Press, 2017.

[19] J. Hernandez-Barragan, C. Lopez-Franco, N. Arana-Daniel and A. Y. Alanis, "Inverse kinematics for cooperative mobile manipulators based on differential evolution," *PeerJ Computer Science,* vol. 7, p. e419, 2021.

[20] J. Hernandez-Barragan, J. Plascencia-Lopez, M. Lopez-Franco, N. Arana-Daniel and C. Lopez-Franco, "Inverse kinematics of robotic manipulators based on hybrid differential evolution and Jacobian pseudoinverse approach," *Algorithms,* vol. 17, p. Art. no. 454, October 2024.

[21] Duke University, "Chapter 3 - Forward Kinematics: the Denavit-Hartenberg Convention," Duke University – cs.duke.edu, [Online]. Available: https://users.cs.duke.edu/~brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf. [Accessed 25 05 2025].

[22] J. Haviland and P. Corke, "Manipulator Differential Kinematics: Part I: Kinematics, Velocity, and Applications [Tutorial]," *IEEE Robotics & Automation Magazine,* vol. 31, no. 4, pp. 149 - 158, 2023.

[23] Automatic Addison, "The Ultimate Guide to Jacobian Matrices for Robotics," automaticaddison, 2020. [Online]. Available: https://automaticaddison.com/the-ultimate-guide-to-jacobian-matrices-for-robotics/. [Accessed 30 05 2025].

[24] RoboticsUnveiled, "Robotics Part 22 – Differential Forward Kinematics," RoboticsUnveiled, 2024. [Online]. Available: https://www.roboticsunveiled.com/robotics-differential-forward-kinematics/. [Accessed 30 05 2025].

[25] P. Corke, "Roll, Pitch, Yaw angles," Peter Corke Robotics and Vision, 26 November 2017. [Online]. Available: https://petercorke.com/robotics/roll-pitch-yaw-angles/. [Accessed 05 June 2025].

[26] Y. Zhang, Y. Li and X. Xiao, "A Novel Kinematics Analysis for a 5-DOF Manipulator Based on KUKA youBot," in *IEEE Conference on Robotics and Biomimetics (ROBIO)*, Zhuhai, China, 2015.

[27] RoboDK, "Robot Singularities: What They Are and How to Avoid Them," RoboDK Blog, 30 November 2022. [Online]. Available: https://robodk.com/blog/robot-singularities/. [Accessed 06 June 2025].

[28] Z. Zhang, J. Z. Zhang, S. Yang and Z. Manchester, "Robots with Attitude: Singularity-Free Quaternion-Based Model-Predictive Control for Agile Legged Robots," arXiv preprint, Evanston, 2024.

[29] J. J. Craig, Introduction to Robotics: Mechanics and Control, 3rd Ed. ed., Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2005.

[30] M. Spong, S. Hutchinson and M. Vidyasagar, Robot Modeling and Control, 1st Ed. ed., Hoboken, NJ, USA: Wiley, 2006.

[31] F. L. Lewis, D. M. Dawson and C. T. Abdallah, Robot Manipulator Control: Theory and Practice, 2nd Ed. ed., New York, NY, USA: Marcel Dekker, 2003.

[32] J. Hess, N. Engelhard, J. Sturm, D. Cremers and W. Burgard, "Manipulability Analysis," in *IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, USA, 2012.

[33] J. W. Burdick, "The Damped Pseudo Inverse," [Online]. Available: https://robotics.caltech.edu/~jwb/courses/ME115/handouts/damped.pdf. [Accessed 27 July 2025].

[34] N. U. M. Wiki, "Mobile Manipulation Capstone," Northwestern University, [Online]. Available: http://hades.mech.northwestern.edu/index.php/Mobile_Manipulation_Capstone. [Accessed 26 July 2025].

[35] B. Siciliano, L. Sciavicco, L. Villani and G. Oriolo, Robotics: Modelling, Planning and Control, London, UK: Springer, 2009.

[36] S. Sojoudi, "Lab 6b: Luenberger Observer Design for Inverted Pendulum," University of California, Berkeley, Berkeley, CA, 2017.

[37] N. S. Nise, Control System Engineering, 7th Edition ed., Hoboken, New Jersey: Wiley, 2015.

[38] F. Y. Xiang, "Robot Dynamics Lecture Notes," Zurich, Switzerland , ETH Zurich, 2017, pp. 1-50.