Inverse Kinematics in Gazebo without ROS using PyKDL

Tim Liehr
Fulda University of Applied Sciences
Fulda, Germany
tim.liehr@informatik.hs-fulda.de

***Abstract*—This project presents an implementation of inverse kinematics (IK) for a 7-DOF robotic arm in the Gazebo simulation environment using PyKDL, without relying on ROS. The Franka Emika Panda robot was modeled as a kinematic chain, and joint angles were computed for given end-effector target positions, considering joint limits and solver stability. The system successfully applied these inverse kinematics solutions to the robot in real time, achieving accurate positioning with minimal deviation. The results demonstrate a lightweight, ROS-free approach to solving inverse kinematics in simulation environments.**

I. Introduction

This research project report is in the context of robotic kinematics and inverse kinematics, which are fundamental topics in robotics and automation. Inverse kinematics is the process of determining the joint parameters needed for a robotic arm to reach a given target position and orientation. It is widely used in industrial automation, robotic manipulators, and humanoid robotics. Traditional approaches to IK include analytical solutions, numerical solvers, and optimization-based methods, each with its own advantages and challenges.

A major challenge in solving IK problems is handling joint constraints and ensuring feasible solutions within the mechanical limits of a robotic system.

A. Project overview

This project focuses on the implementation and evaluation of an IK solver applied to a simulated robotic arm in Gazebo. It further investigates how different solver configurations — such as the use of a weight matrix and initial joint positions — influence the success of finding a valid IK solution.

The inspiration for this work comes from the paper "Continual World: A Robotic Benchmark for Continual Reinforcement Learning." Although the paper primarily addresses the challenges of continual learning in reinforcement learning and evaluates robotic manipulation tasks using a low-dimensional Cartesian action space, it implicitly relies on inverse kinematics to convert end-effector commands into joint angles [6]. While the Continual World benchmark abstracts away the details of joint control by handling inverse kinematics internally, this project explores an explicit implementation of the IK process within a Gazebo simulation — without relying on ROS — using the PyKDL library.

Since the Continual World benchmark employs a 7-degree-of-freedom (7-DOF) manipulator — a robotic arm with seven independently rotatable joints that enable a high degree of flexibility and redundancy — a comparable setup is adopted in this work. Specifically, the Franka Emika Panda robot model provided by OpenRobotics [[3], [4]] is utilized. Figure 1 shows the robot arm within the Gazebo simulation environment.

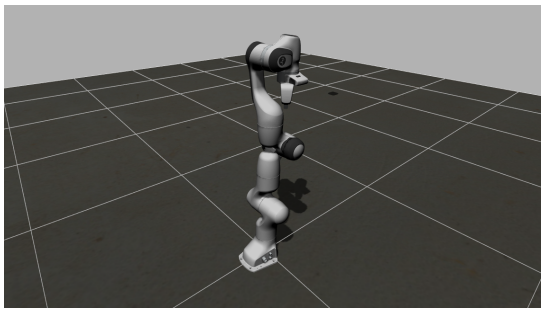

Fig. 1. Panda robot arm in Gazebo GUI.

The project is structured into three main steps (see Figure 2). First, the robot is mapped as a kinematic chain based on its SDF description using PyKDL, enabling access to joint and segment data for further kinematic computations. In the second step, PyKDL is used to solve the inverse kinematics problem for a given target pose of the end effector, determining the joint angles required to reach that pose. Finally, the calculated IK solution is applied to the robot model in Gazebo to move the arm and verify that the desired pose is achieved.

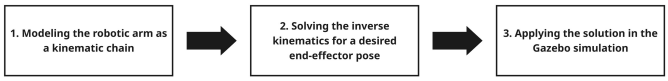

Fig. 2. Inverse Kinematics implementation steps

By focusing on IK in Gazebo, this work aims to bridge the gap between reinforcement learning approaches, which typically operate in a simplified action space, and classical robotic control, where precise joint manipulation via inverse kinematics is essential.

B. Related work

Robotic kinematics — particularly the modeling of forward and inverse kinematics — is a fundamental concept in robot control. As described by Kucuk and Bingul [2], inverse

kinematics poses a significantly more complex computational challenge than forward kinematics, especially for robots with higher degrees of freedom and coupled kinematic chains. Two major solution strategies exist: analytical and numerical approaches. While analytical methods provide exact solutions, they are often only feasible for robots with relatively simple structures. Numerical solvers, in contrast, are more flexible and can be applied to general robot configurations but require iterative algorithms.

Orocos KDL (Kinematics and Dynamics Library) provides a framework for solving both forward and inverse kinematics. Vaish provides an accessible explanation of how to construct forward kinematics models using KDL. [5].

The use of Gazebo as a physics-based simulation environment is well documented in robotic research. Murat Aksu, John L. Michaloski and Frederick M. Proctor present a framework for simulating industrial robotic systems in Gazebo, highlighting its role in developing and validating robot control logic before deployment on physical hardware [1]. Their work also emphasizes the integration with ROS and physics-based realism.

In contrast to previous studies that focus either on the mathematical foundations of inverse kinematics or on general simulation environments, this work aims to bridge both aspects by explicitly implementing and evaluating inverse kinematics with PyKDL in a Gazebo-based setup, entirely without ROS.

C. Goals

The main objectives of this research project are as follows:

- Get all relevant data of the robot arm from the Gazebo simulation: Joint limits and current joint angles and end-effector position
- An IK solution is to be calculated for a valid target position
- Ensure that the solutions comply with the joint limits of the robot arm
- Ensure that if the specified target pose is unreachable, no invalid solution is returned and a corresponding error message is provided.
- Sending the calculated IK solution to the robot arm joints to move the robot arm to the target position in the Gazebo simulation

These goals form the foundation for the implementation of an inverse kinematics pipeline in Gazebo. To represent the robotic arm in Python and validate the computed IK solutions, it is necessary to retrieve relevant data from the simulation: specifically, the current joint angles and the position of the end effector.

The core step involves calculating a valid IK solution for a given, reachable target pose. During this process, the joint limits of the robot must be respected, as the joints do not allow full 360-degree rotation, but instead operate within defined ranges. If the specified target pose is unreachable, the system should provide an appropriate message and refrain from returning an invalid solution.

To apply a calculated IK solution, the robotic arm must be controllable within the simulation. For this purpose, the joints are addressed via a Python script, allowing the computed joint angles to be sent directly to the robot model in Gazebo to execute the corresponding movement.

II. Method Overview

A. Gazebo

Gazebo is a physics-based 3D simulation environment widely used in robotics research and development. It provides accurate simulation of rigid body dynamics, collision detection, and sensor feedback, allowing for the testing and validation of robotic systems in a virtual environment prior to real-world deployment. Robot models are integrated using SDF (Simulation Description Format) files, which define their physical and kinematic properties.

In this project, Gazebo is used to simulate the Franka Emika Panda robot arm. The Gazebo simulation provides real-time data on the current status of the simulation world, including the current state of the robot, which can be accessed by subscribing to specific communication channels known as topics. Topics in Gazebo follow a publish–subscribe model, where simulation components broadcast information (e.g., poses, velocities, joint states), and clients can subscribe to receive that data.

The command `gz topic -l` lists all available topics in the simulation. For instance, the topic `/world/test-world/dynamic_pose/info` publishes dynamic pose information of all simulated entities. Of particular relevance is the position of the robot hand (end effector) in 3D space. Additionally, by integrating the `joint_state_publisher` plugin into the robot's SDF file, joint-specific data — including the current joint angles (in radians) and the respective joint limits — can be accessed via the topic `/model/panda_arm/joints/state`.

This data serves as the input for the inverse kinematics pipeline implemented with PyKDL.

B. Orocos KDL and PyKDL

Orocos KDL (Kinematics and Dynamics Library) is a C++ library developed as part of the Orocos project, which focuses on real-time control software for robotic systems. KDL provides essential tools for modeling robotic kinematics and dynamics, including forward and inverse kinematics, and is particularly well-suited for robotic arms with serial kinematic chains.

PyKDL is the official Python wrapper for Orocos KDL, enabling access to its functionality through Python scripts. This allows users to create kinematic chains, perform kinematic computations, and integrate these calculations into simulation or control environments more conveniently than using the C++ library directly.

PyKDL is maintained in synchrony with Orocos KDL. The latest version, v1.5.1, was released on September 12, 2021. While further development and a new release remain uncertain, the library is actively maintained.

C. Representing the Robot in Python

To interact with the Gazebo simulation and to access real-time data required for inverse kinematics calculations, a custom Python class `PandaRobot` is implemented. This class serves as an interface to represent the current state of the robot within Python.

Upon initialization, the class subscribes to relevant Gazebo topics to continuously retrieve joint states and the position of the end-effector (see II-A). This data is stored in a structured format, making it accessible for tasks such as constructing the kinematic chain, computing inverse kinematics, and validating IK results. It also includes functionality to control the robot by sending rotation commands to individual joints using the corresponding topic.

By encapsulating both the state retrieval and control logic, the PandaRobot class forms the central component for linking simulation data, kinematic modeling, and motion execution.

D. Creating Kinematics Chain

The first step of the inverse kinematics implementation involves representing the 7-degree-of-freedom (7-DOF) Panda robot arm as a kinematic chain. In PyKDL, a kinematic chain consists of multiple segments, each representing a link of the robot. Each segment consists of a joint and a frame: the joint defines the type and axis of motion (typically rotational around a specific axis), while the frame describes the spatial transformation relative to the preceding segment.

To ensure consistency and alignment with standard robotics conventions, the widely used Denavit–Hartenberg (DH) convention is applied, whereby the Z-axis is aligned with the axis of joint rotation.

The kinematic chain is created manually in Python using the `create_chain()` method, in which each segment is added sequentially according to the robot's physical structure and joint configuration. The construction includes all seven joints of the Panda arm, as well as an additional segment representing the end-effector. All transformations and dimensions used are derived from the robot's SDF file to ensure compatibility with the Gazebo simulation model.

This definition allows PyKDL to construct an internal representation of the kinematic structure, which serves as the basis for the subsequent inverse kinematics computation.

E. Calculation of Inverse Kinematics

The inverse kinematics solution is computed using the method `compute_inverse_kinematics(kdl_chain, target_position)`. This function takes the previously defined kinematic chain and a desired end-effector position as input and returns the corresponding joint angles, if a valid solution exists.

To perform the calculation, a solver of type `ChainIkSolverPos_NR_JL` from PyKDL is used. This solver supports joint limits, making it suitable for realistic robot control. It requires the kinematic chain, lower and upper joint limits, a forward kinematics solver (`ChainFkSolverPos_recursive`), and an inverse velocity solver (`ChainIkSolverVel_wdls`). A weight matrix `L` is provided to the velocity solver to constrain the task space to position only — disregarding the end-effector's orientation. This simplification is intentional, as only the position of the end-effector is relevant in this project. Additionally, without the weight matrix, the solver fails to converge even for valid target positions, making it a necessary configuration.

The IK solver also requires an initial guess, which significantly influences the ability to find a solution. In this implementation, the initial joint configuration is set to the mean value between each joint's upper and lower limits. This choice improves convergence compared to starting with zero angles, which often leads to failure if the initial pose is too far from the desired target.

The target pose is represented as a Frame object, constructed from the given target position. An empty `JntArray` is passed to store the computed joint angles. The solver then attempts to compute the joint configuration by minimizing the positional error between the initial and target frames, taking into account the joint limits. If successful, the resulting joint angles are returned. Otherwise, a failure message is issued and no invalid solution is returned.

F. Rotate robot arm joints

After computing an inverse kinematics solution, the joints of the robot arm in the Gazebo simulation must be rotated accordingly so that the end-effector reaches the desired target position. The robot is also controlled via Gazebo topics. Each joint can be addressed individually through the topic `/model/panda_arm/joint/joint_name/0/cmd_pos`, where `joint_name` is replaced with the name of the respective joint. A publisher sends a message of type `Double` containing the computed target rotation to each joint topic.

To ensure that the Gazebo simulation correctly processes the commands, the joint commands are not sent simultaneously. Instead, they are published sequentially, with a delay of three seconds between each message. This timing allows the simulation to execute the joint rotations reliably and avoids potential issues caused by overlapping commands.

G. Implementation

The overall implementation, illustrated in Figure 3, follows a structured sequence that connects the simulation environment, inverse kinematics computation, and joint control. First, an object of the PandaRobot class is instantiated to represent the robot arm in Python. As described in Section II-A, this class subscribes to the relevant Gazebo topics to continuously access the current joint states and the end-effector pose.

Next, the kinematic structure of the robot is manually defined by constructing a kinematic chain using PyKDL. Based on the SDF model of the Franka Emika Panda robot, each joint and link is represented as a segment with appropriate rotational axes and transformations.

For a given target position, an inverse kinematics solution is then computed using a numerical solver that accounts for joint limits and considers only the position of the end-effector. If a valid solution is found, the robot joints are rotated sequentially by publishing the computed angles to their corresponding Gazebo topic. After the motion, the resulting state of the robot is validated by calculating the deviations between the actual end-effector position and the target position, as well as the actual joint angles and the calculated joint angles.

If the solver fails to find a solution, an appropriate error message is displayed in the console and no motion is executed.

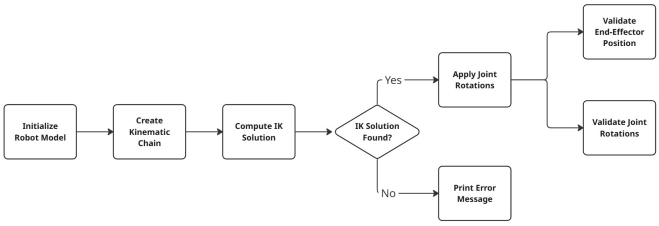

Fig. 3. Implementation Flow

III. Experiments

In this section, a series of experiments are presented to evaluate whether the project goals defined in Section I-C were achieved. Each experiment tests a specific scenario involving inverse kinematics computation and joint rotation in the Gazebo simulation.

A. Experiment 1

In this experiment, the robot starts from its default configuration. The goal is to reach a predefined target position using the computed inverse kinematics solution.

Initial State: Figure 4 shows the robot in its initial configuration on the left.

Input: Target Position: $(-0.2,\ 0.6,\ 0.6)$

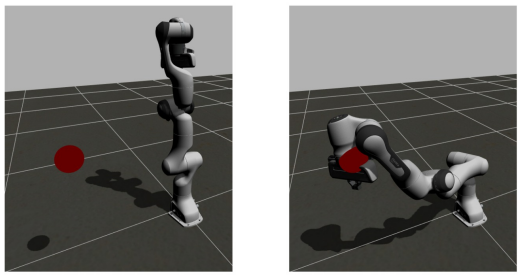

Fig. 4. Experiment 1 – Setup and Execution

Result: Figure 4 on the right shows the final pose of the robot after the joint rotations have been applied. The deviations between the actual end-effector position and the target position, as well as the actual joint angles and the calculated joint angles, are shown in table I.

TABLE I
Deviation Between Target and Final Robot State

End-Effector / Joints	Deviation
End-Effector	(0.0193, 0.0134, 0.0223)
Joint 1	0.0000
Joint 2	0.0045
Joint 3	0.0790
Joint 4	0.0127
Joint 5	0.0186
Joint 6	0.0154
Joint 7	0.0145

B. Experiment 2

In this experiment, the robot arm is commanded to move from a start position to a final target position in several discrete steps.

Initial State: Figure 5 shows the robot in its initial configuration on the left.

Input:
Start Position: $(0.5,\ -0.6,\ 0.4)$
Target Position: $(0.5,\ 0.6,\ 0.4)$
Step size in Y-direction: $\Delta y = 0.3$
Intermediate Positions:

1) $(0.5,\ -0.3,\ 0.4)$
2) $(0.5,\ 0.0,\ 0.4)$
3) $(0.5,\ 0.3,\ 0.4)$

Result: Figure 5 shows the robot arm at each of the positions in the sequence.

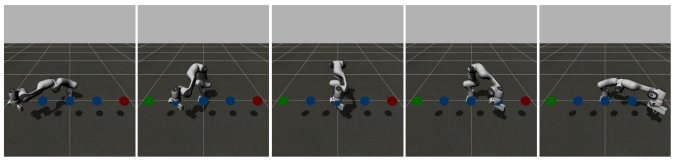

Fig. 5. Experiment 2 – Setup and Execution

C. Experiment 3

In this experiment, the robot is given a target position that is outside of its reachable workspace. The purpose is to evaluate how the system handles situations where no valid inverse kinematics solution can be computed.

Initial State: Figure 6 shows the robot in its original configuration, as well as the unreachable target position (red sphere).

Input: Target Position: $(-0.2,\ 0.0,\ 1.6)$

Result: An error message is printed to the console indicating that no Inverse Kinematics solution could be found.

IV. Discussion

The experiments conducted in Section III demonstrate that all primary goals of the project, as defined in Section I-C, have been successfully achieved.

Experiments 1 and 2 confirm that valid inverse kinematics solutions can be calculated for reachable target positions. In both cases, the end-effector moved to the expected target

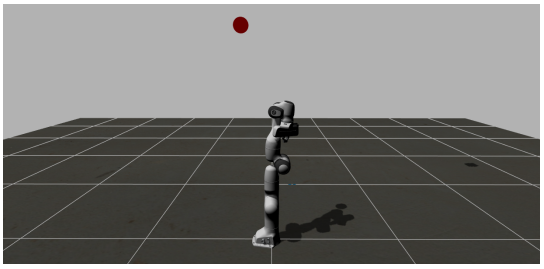

Fig. 6. Experiment 3 – Setup

position in the Gazebo simulation. This verifies the correctness of the IK computation pipeline, as well as the successful retrieval of the required robot data (joint angles, joint limits and end-effector pose) from the simulation environment.

The goal of sending the computed IK solution to the robot joints and executing the corresponding rotation in the simulation is also fulfilled. In both experiments, the robot performs the intended movements and reaches the target positions. However, as shown in Table I of Experiment 1, there is a small deviation between the expected and actual end-effector position. This discrepancy is not due to inaccuracies in the computed IK solution, but rather stems from slight deviations introduced during joint rotation in the simulation. The joint deviation data confirms that some joint rotations in Gazebo do not precisely match the commanded target angles, leading to minor pose errors at the end-effector.

Experiment 3 explicitly verifies the goal of ensuring proper handling of unreachable target pose. The provided target lies outside the robot's workspace, and as expected, the solver fails to compute a solution. The system responds correctly by printing an error message and refraining from executing any joint commands. This confirms that the system safely handles invalid inputs.

In summary, all defined goals were achieved. The implementation proves to be functional, stable, and capable of performing real-time inverse kinematics computations and motion control within a simulation environment.

V. Conclusion and Outlook

This project demonstrates the successful implementation of an inverse kinematics pipeline for a 7-DOF robotic arm within the Gazebo simulation environment. The system is capable of retrieving real-time simulation data, computing valid IK solutions using PyKDL, and executing the resulting joint movements in a controlled and reproducible manner. The experiments confirm that the defined project goals have been fully achieved and that the framework behaves robustly, even in the case of unreachable target positions.

If more time were available, future work would focus on improving the precision of joint control in the Gazebo simulation. Although the IK solver provides accurate results, slight deviations occur due to imperfect execution of joint rotations. Fine-tuning the communication and control mechanisms could further reduce these discrepancies. Additionally, visualizing the constructed kinematic chain would be a valuable enhancement for understanding and debugging the internal structure of the robot model, especially in more complex setups.

References

[1] M. Aksu, J. L. Michaloski, and F. M. Proctor. Virtual experimental investigation for industrial robotics in gazebo environment. In *ASME International Mechanical Engineering Congress and Exposition*, volume 52019, page V002T02A071. American Society of Mechanical Engineers, 2018.

[2] S. Kucuk and Z. Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.

[3] OpenRobotics. Panda arm - fortress merge include demo, 2023.

[4] OpenRobotics. Panda hand - fortress merge include demo, 2023.

[5] S. Vaish. Forward kinematics using orocos kdl, 2017.

[6] M. Wołczyk, M. Zajac, R. Pascanu, Łukasz Kuciński, and P. Miłoś. Continual world: A robotic benchmark for continual reinforcement learning, 2021.

Appendix A: Code and Setup Instructions

System Overview

The system on which the project was developed and tested is a virtual machine (VM) with the following specifications:

- OS: Ubuntu 22.04.5 LTS (Jammy Jellyfish)
- Memory: 8 GB RAM
- CPU: 2 Cores
- GPU: NVIDIA GeForce GTX 1070 - 8 GB

Software Requirements

The following software is required for the project:

- Python 3.10.12
- liborocos-kdl 1.5.1
- pykdl 1.5.1
- numpy 1.21.5

Repository

The repository of the project can be found and pulled under https://git-ce.rwth-aachen.de/tim.liehr/research-project-gazebo-rl

Setup instructions

In order to set up the project after the pull, all dependencies must first be installed. First, gazebo and python must be installed. Then all required packages can be installed with the bash script `./setup-project.sh`.

Once the project has been successfully set up, the project can be started. To do this, the simulation must be opened in the Gazebo GUI using the bash script `./start-simulation.bash` and then started using the "Run the simulation" button. The desired target position that the end-effector of the robot arm should reach must be set in the `simulation/IKPanda.py` script with the variable `target_position`. When the script is executed, an IK solution is calculated for the target position and the joints of the robot arm are rotated accordingly in the Gazebo simulation so that the end-effector is moved to the target position.

All steps for setting up and starting the project are also detailed in the README of the repository.