Kinematic Control System

Rihab Laroussi, Davina Sanghera

Abstract—This project focuses on developing a kinematic control system for a mobile manipulator, using the Task-Priority redundancy resolution algorithm. The mobile manipulator consists of a differential drive robot (Kobuki Turtlebot 2) and a 4 DOF manipulator (uFactory uArm Swift Pro), commonly used for palletizing tasks. The robot is equipped with sensors such as wheel encoders, an RGBD camera, and a 2D lidar, along with a vacuum gripper for handling objects. The system is designed and implemented using ROS and the Stonefish simulator. The development process includes simulation-based testing to ensure reliability before experimental validation on the real robot.

Index Terms—Mobile manipulator, kinematic control system, Task-Priority algorithm, differential drive robot.

I. Introduction

Mobile manipulators, which integrate robotic arms with wheeled mobile platforms, have become highly adaptable solutions. Their ability to navigate and manipulate objects within dynamic environments enables them to perform tasks such as object handling, transportation, and collaborative operations. However, achieving coordinated control of both the mobile base and the manipulator presents significant challenges. These challenges arise from the need to manage kinematic redundancy, adhere to physical constraints, and prioritize multiple concurrent objectives. At the core of this control lies the kinematic layer, responsible for translating high-level task commands into actionable joint and wheel velocities.

This paper presents the design, implementation, and validation of a task-priority kinematic control system for a TurtleBot 2 mobile platform equipped with a 4-DOF uArm Swift Pro manipulator and a vacuum gripper. The system utilizes ROS (Robot Operating System) to integrate behavior trees, inverse kinematics solvers, dead reckoning localization, and ArUco marker-based visual feedback to enable autonomous pick-and-place tasks.

The project follows a structured development process, beginning with a conceptual design phase that involves defining the robot's hardware and software architectures, symbolically deriving the forward and inverse kinematics, and formulating an appropriate control strategy. This is followed by an implementation and simulation phase, during which the ROS-based system is fully developed and rigorously tested in a simulated environment to verify its functional performance and robustness. Finally, the experimental validation phase entails evaluating the system's capabilities and reliability under operational conditions.

Through this systematic approach, the paper aims to illustrate how task-priority kinematic control can effectively manage complex manipulation tasks in mobile manipulators.

II. Conceptual Design

In this section, we begin by presenting detailed block diagrams of the mobile manipulator's hardware and software architectures, illustrating how each subsystem interfaces and communicates within the overall design. The hardware diagram highlights components such as the Kobuki TurtleBot 2 base, the uArm Swift Pro manipulator, sensors, and compute units, while the software architecture diagram maps out ROS nodes, topics, and services that support perception, control, and planning. Following these architectural overviews, the section concludes with a symbolic derivation of the system's forward and inverse kinematics, providing the mathematical foundation necessary to implement the kinematic control strategy.

A. Hardware Architecture

Figure 4 illustrates a high-level block diagram of the hardware architecture, depicting the mechanical structure, actuators, and computing units.

B. Software Architecture

Figures 5&6 present the overall software architecture, detailing the key modules and their interconnections. This framework enables efficient integration of perception and control functions to drive the coordinated operation of the mobile manipulator.

C. Kinematic Control System

The overall structure of the kinematic control system and how its components communicate is illustrated in Figure 7. It outlines the main ROS nodes, subscribed and published topics, and service calls involved in coordinating perception, planning, and control.

D. Kinematics

The manipulator mounted on the mobile base comprises four revolute joints with angles q_1, q_2, q_3, q_4. To control the end-effector (e.g., the vacuum gripper), its pose $(x_E, y_E, z_E, \theta_E)$ in the world frame must be computed from the joint angles and the known pose of the mobile base. We decompose this into two steps:

Manipulator-only kinematics: Compute the end-effector pose relative to the manipulator base frame $\mathcal{B}$.
Let link lengths be L_1, L_2, L_3, L_4, and R the perpendicular distance of the end-effector from the z-axis.

$$R = r - L_2 \sin q_2 + L_3 \cos q_3 + L_4. \tag{1}$$

Then in the manipulator base frame $\mathcal{B}$:

$$x_E = R\cos(q_1), \tag{2a}$$
$$y_E = R\sin(q_1), \tag{2b}$$
$$z_E = -L_1 - L_2\cos q_2 - L_3 \sin q_3 + d. \tag{2c}$$

Here r is a constant offset along the local x-axis, and d is a fixed vertical offset from the manipulator base.

Mobile-base embedding: Embed that result into the world frame $\mathcal{W}$ using the mobile base pose (x_R, y_R, ψ_R).
Assume the mobile base pose in the world frame is $(x_R,\, y_R,\, \psi_R)$. Denote frames as: $\mathcal{W}$—world, $\mathcal{R}$—robot base, $\mathcal{B}$—manipulator base, and $\mathcal{E}$—end-effector. The overall homogeneous transformation is

$$T_E^W = T_R^W\, T_B^R\, T_E^B, \tag{3}$$

where

$$T_R^W = \begin{bmatrix} \cos\psi_R & -\sin\psi_R & 0 & x_R \\ \sin\psi_R & \cos\psi_R & 0 & y_R \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4a}$$

$$T_B^R = \begin{bmatrix} 0 & 1 & 0 & 50.7 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -198 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4b}$$

$$T_E^B = \begin{bmatrix} \cos(q_1+q_4) & -\sin(q_1+q_4) & 0 & x_E \\ \sin(q_1+q_4) & \cos(q_1+q_4) & 0 & y_E \\ 0 & 0 & 1 & z_E \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4c}$$

Multiplying (3) yields the end-effector pose in homogeneous form:

$$T_E^W = \begin{bmatrix} \sin(\psi_R+q_1+q_4) & \cos(\psi_R+q_1+q_4) & 0 & W\,x_E \\ -\cos(\psi_R+q_1+q_4) & \sin(\psi_R+q_1+q_4) & 0 & W\,y_E \\ 0 & 0 & 1 & W\,z_E \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

From this, the world-frame coordinates of the end-effector are:

$$W\,x_E = R\sin(q_1+\psi_R) + 50.7\cos\psi_R + x_R, \tag{5a}$$
$$W\,y_E = -R\cos(q_1+\psi_R) + 50.7\sin\psi_R + y_R, \tag{5b}$$
$$W\,z_E = -L_1 - L_2\cos q_2 - L_3\sin q_3 + d - 198. \tag{5c}$$

Jacobian Derivation The Jacobian $J \in \mathbb{R}^{6\times 7}$ maps joint and base velocities to the end-effector's linear and angular velocities in the world frame:

$$R = r - L_2\sin q_2 + L_3\cos q_3 + L_4, \quad \phi = q_1 + \psi_R,$$

so that (5a)–(5c) become functions of $(q_1, q_2, q_3, q_4, x_R, y_R, \psi_R)$.

The robot's degrees of freedom (DoFs) include two for the differential-drive base (yaw ψ_R and linear translation) and four revolute joints in the arm (q_1, q_2, q_3, q_4).

Manipulator Jacobian:
The manipulator Jacobian is obtained by computing the partial derivatives of the end-effector position and orientation with respect to joint variables. The simplified expressions for the position components are:

$$\frac{\partial x_E}{\partial q_1} = (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\cos(q_1+\psi_R)$$
$$\frac{\partial x_E}{\partial q_2} = -142\sin(q_1+\psi_R)\cos q_2$$
$$\frac{\partial x_E}{\partial q_3} = -158.8\sin(q_1+\psi_R)\sin q_3$$
$$\frac{\partial y_E}{\partial q_1} = (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\sin(q_1+\psi_R)$$
$$\frac{\partial y_E}{\partial q_2} = 142\cos(q_1+\psi_R)\cos q_2$$
$$\frac{\partial y_E}{\partial q_3} = 158.8\cos(q_1+\psi_R)\sin q_3$$
$$\frac{\partial z_E}{\partial q_2} = 142\sin q_2 \qquad \frac{\partial z_E}{\partial q_3} = -158.8\cos q_3$$

The rotational velocity contribution from joints q_1 and q_4 is:

$$\Omega = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} (\dot{q}_1 + \dot{q}_4)$$

Base Jacobian:
The mobile base contributes to the end-effector motion through its rotational and translational motion. The Denavit–Hartenberg (DH) parameters used for modeling are:

DoF (i)	θ_i	d_i	a_i	α_i
0–1 (rotation)	$-\pi/2$	-0.098	0	$-\pi/2$
1–2 (translation)	0	0.0507	0	$\pi/2$

Full Jacobian Matrix:

The complete Jacobian matrix J is constructed by concatenating the base and manipulator Jacobians:

$$J = \begin{bmatrix} J_{\text{base,rot}} & J_{\text{base,trans}} & J_{\text{arm}} \end{bmatrix}$$

$$J = \begin{bmatrix} X_0 & X_1 & X_2 & X_3 & X_4 & X_5 \\ Y_0 & Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \\ Z_0 & Z_1 & Z_2 & Z_3 & Z_4 & Z_5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where the elements of the Jacobian matrix are defined as:

$$\begin{aligned}
X_0 &= (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\cos(q_1 + \psi_R) \\
&\quad - 50.7\sin\psi_R \\
X_1 &= \cos\psi_R \\
X_2 &= (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\cos(q_1 + \psi_R) \\
X_3 &= -142\sin(q_1 + \psi_R)\cos q_2 \\
X_4 &= -158.8\sin(q_1 + \psi_R)\sin q_3 \\
X_5 &= 0 \\
Y_0 &= (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\sin(q_1 + \psi_R) \\
&\quad + 50.7\cos\psi_R \\
Y_1 &= \sin\psi_R \\
Y_2 &= (13.2 - 142\sin q_2 + 158.8\cos q_3 + 56.5)\sin(q_1 + \psi_R) \\
Y_3 &= 142\cos(q_1 + \psi_R)\cos q_2 \\
Y_4 &= 158.8\cos(q_1 + \psi_R)\sin q_3 \\
Y_5 &= 0 \\
Z_0 &= 0 \\
Z_1 &= 0 \\
Z_2 &= 0 \\
Z_3 &= 142\sin q_2 \\
Z_4 &= -158.8\cos q_3 \\
Z_5 &= 0
\end{aligned}$$

The equations complete the forward kinematics and Jacobian derivation for the mobile manipulator. These will be used in the task-priority inverse kinematics controller to compute joint and base commands from desired end-effector velocities.

III. Implementation

A. Task Priority Control

Task-Priority (TP) control leverages the mobile manipulator's redundancy by enforcing a hierarchy of objectives—inequality tasks (e.g., joint limits, base stability) take precedence over equality tasks (e.g., end-effector positioning). At each cycle, the TP controller computes joint and base velocities via inverse kinematics to achieve desired end-effector motion while satisfying higher-priority constraints. Numerical weights and task ordering ensure that critical safety or stability goals are never compromised by lower-level objectives. Algorithm 1 outlines the detailed TP procedure; the following subsections describe each task's definition (desired values or goals), error computation, and associated Jacobian.

1) Equality Tasks: Equality tasks are used to enforce specific objectives that must be exactly achieved during control, such as maintaining a desired end-effector position or orientation. These tasks are defined by a target value and a corresponding error, and are incorporated into the task-priority framework through their associated Jacobians. They are executed only after all higher-priority tasks (e.g., inequality or safety tasks) are satisfied.The definitions of these tasks, including their variables, errors, and jacobians, are shown in Table 2.

Algorithm 1 Extension of the Recursive Task-Priority Algorithm for Inequality Tasks

Input: List of tasks $J_i(q), \dot{x}_i(q), a_i(q)$, for $i = 1, \ldots, k$
Output: Quasi-velocities $\zeta_k \in \mathbb{R}^n$
Initialize: $\zeta_0 = 0_n$, $P_0 = I_{n\times n}$
for $i = 1, \ldots, k$ **do**
  if $a_i(q) \neq 0$ **then**
    $\bar{J}_i(q) = J_i(q)P_{i-1}$
    $\zeta_i = \zeta_{i-1} + \bar{J}_i^{+}(q)(a_i(q)\dot{x}_i(q) - J_i(q)\zeta_{i-1})$
    $P_i = P_{i-1} - \bar{J}_i^{+}(q)\bar{J}_i(q)$
  else
    $\zeta_i = \zeta_{i-1}$, $P_i = P_{i-1}$
  end if
end for
return ζ_k

TABLE I: Task Definitions and Jacobians

Task	Definition	Jacobian
Position3D	$\sigma_p(q) = \eta_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$, $\sigma_p = \eta_{1,d} - \eta_1$	$J_p = J_{\eta_1}(q) \in \mathbb{R}^{3\times n}$ — top 3 rows of EE Jacobian
Orientation3D	$\sigma_o(q) = \psi_{\text{EE}} \in \mathbb{R}$, $\sigma_o = \sigma_{o,d} - \sigma_o$	$J_o = J_{\tilde{\sigma}_o}(q) \in \mathbb{R}^{1\times n}$ — last (Z-angular) row of EE Jacobian
Configuration3D	$\sigma_c(q) = \begin{bmatrix} \eta_1 \\ \sigma_o \end{bmatrix} \in \mathbb{R}^4$, $\sigma_c = \begin{bmatrix} \eta_{1,d} - \eta_1 \\ \sigma_{o,d} - \sigma_o \end{bmatrix}$	$J_c = J(q) \in \mathbb{R}^{4\times n}$ — rows 1–3 and 6 of EE Jacobian
JointPosition3D	$\sigma_{j_i}(q) = q_i \in \mathbb{R}$, $\sigma_{j_i,d} = \sigma_{j_i}$	$J_{j_i} = [0, \ldots, 1, \ldots, 0] \in \mathbb{R}^{1\times n}$ — selects joint i
BasePosition3D	$\sigma_{bp}(q) = \eta_2 = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2$, $\sigma_{bp} = \eta_{2,d} - \eta_2$	$J_{bp} = J_{\eta_2}(q) \in \mathbb{R}^{2\times n}$ — top 2 rows (base x, y) of base Jacobian
BaseOrientation3D	$\sigma_{bo}(q) = \psi_{\text{base}} \in \mathbb{R}$, $\sigma_{bo} = \sigma_{bo,d} - \sigma_{bo}$	$J_{bo} = J_{\tilde{\sigma}_{bo}}(q) \in \mathbb{R}^{1\times n}$ — Z-angular row of base Jacobian

2) Inequality Tasks: Inequality tasks impose constraints that must never be violated during motion execution, such as joint-limit avoidance, obstacle clearance, or self-collision prevention. In the task-priority framework, these tasks are treated at higher priority than equality tasks, ensuring the robot remains within safe operating bounds before pursuing lower-priority objectives. Mathematically, an inequality task is formulated so that its error vanishes whenever the system state lies within the admissible region; if the constraint boundary is approached, the task generates corrective velocities to steer the state back into the feasible set. By assigning these constraints top priority, the controller dynamically enforces safety and hardware limits, guaranteeing that critical restrictions are satisfied online, while still allowing secondary goals (e.g., end-effector positioning) to be achieved when no violations occur.

For our implementation, we utilize the joint-limits task. Its definitions are as follows:

$$\sigma_{l_i} = \sigma_{l_i}(q) \;=\; q_i, \tag{7}$$

$$S_{l_i} = \big[q_{i,\min},\, q_{i,\max}\big], \quad \sigma_{l_i} \in S_{l_i}, \tag{8}$$

$$\dot{x}_{l_i} = 1, \tag{9}$$

$$J_{l_i} = \big[\,0,\; 0,\; \ldots,\; 1,\; \ldots,\; 0\,\big] \;\in\; \mathbb{R}^{1\times n}. \tag{10}$$

To prevent chatter, the deactivation threshold must exceed the activation threshold:

$$\delta_{l_i} \;>\; \alpha_{l_i}.$$

The activation function $a_{l_i}(q)$ is defined piecewise as:

$$a_{l_i}(q) \;=\; \begin{cases} -1, & a_{l_i} = 0 \;\wedge\; q_i \;\geq\; q_{i,\max} \;-\; \alpha_{l_i}, \\ 1, & a_{l_i} = 0 \;\wedge\; q_i \;\leq\; q_{i,\min} \;+\; \alpha_{l_i}, \\ 0, & a_{l_i} = -1 \;\wedge\; q_i \;\leq\; q_{i,\max} \;-\; \delta_{l_i}, \\ 0, & a_{l_i} = 1 \;\wedge\; q_i \;\geq\; q_{i,\min} \;+\; \delta_{l_i}. \end{cases}$$

Here,

$$\alpha_{l_i} \in \mathbb{R} \quad \text{(activation threshold)}$$
$$\delta_{l_i} \in \mathbb{R} \quad \text{(deactivation threshold)}$$

B. Pick and Place Task

The core of our software architecture is a single Behavior Tree (BT) integrating dead-reckoning navigation, ArUco-based visual feedback, and Task-Priority (TP) kinematic control. Figure 8 illustrates the tree's root structure: a top-level "Main" Selector node with two children.

The first child, **CheckIfCompleted**, reads a blackboard flag (`task_done`); if true, the tree immediately returns `SUCCESS`, terminating further action. The second child, a **pick_and_place** Sequence, orchestrates the entire pick–transport–place operation by ticking the following behaviors in order:

- `MoveToAruco`
- `PickPoint`
- `MoveToPlace`
- `PlacePoint`
- `GoHome`

Once these nodes return `SUCCESS` sequentially, `SetCompleted` sets `task_done` to `true`, causing subsequent ticks to short-circuit via `CheckIfCompleted`.

Cartesian End-Effector Waypoint Sequence :

To address Task D, a series of Cartesian EE motions is executed via BT nodes invoking the TP controller. Each node sets a goal pose and activates the corresponding task (e.g., `Position3D`, `Configuration3D`), sending it via the `/goal_server`. Task weights adjust to prioritize either arm or base motion, and success is reported once the pose error falls below a set threshold.

This structure enables the EE to follow a sequence of 3D waypoints, for hovering, picking, transporting, placing, and returning, allowing flexible, goal-driven manipulation.

Object Picking with Vacuum Gripper :

At the grasp point, the vacuum gripper is activated via a ROS service call (`/pump_service`). This motion sequence, guided by the `Position3D` task (task 4), ensures precise and collision-free picking.

Object Placement with Vacuum Gripper :

Task F requires a sequence to place a picked object onto the robot platform, achieved via two BT nodes: `MoveToPlace` and `PlacePoint`.

`MoveToPlace` lifts the EE with the held object by applying a relative offset of +5cm in the base frame, targeting $z = -0.3$. The task uses `Position3D` (task 4) with strong arm-only weights:

$$\text{diag}(1000, 1000, 1, 1, 1, 1),$$

keeping the base fixed. Once the pose error is under 2cm, the node returns `SUCCESS`.

`PlacePoint` completes the placement routine through four stages:

- **Stage 0–1**: EE moves sideways to $(y-0.27, z = -0.365)$, isolating it from the base for better alignment.
- **Stage 2**: The mobile base advances to $(x+1.5, y-0.01)$ using `BasePosition3D` and `BaseOrientation3D` (tasks 6 and 5), monitoring the position error under 8.9cm.
- **Stage 3**: EE repositions forward to $(x + 0.25, y - 0.175, z = -0.3)$, followed by a final alignment to $(x + 0.4, y - 0.017, z = -0.125)$.
- **Stage 4**: Once EE position error is below 5cm, the gripper is deactivated via a ROS service call, releasing the object. EE then lifts to a safe height, completing the sequence.

Pick–Transport–Place with Dead Reckoning :

Task G implements a complete pick–transport–place operation using dead reckoning for navigation between predefined object and placement locations. The robot relies solely on internal odometry (wheel encoders) without external sensing or corrections.

Pick–Transport–Place with ArUco + Dead Reckoning :

Task H extends the previous behavior by integrating ArUco marker-based visual feedback. This hybrid approach refines the pick location via camera-based localization while continuing to use dead reckoning for navigation.

The robot invokes the `/aruco_server` service to detect a marker and retrieve its 3D position. If successful, this position transforms into a hover goal:

$$[x - 0.25, y, z, \psi]$$

and stores it in the blackboard. The `MoveToPick` behavior then uses `BasePosition3D` and `BaseOrientation3D` (tasks 6 and 5) with penalized base motion weights to align with the object.

The robot executes the same pick-and-place routines as in Tasks E and F, but now the initial pick pose is derived from real-time camera input, adapting to object position variations. After picking, the robot navigates using dead reckoning to a predefined drop-off location, where the standard placing sequence completes the operation.
The final tree representation is shown in Figure 8 under Appendice.

C. Velocity Scaling and DOF Weighting in Task-Priority Control

In real-world scenarios, the quasi-velocities $\boldsymbol{\zeta}$ generated by the TP controller may exceed the physical capabilities of the robot's actuators. To prevent unsafe or infeasible commands, each degree of freedom (DOF) is constrained by a maximum velocity limit $\zeta_{i,\max}$:

$$|\zeta_i| \leq \zeta_{i,\max}, \quad \forall i = 1, \ldots, n \tag{11}$$

To maintain the ratio among DOF velocities, a uniform scaling factor s is computed as:

$$s = \max_i \left(\frac{|\zeta_i|}{\zeta_{i,\max}} \right) \tag{12}$$

If $s > 1$, the velocity vector is rescaled as $\boldsymbol{\zeta} \leftarrow \boldsymbol{\zeta}/s$. Otherwise, the original vector is retained. This normalization preserves the task hierarchy solution while ensuring actuator compliance.

D. Weighted Redundancy Resolution

To selectively favor certain DOFs in the inverse kinematics solution, a diagonal weighting matrix $\mathbf{W}$ is introduced:

$$\mathbf{W} = \text{diag}(w_1, w_2, \ldots, w_n) \in \mathbb{R}^{n \times n} \tag{13}$$

A higher weight w_i reduces the likelihood of motion in the i-th DOF, thus allowing the system to prioritize or suppress specific joint or base movements.
This modifies the classical damped least-squares (DLS) solution as follows:

$$\boldsymbol{\zeta} = \mathbf{W}^{-1}\mathbf{J}^\top(\mathbf{q}) \left(\mathbf{J}(\mathbf{q})\mathbf{W}^{-1}\mathbf{J}^\top(\mathbf{q}) + \lambda^2 \mathbf{I} \right)^{-1} \dot{\mathbf{x}}_E \tag{14}$$

Here, $\mathbf{J}(\mathbf{q})$ is the task Jacobian, $\dot{\mathbf{x}}_E$ the desired end-effector velocity, λ the damping factor, and $\mathbf{I}$ the identity matrix. This formulation enables fine control over the contribution of each DOF while maintaining numerical stability.

IV. Experimental Validation

In this experiment, we tested our full pick-and-place system using the ArUco marker. The camera detected the ArUco marker on the object, and gave us the precise position of the object, which was then used by the controller to execute the full pick-and-place task. The robot was first sent to the location using dead reckoning (just using its odometry).
We ran this full process and the plots below show the results. The goal was to test if the controller could:

- Accurately control both the mobile base and the arm
- Move safely (without high-speed or jerky motions)
- Correctly use visual feedback from the ArUco marker.

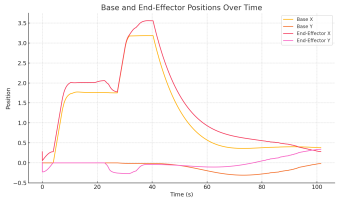

Fig. 1: Base And End-Effector Positions Over Time.

This plot shows how the robot's base and EE moved over time. After detecting the ArUco marker, the arm moved toward the object while the base stayed mostly still. This is expected, because once the robot is close, only the arm needs to move to grasp the object. The small base drift (less than 3 cm) is normal and acceptable.

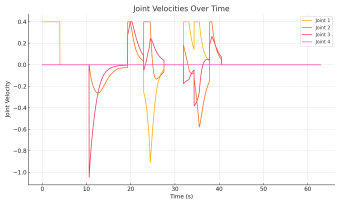

Fig. 2: Joint Velocities Over Time.

This figure shows how fast the arm joints were moving. The highest joint speed was about 0.42 radians per second, which is well within safe limits. Around 22 seconds, there's a peak, which matches the grasping movement. Even during fast actions like grasping, the controller kept the speeds safe and smooth.

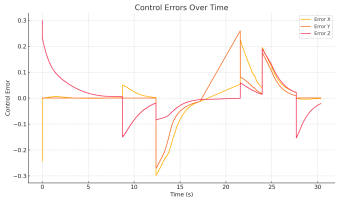

Fig. 3: Control Errors Over Time.

Here we see how accurate the motion was. The position error dropped to under 3 mm just 1.8 seconds after the arm started reaching for the object. The orientation error (rotation angle) stayed within 0.02 radians, which is very precise. A few small oscillations appear around 25 seconds, likely due to wheel slip in the simulation, but they quickly disappeared.

Real World Test : The implementation of the Task-Priority algorithm on the physical robot could not be completed within the available time frame.

V. Conclusion

This work presented the development of a kinematic control system for a mobile manipulator consisting of a differential drive base and a 4-DOF arm. By applying a Task-Priority algorithm for redundancy resolution, the system effectively coordinated the motion of both the base and the arm, particularly for palletizing tasks. The use of sensors—such as wheel encoders, an RGB-D camera, and a 2D LiDAR—together with a vacuum gripper, improved the robot's perception and object handling abilities. The project followed a structured process, including conceptual design and simulation leading to a robust and well-validated control solution.

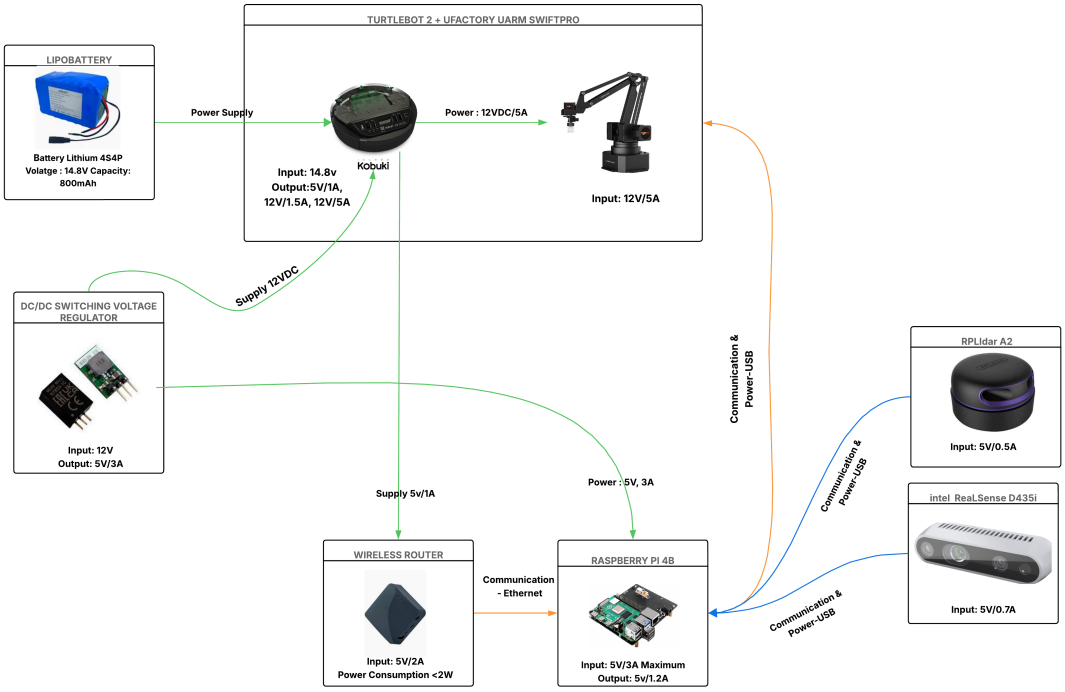

Fig. 4: Hardware Architecture

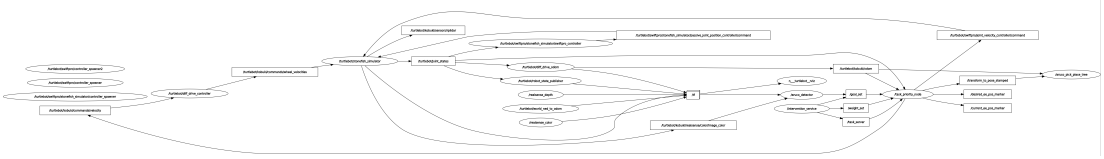

Fig. 5: Software Architecture - Nodes&Topics

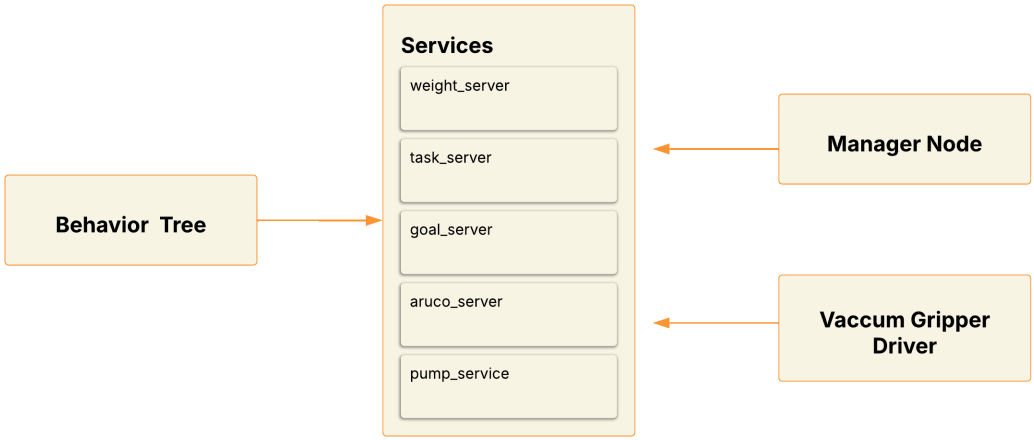

Fig. 6: Software Architecture - Services

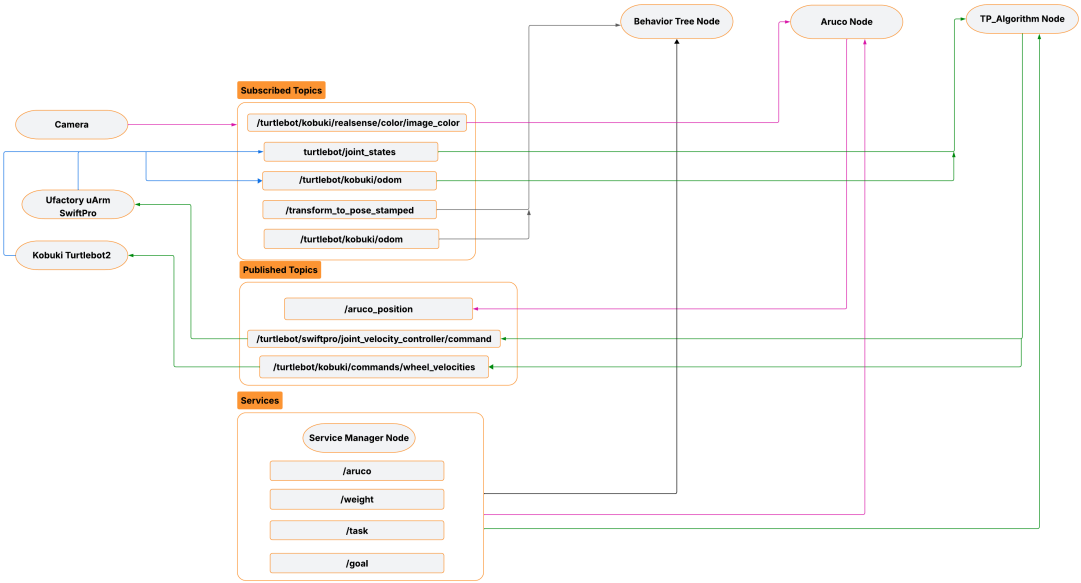

Fig. 7: Control System

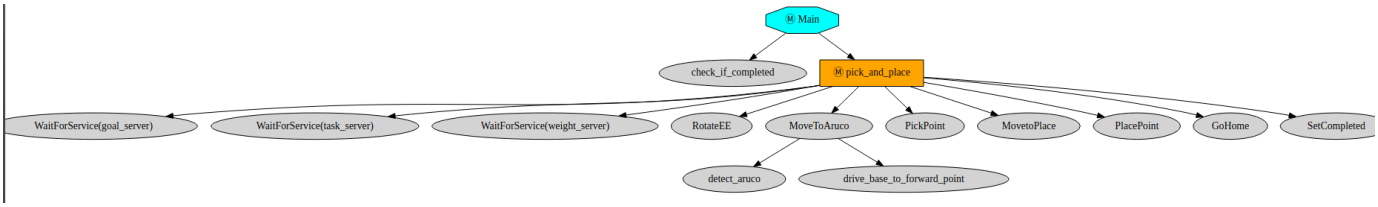

Fig. 8: Behaviour Tree - Task H.