Kinematic Control of a Vehicle–Manipulator System Using Task-Priority Redundancy Resolution

Pravin Oli (u1999699), Gebrecherkos G. (u1999542)

Abstract—This paper presents a task-priority-based kinematic control framework for a mobile manipulator performing pick, transport, and place operations. The proposed system integrates resolved-rate control, behavior tree planning, dead reckoning for navigation, and ArUco marker-based visual feedback. Experimental validation demonstrates accurate trajectory tracking, smooth task sequencing, and reliable performance under joint and motion constraints. The overall architecture is modular, scalable, and suitable for structured indoor environments.

Index Terms—Mobile manipulation, task-priority control, behavior trees, ArUco detection, dead reckoning, robot kinematics.

1 Introduction

VEHICLE-manipulator systems, which integrate a mobile robotic base with a dexterous arm, are increasingly important in modern robotics due to their versatility in dynamic and semi-structured environments. Their applications span warehouse automation, industrial palletizing, search-and-rescue missions, and domestic service robotics. A key challenge in controlling such systems lies in managing the redundancy introduced by multiple degrees of freedom while ensuring efficient and prioritized task execution.

This work presents the design and implementation of a kinematic control system for a mobile manipulator using the Task-Priority Redundancy Resolution Algorithm. The experimental platform consists of a differential-drive mobile base (Kobuki Turtlebot 2) integrated with a 4-DOF manipulator (uFactory uArm Swift Pro). This hardware setup, commonly found in palletizing and automation tasks, is equipped with a vacuum gripper, wheel encoders, a 2D LiDAR (RPLidar A2), and an RGB-D camera (Intel RealSense D435i) to enable both actuation and perception. The core objective is to implement a task-priority control framework capable of simultaneously managing multiple tasks such as end-effector positioning, joint limit avoidance, and base orientation.

2 Related Works

Kinematic control for redundant manipulators has been widely studied, with the task-priority framework enabling simultaneous task execution through a hierarchy of priorities. Baerlocher and Boulic [1] introduced a hierarchical projection method where each task operates in the null space of higher-priority tasks—ideal for mobile manipulators requiring coordinated base-arm control.

• *Master students, Erasmus Mundus Master's Program in Intelligent Field Robotic Systems (IFRoS), University of Girona, Spain.*
Email: pravin.oli.08@gmail.com, chereg2016@gmail.com

Cieślak et al. [2] applied this approach to field robotics, integrating obstacle avoidance as a low-priority task to ensure adaptability in cluttered environments. The system design in this work follows best practices from the Hands-On Intervention (HOI) lectures [3], emphasizing ROS-based architecture and real-time coordination.

In addition, behavior trees have emerged as a modular and reactive strategy for sequencing robotic tasks. The `py_trees` library [4] offers a flexible Python framework for building behavior trees, which has proven effective in orchestrating complex pick-and-place operations in robotic systems.

3 System Design

The proposed system architecture is designed to enable task-priority-based kinematic control for a mobile manipulator. It integrates perception, computation, and actuation within a unified ROS-based framework. The architecture is compatible with both simulation and real-world deployment, facilitating seamless testing and validation. It is organized into three main layers: hardware architecture, software infrastructure, and control logic.

3.1 Hardware Architecture

The physical platform is built on the Kobuki TurtleBot 2, a differential-drive mobile base responsible for locomotion and low-level feedback such as odometry, wheel encoders, and inertial sensing. Mounted on the base is a 4-DOF uFactory uArm Swift Pro manipulator equipped with a vacuum gripper, enabling it to perform standard pick-and-place operations.

A Raspberry Pi 4B serves as the primary compute unit, interfacing with all sensors and actuators. Power is supplied by a 4S4P Lithium-Ion battery (14.8 V), regulated to 5 V/3 A via a DC/DC converter to support peripheral components including:

- **Intel RealSense D435i RGB-D Camera:** Provides real-time depth and RGB data for visual servoing and ArUco marker pose estimation.
- **RPLidar A2:** Offers 2D laser scans for mapping and obstacle detection.
- **GL-iNet Wireless Router:** Enables external monitoring, debugging, and communication over WiFi.

Fig. 1 illustrates the hardware layout and power distribution among the components.

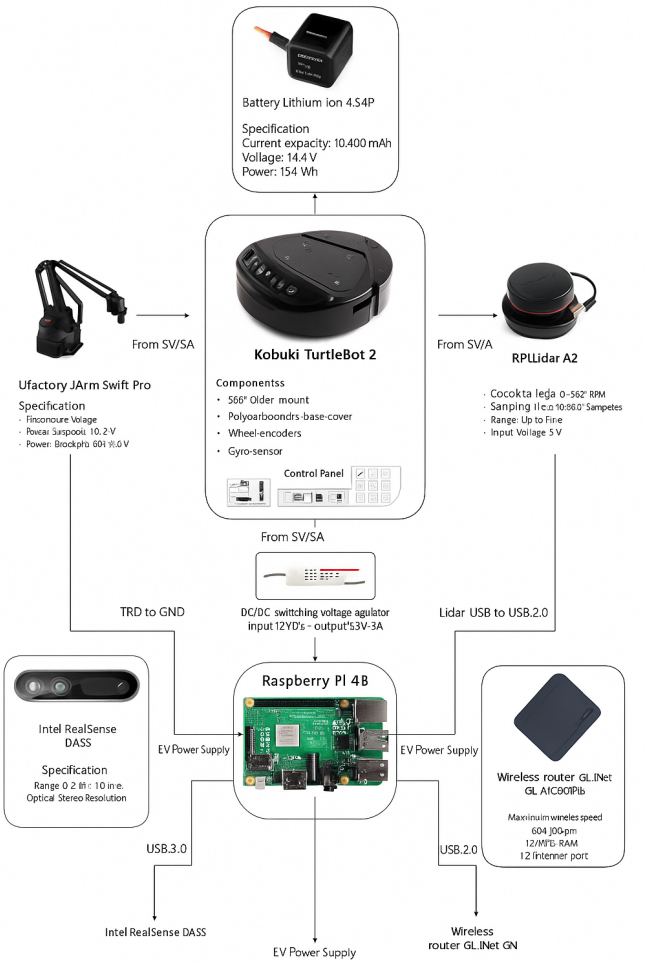


Fig. 1: Hardware architecture of the mobile manipulator platform.

3.2 Software Architecture

The software system is built entirely on the Robot Operating System (ROS), following a modular, node-based structure. All inter-node communication is handled through ROS topics and services, ensuring real-time data exchange and system modularity. Fig. 2 depicts the overall communication topology.

Sensor feedback is obtained from:

- `/turtlebot/joint_states`: Joint position and velocity feedback.
- `/turtlebot/kobuki/sensors/realsense/color/image color`: RGB camera feed.

Control commands are sent to:

- `/turtlebot/kobuki/commands/wheel_velocities`: Base motion control.
- `/turtlebot/swiftpro/joint velocity controller/command`: Manipulator joint control.
- `/turtlebot/swiftpro/vacuum_gripper/pump_state`: Gripper actuation.

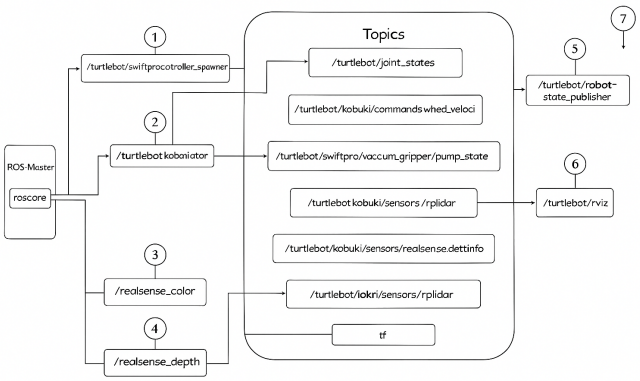


Fig. 2: Software architecture showing ROS communication topology.

3.3 Control Architecture

The control logic as shown Fig. 3 is structured as a task-priority behavior tree, organizing multiple control objectives into a hierarchical execution flow. High-priority tasks such as end-effector pose regulation and joint limit avoidance are enforced first, followed by lower-priority tasks such as base orientation.

The desired end-effector pose is estimated from ArUco marker detection and passed dynamically to the controller. The task-priority node resolves joint velocities using null-space projection, ensuring that lower-priority objectives do not interfere with higher-priority constraints.

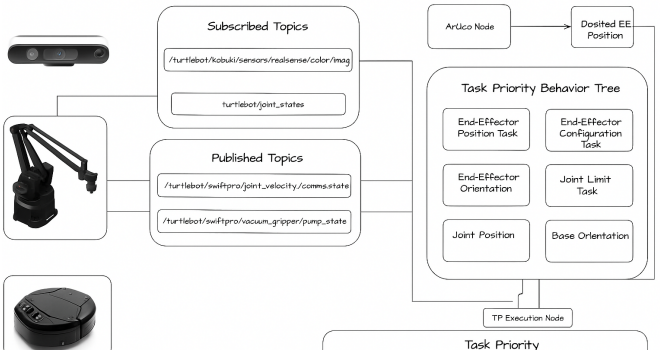


Fig. 3: Task-priority-based control architecture integrating arm and base motion.

4 Kinematics Formulation

This section presents the kinematic formulation of the mobile manipulator, consisting of a differential-drive base and

a 4-DOF uArm Swift Pro. The derivation is structured in two stages. First, the forward kinematics and analytical Jacobian of the manipulator are derived independently based on its geometric model. Then, the manipulator kinematics are extended by incorporating the mobile base to obtain the complete system kinematics in the world frame.

4.1 Manipulator Kinematics

The structure of the uArm Swift Pro is shown in Fig. 4. The robot employs a parallel linkage mechanism, and its kinematics are derived using trigonometric relationships. The reference frame follows the North-East-Down (NED) convention, with the origin placed at the manipulator base and an offset of $50.7\,\text{mm}$ along the x-axis.

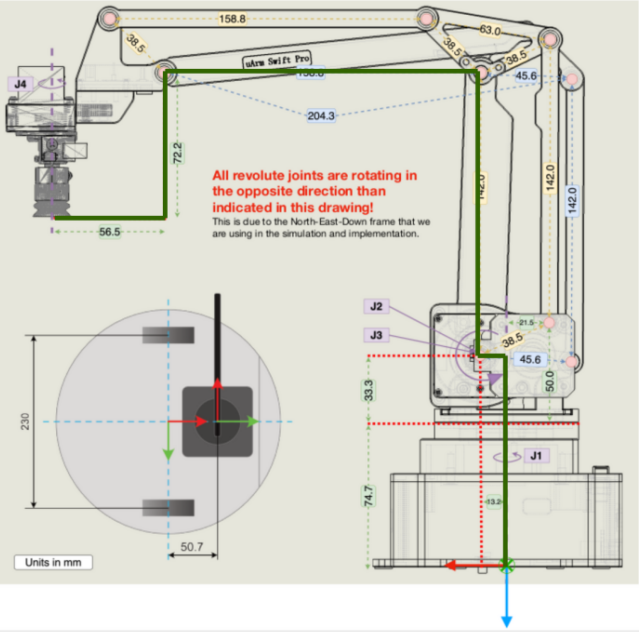


Fig. 4: Mechanical configuration and physical dimensions of the uArm Swift Pro mounted on the Kobuki Turtlebot 2.

The arm's geometry is further illustrated in Fig. 5, which presents both the side view (R-Z plane) and top view (X-Y plane) of the kinematic chain. The key joint variables are denoted as q_1 through q_4, with segment lengths and offsets labeled accordingly.

The configuration vector of the system is defined as:

$$\mathbf{q} = \begin{bmatrix} \boldsymbol{\eta} & \mathbf{q}_a \end{bmatrix}^\top, \quad \boldsymbol{\eta} = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top, \quad \mathbf{q}_a = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^\top \tag{1}$$

Based on the geometric configuration, the forward kinematics of the manipulator end-effector in the arm frame are given by:

$$R = 158.8\cos(q_3) - 142\sin(q_2) + 56.5 + 13.2 \tag{2}$$
$${}^{A}E_x = R\cos(q_1) \tag{3}$$
$${}^{A}E_y = R\sin(q_1) \tag{4}$$
$${}^{A}E_z = 72.2 - 108 - 142\cos(q_2) - 158.8\sin(q_3) \tag{5}$$

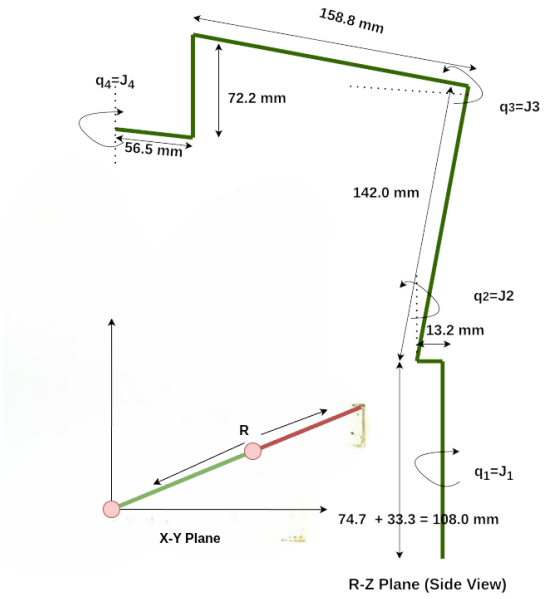


Fig. 5: Manipulator geometry: side view (R-Z plane) and top view (X-Y plane).

The analytical Jacobian is computed by differentiating the end-effector position with respect to the joint variables:

$$J_i(q) = \begin{bmatrix} \frac{\partial E_x}{\partial q_1} & \frac{\partial E_x}{\partial q_2} & \frac{\partial E_x}{\partial q_3} & \frac{\partial E_x}{\partial q_4} \\ \frac{\partial E_y}{\partial q_1} & \frac{\partial E_y}{\partial q_2} & \frac{\partial E_y}{\partial q_3} & \frac{\partial E_y}{\partial q_4} \\ \frac{\partial E_z}{\partial q_1} & \frac{\partial E_z}{\partial q_2} & \frac{\partial E_z}{\partial q_3} & \frac{\partial E_z}{\partial q_4} \end{bmatrix} \tag{6}$$

4.2 Full System Kinematics

To describe the complete manipulator in the global reference frame, the base's motion is incorporated. The transformation from arm frame to world frame is constructed using homogeneous transformations:

$${}^{W}T_E = {}^{W}T_R \cdot {}^{R}T_A \cdot {}^{A}T_E \tag{7}$$

Each transformation matrix is defined as:

$${}^{W}T_R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x \\ \sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$${}^{R}T_A = \begin{bmatrix} 0 & 1 & 0 & 0.0507 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -H_R \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$${}^{A}T_E = \begin{bmatrix} \cos(q_1+q_4) & -\sin(q_1+q_4) & 0 & {}^{A}E_x \\ \sin(q_1+q_4) & \cos(q_1+q_4) & 0 & {}^{A}E_y \\ 0 & 0 & 1 & {}^{A}E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

From this, the end-effector position in the world frame becomes:

$${}^{W}E_x = R\sin(q_1+\theta) + 50.7\cos(\theta) + x \tag{11}$$
$${}^{W}E_y = -R\cos(q_1+\theta) + 50.7\sin(\theta) + y \tag{12}$$
$${}^{W}E_z = 72.2 - 108 - 142\cos(q_2) - 158.8\sin(q_3) - H_R \tag{13}$$

4.3 Full System Jacobian

To generalize motion control, the full system Jacobian incorporates both the mobile base and the manipulator:

$$\dot{\mathbf{E}} = J(q) \begin{bmatrix} \boldsymbol{\nu} \\ \dot{\mathbf{q}}_a \end{bmatrix}, \quad \text{where} \quad \boldsymbol{\nu} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{bmatrix}^\top \tag{14}$$

The base is modeled using Denavit–Hartenberg parameters as a 2-DOF mechanism (prismatic and revolute), as shown in Table 1:

TABLE 1: Denavit–Hartenberg Parameters for Mobile Base

DOF	θ_i	a_i	d_i	α_i
1	$-\pi/2$	$-H_R$	0	$-\pi/2$
2	0	0.0507	0	$\pi/2$

Using the recursive formulation, the base Jacobian J_{base} is computed as:

$$J_i = \begin{bmatrix} \rho_i \, \mathbf{z}_{i-1} \times (\mathbf{O}_n - \mathbf{O}_{i-1}) + (1 - \rho_i) \, \mathbf{z}_{i-1} \\ \rho_i \, \mathbf{z}_{i-1} \end{bmatrix} \tag{15}$$

The final Jacobian used in control is the concatenation:

$$J(q) = \begin{bmatrix} J_{\text{base}} & J_{\text{arm}} \end{bmatrix} \tag{16}$$

This results in a combined $6 \times n$ Jacobian matrix $J(q)$ used for resolved-rate control and task-priority execution.

5 Task Priority Formulation

The control strategy for the mobile manipulator leverages task-priority-based redundancy resolution, allowing multiple objectives to be achieved simultaneously while respecting a strict hierarchy. This section builds upon resolved-rate motion control and introduces both equality and inequality tasks within a recursive null-space framework.

5.1 Resolved-Rate Motion Control

Resolved-rate control computes the system's generalized velocities that realize a desired end-effector velocity. The relationship is given by:

$$\dot{\boldsymbol{\sigma}} = J(\mathbf{q})\boldsymbol{\zeta} \tag{17}$$

where $\dot{\boldsymbol{\sigma}}$ is the task-space velocity, $J(\mathbf{q})$ is the system Jacobian, and $\boldsymbol{\zeta}$ is the generalized velocity vector (including joint and base velocities).

To drive the system toward a desired task-space pose $\boldsymbol{\sigma}_d$, the control law is:

$$\boldsymbol{\zeta} = J^\dagger(\mathbf{q}) \left(\dot{\boldsymbol{\sigma}}_d + K\tilde{\boldsymbol{\sigma}} \right) \tag{18}$$

where $\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_d - \boldsymbol{\sigma}$ is the task error, and K is a positive-definite gain matrix.

5.2 Task Types and Control Objectives

5.2.1 Equality Tasks

Equality tasks are defined as regulation objectives that must be satisfied exactly or tracked over time:

$$\dot{\boldsymbol{\sigma}}_i = \dot{\sigma}_i + K\tilde{\sigma}_i, \quad \tilde{\sigma}_i = \sigma_{i,d} - \sigma_i \tag{19}$$

Typical equality tasks include:

- End-effector position and orientation tracking
- Base posture control
- Desired joint positions or trajectories

Algorithm 1 Recursive Task-Priority Control

Require: List of tasks $\{J_i(\mathbf{q}), \dot{\boldsymbol{\sigma}}_i, a_i(\mathbf{q})\},\ i = 1 \dots k$
Ensure: Output velocity vector $\boldsymbol{\zeta}_k$
Initialize: $\boldsymbol{\zeta}_0 \leftarrow \mathbf{0}$, $P_0 \leftarrow I_{n \times n}$
for $i = 1$ to k **do**
  if $a_i(\mathbf{q}) \neq 0$ **then**
    $\bar{J}_i \leftarrow J_i P_{i-1}$
    $\boldsymbol{\zeta}_i \leftarrow \boldsymbol{\zeta}_{i-1} + \bar{J}_i^\dagger \left(a_i(\mathbf{q})\dot{\boldsymbol{\sigma}}_i - J_i \boldsymbol{\zeta}_{i-1} \right)$
    $P_i \leftarrow P_{i-1} - \bar{J}_i^\dagger \bar{J}_i$
  else
    $\boldsymbol{\zeta}_i \leftarrow \boldsymbol{\zeta}_{i-1}$, $P_i \leftarrow P_{i-1}$
  end if
end for
return $\boldsymbol{\zeta}_k$

5.2.2 Inequality Tasks (Set Constraints)

Set-based (inequality) tasks ensure that certain variables remain within predefined safety bounds:

$$\sigma_i(\mathbf{q}) \in \mathcal{S}_i = [q_{i,\min}, q_{i,\max}] \tag{20}$$

Examples:

- Joint limit avoidance
- Obstacle avoidance margins

These tasks are activated only when the variable approaches the boundary of $\mathcal{S}_i$, using an activation function $a_i(\mathbf{q})$:

$$a_{li}(\mathbf{q}) = \begin{cases} -1, & q_i \geq q_{i,\max} - \alpha_{li} \\ 1, & q_i \leq q_{i,\min} + \alpha_{li} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

where α_{li} defines the threshold margin for activation.

5.3 Recursive Task-Priority Formulation

To manage multiple tasks of differing priority, the Recursive Task-Priority (TP) algorithm is used. Each task is projected into the null space of all higher-priority tasks.

5.3.1 Initialization

$$\boldsymbol{\zeta}_0 = \mathbf{0}, \quad P_0 = I_{n \times n} \tag{22}$$

5.3.2 Recursive Update

For task $i = 1, \dots, k$ (with k total tasks), compute:

$$\bar{J}_i = J_i P_{i-1} \tag{23}$$

$$\boldsymbol{\zeta}_i = \boldsymbol{\zeta}_{i-1} + \bar{J}_i^\dagger \left(a_i(\mathbf{q})\dot{\boldsymbol{\sigma}}_i - J_i \boldsymbol{\zeta}_{i-1} \right) \tag{24}$$

$$P_i = P_{i-1} - \bar{J}_i^\dagger \bar{J}_i \tag{25}$$

Inactive tasks ($a_i(\mathbf{q}) = 0$) are skipped:

$$\boldsymbol{\zeta}_i = \boldsymbol{\zeta}_{i-1}, \quad P_i = P_{i-1}$$

5.3.3 Final Output

After processing all tasks:

$$\boldsymbol{\zeta}_k \quad \text{(final quasi-velocity vector)} \tag{26}$$

5.4 Extensions for Practical Implementation

Velocity Scaling: To enforce actuator velocity limits:

$$s = \max_i \left(\frac{|\zeta_i|}{\zeta_{i,\max}} \right) \tag{27}$$

$$\zeta \leftarrow \frac{\zeta}{s}, \quad \text{if } s > 1 \tag{28}$$

Solution Weighting: To prioritize or penalize certain joints:

$$\zeta = W^{-1} J^\top \left(J W^{-1} J^\top \right)^{-1} \dot{\sigma}, \quad W = \text{diag}(w_1, \ldots, w_n) \tag{29}$$

where W is a diagonal weight matrix to modulate effort across degrees of freedom.

6 Implementation Strategy

The objective of this project is to execute a complete pick, transport, and place operation using a task-priority control framework combined with behavior tree planning. The process is divided into distinct phases: ArUco detection, base approach, picking, transport, and placement.

6.1 Task Priority Control

The control system is structured using a task-priority framework. After deriving the forward and inverse kinematics, multiple task controllers are defined, including joint limit enforcement, end-effector positioning and configuration, joint posturing, and mobile base control. Joint limits are given the highest priority to prevent actuator saturation and unsafe configurations.

6.2 Task Sequencing with Behavior Trees

To manage the sequential execution of tasks, a behavior tree is employed. The modular structure of the tree, built using the `py_trees` library, enables hierarchical planning, reactive task switching, and intuitive debugging. Each subtask—such as approaching, picking, and placing—is encapsulated in its own branch for modularity and clarity.

6.3 Navigation via Dead Reckoning

The robot navigates using dead reckoning based on wheel encoder feedback. The left and right wheel angular velocities ω_L and ω_R are used to compute the robot's linear and angular velocities:

$$V = \frac{r}{2}(\omega_R + \omega_L) \tag{30}$$

$$W = \frac{r}{L}(\omega_R - \omega_L) \tag{31}$$

With linear velocity V, angular velocity W, and heading θ, the pose $[x, y, \theta]^\top$ is updated at each timestep k using:

$$x_k = x_{k-1} + V \cdot \cos(\theta_{k-1}) \cdot \Delta t \tag{32}$$

$$y_k = y_{k-1} + V \cdot \sin(\theta_{k-1}) \cdot \Delta t \tag{33}$$

$$\theta_k = \theta_{k-1} + W \cdot \Delta t \tag{34}$$

6.4 Visual Navigation with ArUco Detection

AruCo markers are affixed to target objects to support visual servoing. The RealSense D435i camera mounted on the mobile base detects the marker and publishes its pose relative to the world frame. This pose is used to guide the robot's base toward the target. The ArUco detection stage is shown in Fig 6.

6.5 Execution Flow

The full pick-and-place sequence is composed of the following steps:

1) ArUco Detection

The robot first detects the marker and extracts its global pose for base alignment (Fig. 6a).

2) Base Approach

The mobile base navigates toward the ArUco-detected position, stopping at a safe threshold distance. Base Jacobians are computed from DH parameters (Fig. 6b).

3) End-Effector Positioning

A dedicated subtree adjusts the end-effector's vertical (Z-axis) position to prepare for picking (Fig. 6c).

4) End-Effector Configuration

A separate subtree configures the end-effector's desired pose before and after picking (Fig. 6d).

5) Object Picking

The vacuum gripper is activated to grasp the object at the detected location (Fig. 6e).

6) Transport and Placement

The robot transports the object to the designated placement zone and lowers the end-effector for release (Fig. 6f).

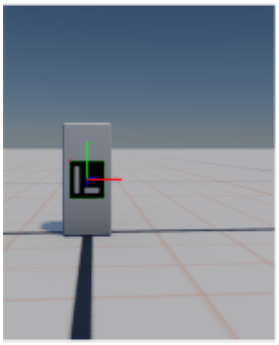
TABLE 2: Usage of Task Blocks in Different Execution Phases

Task Block	Execution Phase
`Joint1- to Joint4 Limit Tasks, Base Task`	During ArUco detection and base navigation (**??**, 6b)
`Joint1 to Joint4 Limit Tasks, End-Effector Position Task`	During alignment near object and before picking (6c)

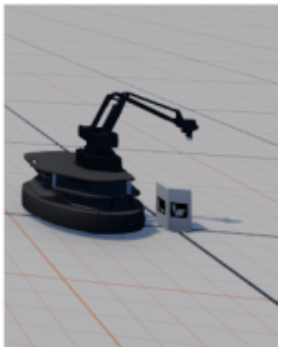
Refer to Fig. 6g for the full behavior tree that integrates all phases.

7 Experiment and Results

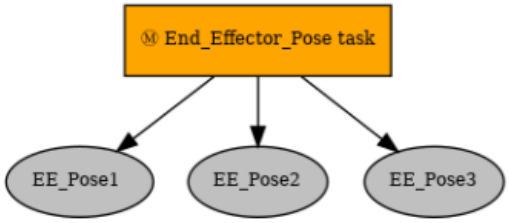
This section presents experimental validation of the proposed system through two main stages: (A) individual task testing and (B) full pick-and-place execution. Each task is evaluated using base and end-effector trajectories, end-effector error plots, and joint velocity profiles.



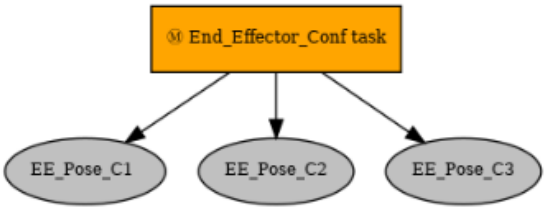
(a) AruCo Detection



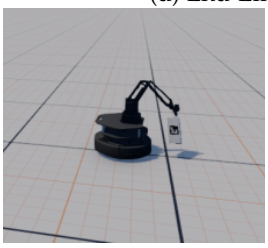
(b) Base Approach



(c) End-Effector Positioning Tree

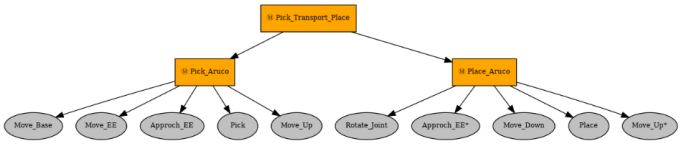


(d) End-Effector Configuration Tree



(e) Object Picking

(f) Object Placing



(g) Full Behavior Tree

Fig. 6: Task execution flow from ArUco detection to object placement.

7.1 Individual Task Testing

7.1.1 End-Effector Position Task

In this experiment, a predefined position is given to the end-effector while including base motion, under joint limit constraints. The robot's base and end-effector follow the desired path as shown in Fig. 7. The error initially starts high and gradually reduces to zero. The joint velocity profile reveals the dynamics required to reach the goal.

7.1.2 End-Effector Configuration Task

Here, a sequence of configuration targets is provided via behavior tree execution. The end-effector reaches each pose in sequence. The motion trajectory, joint velocities, and position errors are plotted in Fig. 8. As expected, the error decreases to zero as each configuration is reached.

7.2 Full Task Execution

7.2.1 Pick and Place using Dead Reckoning

In this task, pick and place are performed at predefined locations. The robot navigates using wheel odometry (dead reckoning). The base moves primarily along the x-axis as shown in Fig. 9. Position error spikes before each goal and decreases after reaching it. Joint velocities reveal the motion profile, with early peaks during the pick phase.

7.2.2 4) Pick, Transport, and Place using ArUco Feedback

Finally, the robot uses ArUco marker feedback to locate the object dynamically and perform pick-and-place. The trajectory Fig. 10 shows the robot moving to the detected marker and returning to the drop-off point. The error plot confirms convergence, and the velocity profile shows base and joint coordination.

8 Conclusion

This work presented the implementation of a task-priority-based control strategy for a mobile manipulator performing a pick, transport, and place operation. The system integrates kinematic control, behavior tree-based planning, dead reckoning navigation, and ArUco marker-based visual feedback.

The proposed method successfully demonstrated the execution of complex sequences while maintaining joint limits, smooth trajectory tracking, and real-time task coordination. Experimental results showed that the robot could accurately reach desired configurations, minimize end-effector error, and adapt to dynamically estimated object locations using visual input. The combination of recursive task-priority control and modular behavior planning offers a scalable and robust solution for mobile manipulation tasks in structured environments.

References

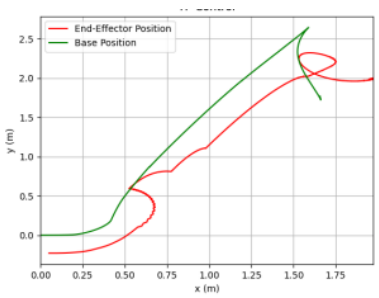
[1] P. Baerlocher and R. Boulic, "Task-priority formulations for the kinematic control of highly redundant articulated structures," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications*, 1998, p. 323–329.

[2] P. Cieślak *et al.*, "Practical formulation of obstacle avoidance in the task-priority framework for use in robotic inspection and intervention scenarios," *Robotics and Autonomous Systems*, vol. 124, p. 103396, 2020.

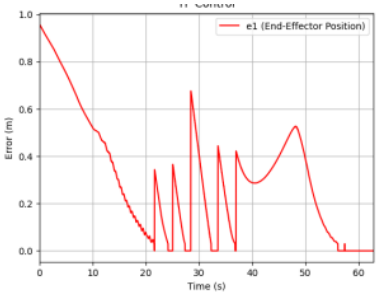
[3] P. Cieślak, "Hands-on intervention lecture slides," 2023, spring, University of Girona.

[4] Splintered-Reality, "Splintered-reality/py_trees: Python implementation of behaviour trees," https://github.com/splintered-reality/py_trees, 2023, accessed: 10 June 2023.

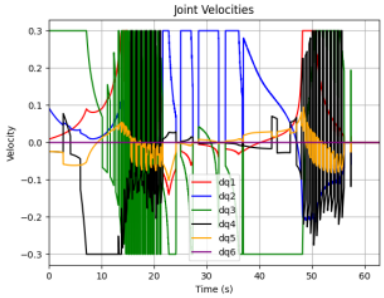
Click here Link to video
Click here Link to Github



(a) Trajectory of base and end-effector

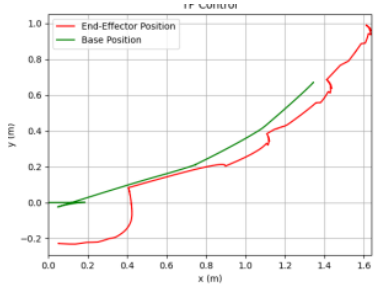


(b) End-effector position error over time

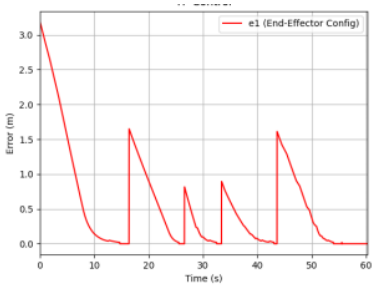


(c) Joint velocity profile

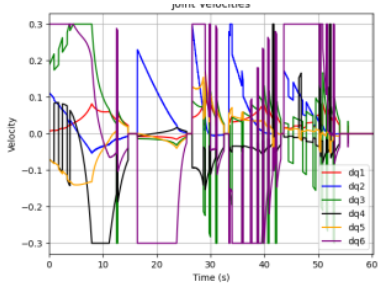
Fig. 7: Performance of End-Effector Position Task (section 7.1.1 Individual Task Testing)



(a) Trajectory of base and end-effector

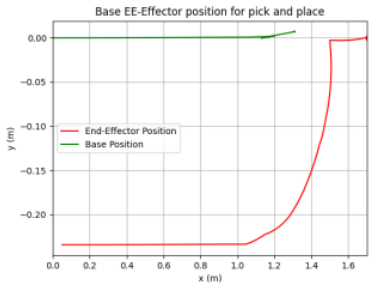


(b) Configuration error over time



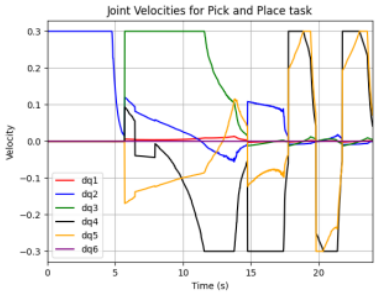
(c) Joint velocity profile

Fig. 8: Performance of End-Effector Configuration Task (section 7.1.2 Individual Task Testing)



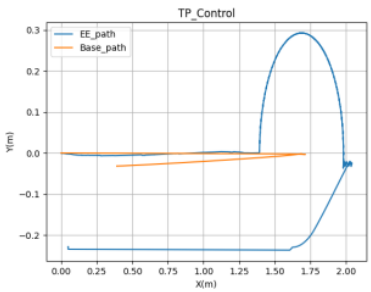
(a) Base and end-effector trajectory

(b) Position error over time

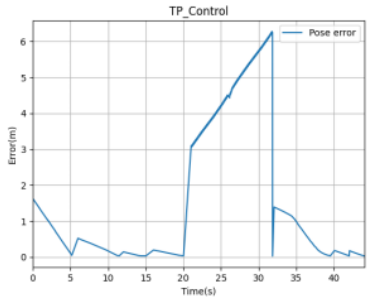


(c) Joint velocity profile

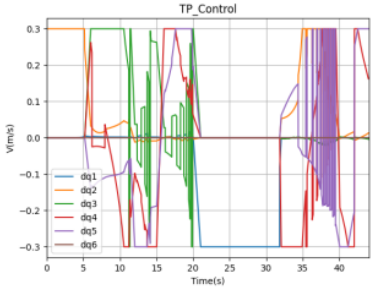
Fig. 9: Pick and Place Task using Dead Reckoning (section 7.2.1 Full Task Execution)



(a) Trajectory using visual servoing



(b) Joint velocities during task



(c) Position error convergence

Fig. 10: Pick, Transport, and Place using ArUco-Based Feedback (section 7.2.2 Full Task Execution)