Design & Implementation of Task-Priority Kinematic Control System for a Mobile Manipulator

Ahmed Alghfeli Hassan Alhosani Reem Almheiri

I. Abstract

This report presents the design and implementation of a Task-Priority Kinematic Control System for a Mobile Manipulator. The project focuses on utilizing the Task-Priority redundancy resolution algorithm to control a Kobuki Turtlebot 2 equipped with a 4 DOF manipulator. The system incorporates forward and inverse kinematics calculations to enable precise control of the manipulator's end-effector. The goal is to develop a comprehensive control system that allows for efficient task execution, showcasing the potential of mobile manipulators in real-world applications. The report outlines the hardware and software architectures, discusses the forward and inverse kinematics algorithms, and presents the implementation of the task priority algorithm. Various scenarios, including end-effector positioning, end-effector configuration transitions, and ArUco marker-based pick-and-place tasks, were executed to validate the system's performance. The results demonstrate the effectiveness and reliability of the implemented control system, highlighting its potential for practical applications.

II. Introduction

The ability of robotic manipulators to perform multiple tasks simultaneously with precision is crucial in various applications. The task-priority scheme plays a significant role in effectively managing multiple tasks. This project focuses on the implementation of the Task-Priority redundancy resolution algorithm to control a mobile manipulator using a forward and inverse kinematics control system. The experimental platform chosen for this project is the Kobuki Turtlebot 2, which is equipped with a 4 DOF manipulator. The end-effector of the manipulator is equipped with a vacuum gripper, enabling pick and place tasks. The goal of this project is to develop a comprehensive control system that allows for efficient task execution and demonstrates the potential of mobile manipulators in real-world applications.

To have a deep understanding of the functioning of the robot's software and hardware architecture to discover the possibilities and necessary developments. The figure below shows the architecture of the Kobuki Turtlebot.

A. Block diagram of robot hardware architecture

A high-level block diagram of the robot hardware architecture, including the mechanical structure, actuators, sensors, and compute unit. Raspberry Pi on TurtleBot 2: Broadcom

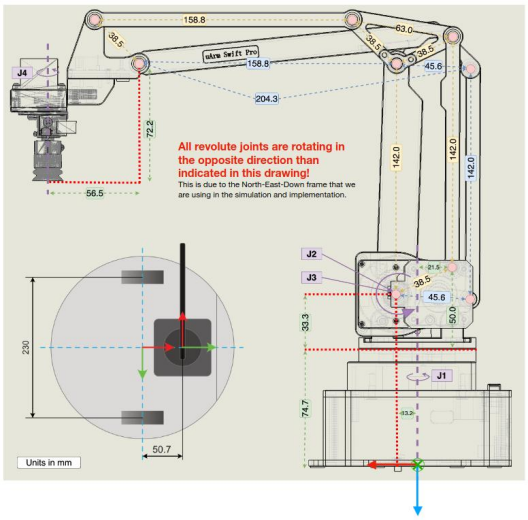

Fig. 1. Robot model

BCM2837 SoC with 1.2GHz quad-core ARM CortexA53 CPU. It handles communication protocols and data exchange with the computer and robot. RPLIDAR A2 sensor scans environment using laser triangulation. Raspberry Pi communicates with LiDAR for obstacle avoidance.

TurtleBot 2 control: Raspberry Pi communicates asynchronously at 115200bit/s. Frames contain header, payload, length, and Checksum. Movement commands are sent in frames. High-capacity lithium-Ion battery powers the robot, with a switching regulator to convert voltages.

Intel RealSense D435i camera: Wide field of view (approximately 85 degrees). It uses global shutter cameras for simultaneous pixel capture.

uArm SwiftPro: Used for education, compatible with ROS, Arduino, GRABCAD, etc. It has a 0.2mm position repeatability, stepper motor, and 12-bit encoder. uArm has 3 Degrees of Freedom and power input of 100 240V 50/60Hz Output: 12V5A 60W.

B. Block diagram of the software architecture

A detailed block diagram of the software architecture used in the robot simulation setup.

C. Block diagram of the control architecture

A detailed functional block diagram of the designed kinematic control system, including all necessary components of

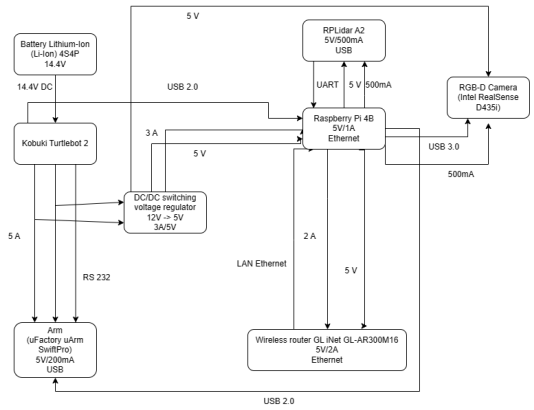

Fig. 2. Hardware architecture

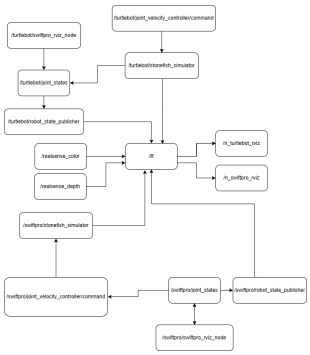

Fig. 3. oftware architecture

the control system, interfacing with robot architecture, inputs, outputs, and message types.

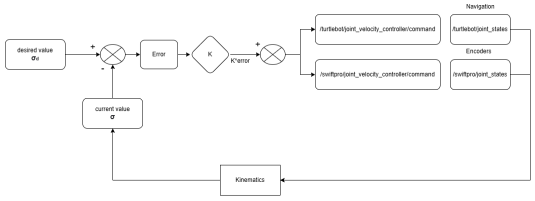

Fig. 4. Control architecture

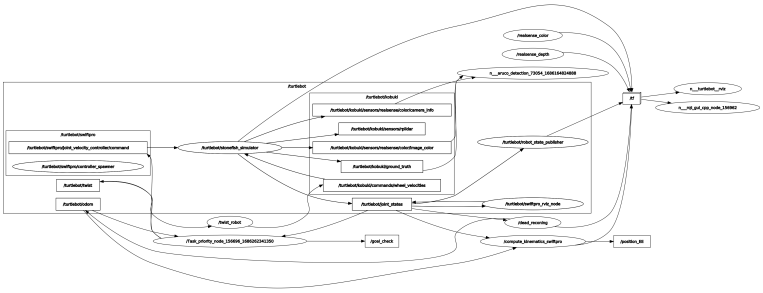

Fig. 5. rqt graph showing running nodes

D. *forward kinematics*

Because it is difficult to construct a Denavit–Hartenberg table due to the nature of the arm. Derivation was done geometrically!

Fig. 6. Derivation

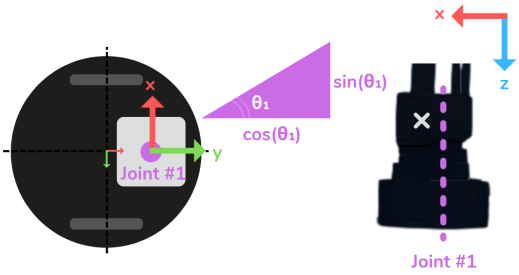

Fig. 7. Joint 1 effect

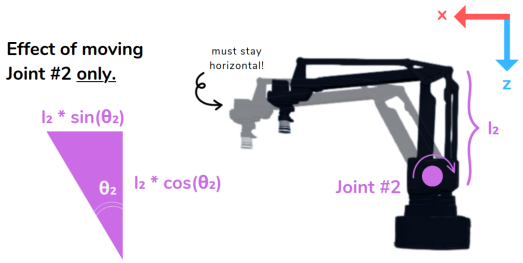

Fig. 8. Joint 2 effect

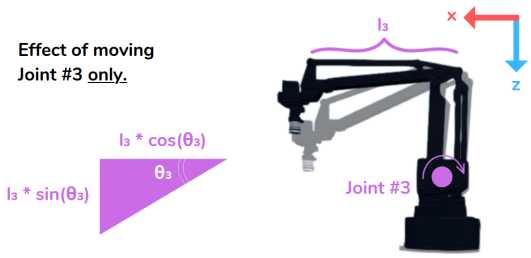

Fig. 9. Joint 3 effect

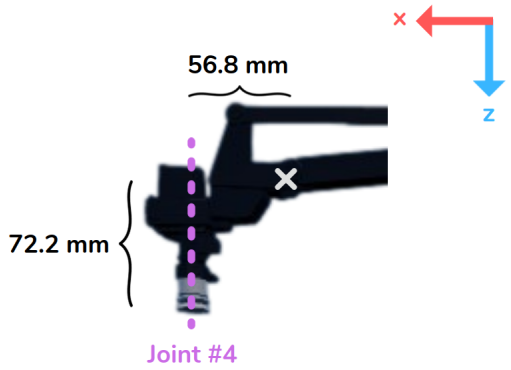

Fig. 10. Joint 4 effect

putting everything together we the get End-Effector transformation:

$$^{armbase}T_{ee} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (69.7 - 142sin(q2) + 158.8cos(q3))cos(q1) \\ (69.7 - 142sin(q2) + 158.8cos(q3))sin(q1) \\ -35.8 - 142cos(q2) - 158.8sin(q3) \end{bmatrix}$$

To get the transformation from the arm base to the robot base **tf_echo** was used to obtain the fixed transformation:

$$^{robotbase}T_{armbase} = \begin{bmatrix} \cos(-\frac{\pi}{2}) & -\sin(-\frac{\pi}{2}) & 0 & 51 \\ \sin(-\frac{\pi}{2}) & \cos(-\frac{\pi}{2}) & 0 & 0 \\ 0 & 0 & 1 & -198 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using dead reckoning to estimate the robot's pose we get:

$$^{world}T_{robotbase} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 & x_{robot} \\ \sin(yaw) & \cos(yaw) & 0 & y_{robot} \\ 0 & 0 & 1 & z_{robot} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{world}T_{ee} = ^{world}T_{robotbase} \cdot ^{robotbase}T_{armbase} \cdot ^{armbase}T_{ee}$$

To incorporate the full transformation with the robot, we have added two additional joints: a prismatic joint to represent linear velocity and a revolute joint to represent angular velocity. These joints enable the robot to have a complete range of motion, allowing it to move and rotate smoothly and accurately.

$$T = \begin{bmatrix} \sin(\theta_r) & \cos(\theta_r) - \sin(\theta_r) & 0 & x_w + d \cdot \cos(\theta_r) \\ -\cos(\theta_r) & \cos(\theta_r) + \sin(\theta_r) & 0 & y_w + d \cdot \sin(\theta_r) \\ 0 & 0 & 1 & z_w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} x_{ee}sin(\theta_r) + 51cos(\theta_r) + dcos(\theta_r) + y_{ee}(cos(\theta_r) - sin(\theta_r)) + x_r \\ 51sin(\theta_r) + dsin(\theta_r) + y_{ee}(cos(\theta_r) + sin(\theta_r)) - x_{ee} * cos(\theta_r) + y_r \\ z_r + z_{ee} - 198 \end{bmatrix}$$

E. inverse kinematics - jacobian

The Jacobian provides a relationship between joint velocities and end-effector velocities, enabling us to control the motion of a robot. To calculate the linear velocity, we take the partial derivative of the transformation's x, y, and z components with respect to the parameter representing linear displacement (d). To obtain the angular velocity, we take the partial derivative of the transformation with respect to the parameter representing angular displacement. Finally, We take the partial derivatives of the transformation with respect to each joint variable (q1, q2, q3, q4) to obtain the joint velocities. This enables us to calculate how changes in each joint's angle affect the robot's total motion.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial d} & \frac{\partial x}{\partial \theta_r} & \frac{\partial x}{\partial q1} & \frac{\partial x}{\partial q2} & \frac{\partial x}{\partial q3} & \frac{\partial x}{\partial q4} \\ \frac{\partial y}{\partial d} & \frac{\partial y}{\partial \theta_r} & \frac{\partial y}{\partial q1} & \frac{\partial y}{\partial q2} & \frac{\partial y}{\partial q3} & \frac{\partial y}{\partial q4} \\ \frac{\partial z}{\partial d} & \frac{\partial z}{\partial \theta_r} & \frac{\partial z}{\partial q1} & \frac{\partial z}{\partial q2} & \frac{\partial z}{\partial q3} & \frac{\partial z}{\partial q4} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

III. Implementation

After obtaining the forward and inverse kinematics, a task priority algorithm is implemented using various tasks. These tasks include:

- end-effector position.
- end-effector orientation.
- end-effector configuration.
- joint position.
- and joint limits.

The recursive task-priority algorithm is an approach that allows for the handling of multiple tasks with different priorities in a recursive manner. This algorithm ensures that higher priority tasks are given precedence while still allowing lower priority tasks to be executed. The recursive Task-Priority algorithm takes a list of tasks as an input, and then initialise task velocity = matrix of zeros of dimension (n x 1), Initialize null-space projector as P0 = Identity matrix of dimension (n x n). Where n represents number of robot DOF. then:

1) Iterate over tasks.
2) Update task state.
3) Compute augmented Jacobian Jbar.
4) Compute task velocity dq:

$$\zeta_i = \zeta_{i-1} + \overline{J_i^\dagger}(q)(\dot{x}_i(q) - J_i(q)\dot{\zeta}_{i-1})$$

5) Update null-space projector:

$$P_i = P_{i-1} - \overline{J_i^\dagger}(q)\overline{J_i}(q)$$

To address the joint limit task, a different approach is used. In order to ensure that the robot reaches its desired goal without violating joint limits, a (DLS) weighting strategy is utilized. The DLS weighting is adjusted in a way that when the robot is far from the desired goal, the emphasis is placed on keeping the arm stationary while allowing the robot to move freely. As the robot approaches the desired goal, the DLS weighting is adjusted to prioritize arm movement and slow down the robot's motion. This approach helps to prevent joint limit violations while ensuring smooth and accurate reaching of the desired goal.

Definition of different task hierarchies is presented below:

1) One task: 1- End-Effector Position.
2) One task: 1- End-Effector Configuration.
3) Two tasks: 1- End-Effector Position 2- Joint 1 Position.
4) Two tasks: 1- Joint limit 2- End-Effector Configuration.

- End-Effector Position **[x, y, z]**.
- End-Effector Orientation **[angle of EE]**.
- End-Effector Configuration **[x, y, z, angle of EE]**.
- Joint Position **position of q**.
- Joint Limits **joint 1 and 4 [-90, 90] joint 2 and 3 [-90, 0.05]**.

The implementation process consists of several phases, each with a set of steps that outline the procedure:

1) Initialization
   a) Set up the robot and all required hardware components.
   b) Establish communication between the robot and the control system.
   c) Calibrate sensors and actuators for accurate measurements and movements.
2) Forward and Inverse Kinematics
   a) Develop algorithms to calculate the forward kinematics, mapping joint positions to end-effector position and orientation.
   b) Implement inverse kinematics to determine joint positions based on desired end-effector position and orientation.
3) Task Priority Algorithm
   a) Define the tasks to be accomplished, such as end-effector position, orientation, configuration, joint position, and joint limits.
   b) Assign priority levels to the tasks based on their importance and dependencies.
   c) Implement the task priority algorithm to handle multiple tasks concurrently, considering their priorities.
4) Weighted DLS
   a) Implement a Dynamic Load Sharing (DLS) weighting approach to ensure the robot respects joint limits while achieving task goals.
   b) Adjust the DLS weighting dynamically based on the proximity to the desired goal, prioritizing arm movement as the goal is approached.
5) Testing and Refinement
   a) Execute test scenarios to validate the implemented algorithm and verify the robot's performance.
   b) Collect data and evaluate the robot's behavior in achieving different tasks and handling joint limits.
   c) Analyze the results and refine the algorithm if necessary, iterating the implementation process as needed.

IV. Results & Discussion

To validate the implementation of our system, we define and execute various scenarios. These scenarios cover different aspects and functionalities of the system, allowing us to assess its performance and identify any potential issues. Additionally, we generate various plots to analyze the system's behavior and performance:

- Plots presenting motion of the mobile base and the end effector in the X-Y plane.
- Plots presenting the evolution of control errors over time.
- Plots presenting the evolution of joint velocities over time.

Here are the scenarios, including a description of each of the performed tasks:

1) Move End Effector Toward Desired Position:
   - This scenario involves moving the end effector towards a specified position in space.

- Video link: https://we.tl/t-SNmvia79Tz

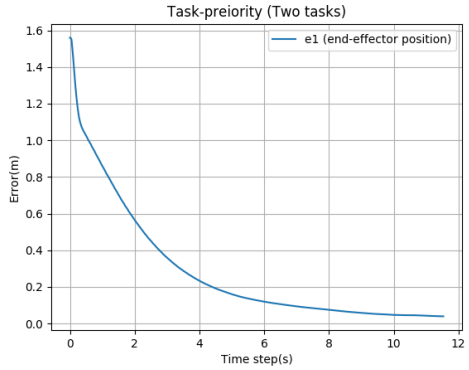

Fig. 11. evolution of the end-effector position task error over time.

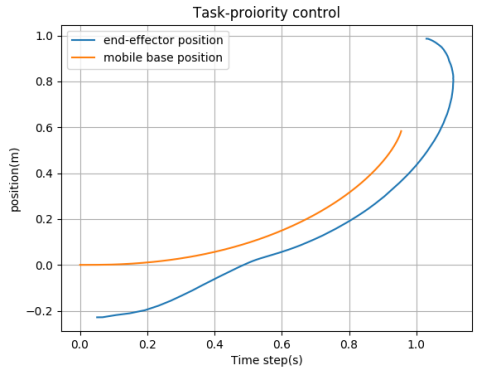

Fig. 12. evolution of the mobile base position and the end-effector position, on the X-Y plane.

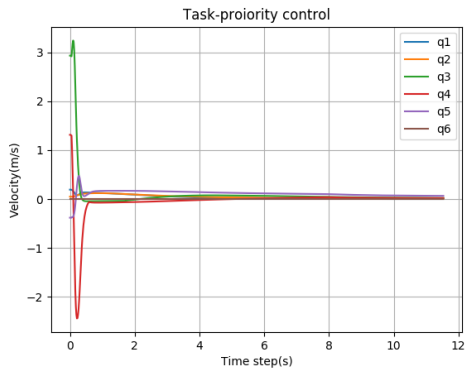

Fig. 13. Evolution of joints velocities.

2) Sequence of Motions Between Desired Cartesian End-Effector Configurations:
   - In this scenario, a sequence of motions is executed to transition between different desired Cartesian end-effector configurations.
   - Video link: https://we.tl/t-RzNPdMb7Fm

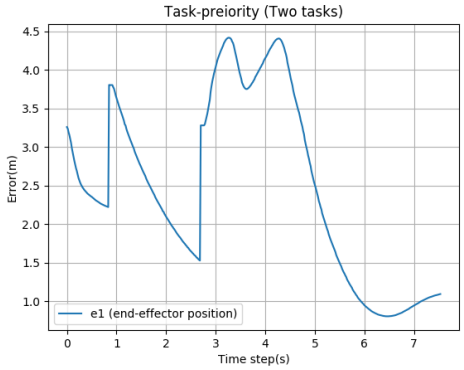

Fig. 14. evolution of the end-effector position task error over time.

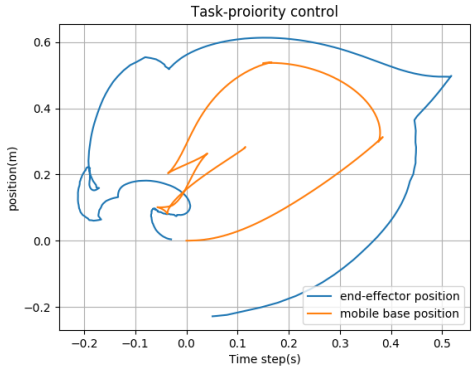

Fig. 15. evolution of the mobile base position and the end-effector position, on the X-Y plane.

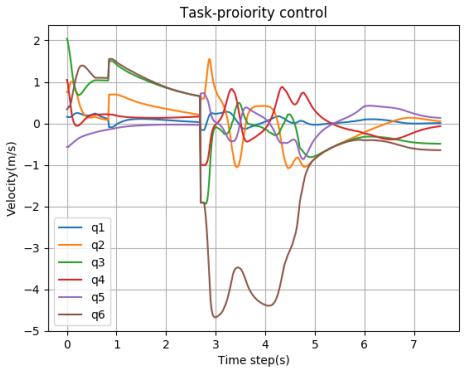

Fig. 16. Evolution of joints velocities.

3) ArUco Detection - Pick and Place:
   - This scenario focuses on ArUco detection, a marker-based tracking system. The system detects ArUco markers and performs a pick-and-place task based on the marker's position.
   - Video link: https://we.tl/t-RFGwjCkXSC

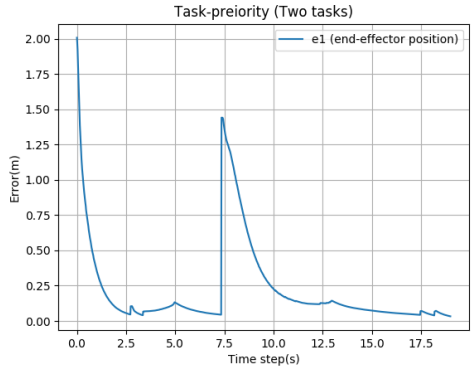

Fig. 17. evolution of the end-effector position task error over time.

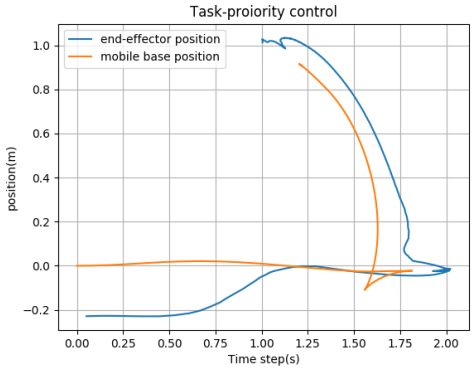

Fig. 18. evolution of the mobile base position and the end-effector position, on the X-Y plane.

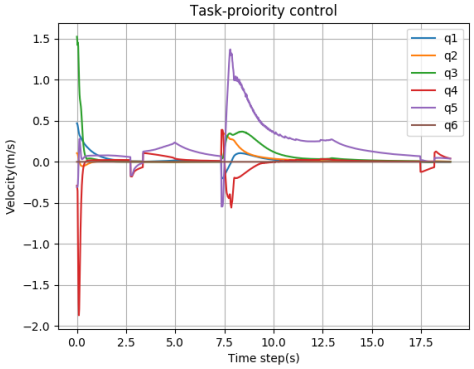

Fig. 19. Evolution of joints velocities.

By conducting these scenarios and analyzing the corresponding data and plots, we can ensure the effectiveness and reliability of our system implementation.

A. *result of real robot*

The robot detects an Aruco marker, which indicates the position of a box. It then proceeds to move towards the box, picks it up, and finally navigates towards the desired goal to drop off the box.

video link: https://we.tl/t-elwAKIz1BX

V. Conclusion

In conclusion, the goal of this project was to develop and implement a task-priority kinematic control system for a mobile manipulator. The goal was to provide a comprehensive control system that enables effective job execution and shows the use of mobile manipulators in real-world environments. The efficiency and dependability of the installed control system were evaluated through the examination of the data and charts produced throughout the scenarios. Overall, the task-priority kinematic control system for the mobile manipulator that was developed and put into use showed promise for attaining accurate and effective multi-task execution. This project advances the field of mobile manipulator technology and creates opportunities for its use in a variety of practical contexts, including the industrial, logistics, and service sectors.