Task-Priority Based Kinematic Control for a Mobile Manipulator

Solomon Chibuzo Nwafor[1], Enis Hidri[2], and Muhammad Faran Akram[3]

Computer Vision and Robotics Institute, University of Girona, Spain

Abstract— This paper presents a full task-priority control system for a mobile manipulator simulated in ROS 1. The robot combines a differential-drive Turtlebot base with a 4 DOF uArm Swift Pro. The controller solves a recursive velocity-level optimization problem that enforces strict priorities across equality and inequality tasks. Each task is defined by its Jacobian, target evolution, and activation state, then projected into the null space of higher-priority constraints. Control modules are executed as behavior trees, where motion is organized into pick-and-place pipelines. ArUco detection provides online goal acquisition, while dead reckoning handles fallback navigation. The solver integrates joint limits, Cartesian tracking, and base suppression into a single stack using damped least-squares resolution. Implementation covers five structured tasks. Results confirm safe actuation under joint saturation, consistent convergence to Cartesian goals, and stable velocity profiles. The system adapts to both predefined and vision-driven goals. Trajectories emerge from task composition and redundancy handling, without requiring precomputed paths. The approach generalizes to broader mobile manipulation applications needing safe online adaptation.

Keywords—Task-Priority Control, Mobile Manipulator, Recursive Kinematics, Behavior Trees, ArUco-Based Perception

1 INTRODUCTION

Mobile manipulators must resolve redundant actuation while satisfying joint and base constraints. This work implements a velocity-level controller using recursive task-priority resolution [1]. The system runs on ROS 1 and is tested on a simulated Turtlebot-SwiftPro platform.

Each control objective is defined as an equality or inequality task. These tasks are stacked by priority and solved in real time using a damped least-squares algorithm projected into the null space of higher-priority tasks [2]. Constraints include joint limits, base suppression, and Cartesian tracking.

Task execution is modularized into a behavior tree [3]. Nodes perform predefined or reactive motions. Goals include navigating to target poses, performing pick-and-place routines, and reacting to ArUco-based visual feedback.

The full stack integrates analytical Jacobians, forward kinematics, a prioritized control core, and perception modules into a unified runtime. Results are evaluated across five structured tasks by analyzing convergence, error regulation, and velocity behavior.

2 CONCEPTUAL DESIGN

2.1 Hardware Architecture

The mobile manipulator integrates a differential-drive Turtlebot 2 with a 4 DOF Swift Pro arm. A Raspberry Pi 4B manages sensor streams and actuation, while a 2D RPLidar A2 and an Intel RealSense D435i handle planar scanning and depth perception. Power is drawn from a 4S4P Li-Ion battery and distributed through a DC/DC voltage regulator. USB and serial interfaces connect the onboard modules, and a wireless router enables remote access. A complete overview of the physical layout and wiring is diagrammed in Appendix A.

2.2 Software Architecture (Simulation)

Simulation is handled through a ROS 1 environment built around the Stonefish simulator. Launching the system with `turtlebot_hoi.launch` brings up the base and arm descriptions, along with task-specific modules for control, localization, and detection. ROS topics and services link these nodes through standard channels for velocity commands, joint states, sensor feeds, and transform broadcasting. Appendix B provides a full view of the running graph and interface structure.

2.3 Control Architecture (Design)

Control is based on a velocity-level task-priority formulation. Each task defines an error, Jacobian, and target velocity, stacked by priority into a hierarchy. The controller solves this stack using a weighted damped least-squares method and outputs twist commands for the base and joint velocities for the arm. These commands are published in real time to ROS interfaces. The complete control structure, including the solver and message flows, is outlined in Appendix C.

Authors Emails: [1]u1999124@campus.udg.edu, [2]u1999254@campus.udg.edu, [3]u1999088@campus.udg.edu

2.4 Forward and Inverse Kinematics

Figure 1 shows the Swift Pro manipulator in its zero configuration, aligned to the North-East-Down frame convention. This frame affects joint direction: all revolute joints rotate opposite to standard robotics diagrams. The link dimensions and joint labels visible in the diagram form the reference structure for the forward kinematics and Jacobian derivations that follow.

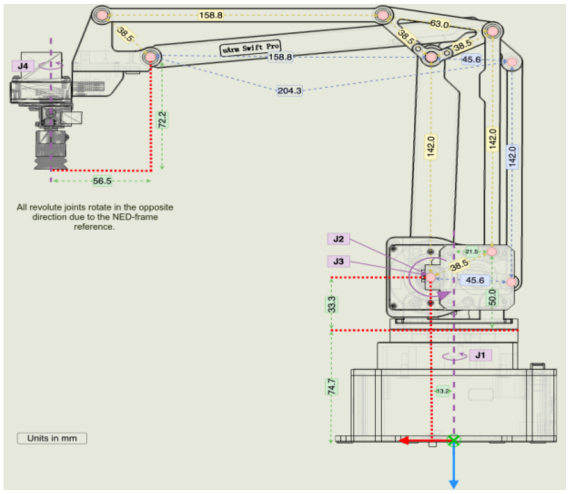

Figure 1: SwiftPro Dimensions

2.4.1 Forward Kinematics

The forward kinematics of the uArm SwiftPro are derived using a geometric formulation expressed in the NED base frame. The joint configuration vector is:

$$q = [\theta_1, \theta_2, \theta_3, \theta_4]^T \tag{1}$$

Relevant link dimensions and offsets are: $a_1 = 0.0132$, $a_2 = 0.1588$, $a_3 = 0.056$, $d_1 = 0.108$, $d_2 = 0.142$, $d_3 = 0.0722$

In the x-z projection shown in Figure 2, the scalar horizontal displacement d from the base to the wrist is given by:

$$d = a_2 \cos(\theta_3) - d_2 \sin(\theta_2) + a_3 + a_1 \tag{2}$$

This horizontal offset defines the planar position of the wrist, while vertical height is determined by joint angles and link offsets. The end-effector position in Cartesian coordinates becomes:

$$x_0 = d \cos(\theta_1) \tag{3}$$
$$y_0 = d \sin(\theta_1) \tag{4}$$
$$z_0 = d_3 - d_1 - d_2 \cos(\theta_2) - a_2 \sin(\theta_3) \tag{5}$$

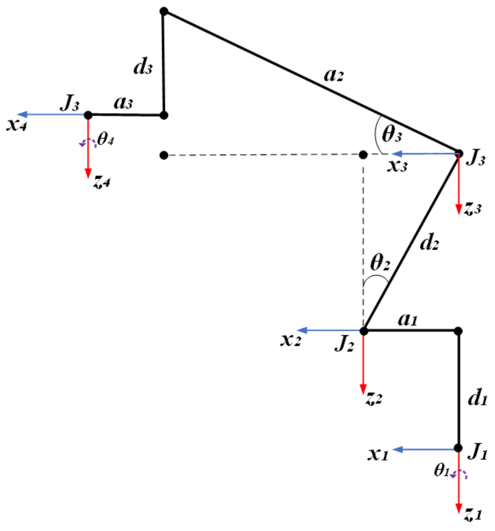

Figure 2: Projection of link geometry in the x-z plane

The x-y projection in Figure 3 clarifies how θ_1 determines base rotation about the vertical axis.

To incorporate end-effector orientation, a composite rotation about $\theta_1 + \theta_4$ is applied. The resulting transformation from the base to the end-effector is:

$$\mathbf{T}_{\text{EE}} = \begin{bmatrix} \cos(\theta_1+\theta_4) & -\sin(\theta_1+\theta_4) & 0 & x_0 \\ \sin(\theta_1+\theta_4) & \cos(\theta_1+\theta_4) & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

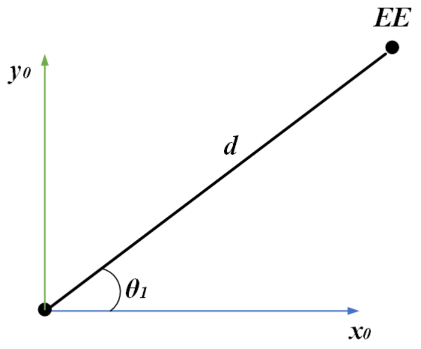

Figure 3: Top-down projection in the x-y plane

2.4.2 Analytical Jacobian

The linear velocity Jacobian $\mathbf{J}_v$ is computed from the partial derivatives of the end-effector position (x_0, y_0, z_0) with respect to each joint variable in the configuration vector:

$$q = [\theta_1, \theta_2, \theta_3, \theta_4]^T \tag{7}$$

For θ_1, which rotates the base in the horizontal plane:

$$\frac{\partial x_0}{\partial \theta_1} = -d\sin(\theta_1), \quad \frac{\partial y_0}{\partial \theta_1} = d\cos(\theta_1), \quad \frac{\partial z_0}{\partial \theta_1} = 0 \quad (8)$$

Differentiating the scalar horizontal displacement with respect to θ_2:

$$\frac{\partial d}{\partial \theta_2} = -d_2\cos(\theta_2) \quad (9)$$

$$\frac{\partial x_0}{\partial \theta_2} = -d_2\cos(\theta_2)\cos(\theta_1) \quad (10)$$

$$\frac{\partial y_0}{\partial \theta_2} = -d_2\cos(\theta_2)\sin(\theta_1) \quad (11)$$

$$\frac{\partial z_0}{\partial \theta_2} = d_2\sin(\theta_2) \quad (12)$$

For joint θ_3, contributing through a_2:

$$\frac{\partial d}{\partial \theta_3} = -a_2\sin(\theta_3) \quad (13)$$

$$\frac{\partial x_0}{\partial \theta_3} = -a_2\sin(\theta_3)\cos(\theta_1) \quad (14)$$

$$\frac{\partial y_0}{\partial \theta_3} = -a_2\sin(\theta_3)\sin(\theta_1) \quad (15)$$

$$\frac{\partial z_0}{\partial \theta_3} = -a_2\cos(\theta_3) \quad (16)$$

Joint θ_4 affects only orientation and does not contribute to the Cartesian position:

$$\frac{\partial x_0}{\partial \theta_4} = 0, \quad \frac{\partial y_0}{\partial \theta_4} = 0, \quad \frac{\partial z_0}{\partial \theta_4} = 0 \quad (17)$$

The linear velocity Jacobian becomes:

$$\mathbf{J}_v = \begin{bmatrix} -d\sin(\theta_1) & -d_2\cos(\theta_2)\cos(\theta_1) & -a_2\sin(\theta_3)\cos(\theta_1) \\ d\cos(\theta_1) & -d_2\cos(\theta_2)\sin(\theta_1) & -a_2\sin(\theta_3)\sin(\theta_1) \\ 0 & d_2\sin(\theta_2) & -a_2\cos(\theta_3) \end{bmatrix} \quad (18)$$

Each joint also contributes a fixed rotation axis to the angular velocity of the end-effector. In the base frame, the angular velocity Jacobian is:

$$\mathbf{J}_\omega = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad (19)$$

This corresponds to the following joint axes:

- Joint 1: rotates about the base Z-axis $\rightarrow [0,0,1]^T$
- Joint 2: local X-axis $\rightarrow [0,-1,0]^T$
- Joint 3: local X-axis $\rightarrow [-1,0,0]^T$
- Joint 4: again about base Z-axis $\rightarrow [0,0,1]^T$

Combining both components, the full analytical Jacobian is:

$$\mathbf{J}(q) = \begin{bmatrix} \mathbf{J}_v \\ \mathbf{J}_\omega \end{bmatrix} \quad (20)$$

2.5 Base Integration

All previous kinematic derivations have been formulated with respect to the manipulator base frame A. To express the end-effector pose in the world frame W, we must incorporate the known global pose of the mobile base: ${}^W x_R$, ${}^W y_R$, and ${}^W \psi_R$.

The vertical offset from the robot base to the manipulator base is given by a constant d_{base}, which is embedded in the transform ${}^R T_A$. The parameters used in this transform are derived from the Denavit-Hartenberg (DH) convention, as presented in Table 1.

Table 1: DH Parameters for Base-to-Arm Transform ${}^R T_A$

DOF	θ	d	a	α
1	$-\frac{\pi}{2}$	-0.198	0.0	$-\frac{\pi}{2}$
2	0.0	0.0507	0.0	$\frac{\pi}{2}$

The complete frame hierarchy follows: $W \rightarrow R \rightarrow A \rightarrow E$, where E is the end-effector.

Figure 4 shows a top-down view of the robot with its coordinate frame. The offset from the robot's geometric center to the manipulator base is 50.7 mm along the robot's x-axis.

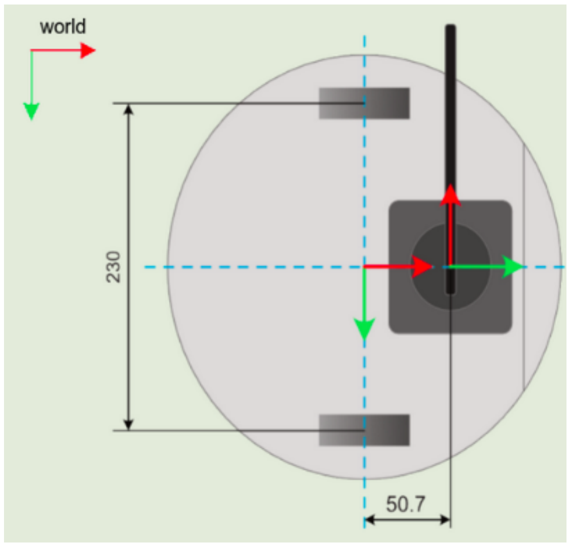

Figure 4: Top-down view of mobile base orientation and arm offset in world frame

The transformation from world to end-effector is expressed as:

$$^{W}T_E = {}^{W}T_R \cdot {}^{R}T_A \cdot {}^{A}T_E \tag{21}$$

Where:

- $^{W}T_R$ defines the robot base pose in world frame. - $^{R}T_A$ defines the static offset between robot and manipulator base. - $^{A}T_E$ is the transformation from manipulator base to end-effector, computed from forward kinematics.

Substituting each matrix:

$$\begin{aligned} ^{W}T_E = &\begin{bmatrix} \cos\psi_R & -\sin\psi_R & 0 & ^{W}x_R \\ \sin\psi_R & \cos\psi_R & 0 & ^{W}y_R \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\cdot \begin{bmatrix} 0 & 1 & 0 & 50.7 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -d_{\text{base}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\cdot \begin{bmatrix} \cos(\theta_1+\theta_4) & -\sin(\theta_1+\theta_4) & 0 & x_E \\ \sin(\theta_1+\theta_4) & \cos(\theta_1+\theta_4) & 0 & y_E \\ 0 & 0 & 1 & z_E \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{22}$$

The resulting position of the end-effector in world coordinates is:

$$\begin{aligned} ^{W}x_E = &(13.2 - 142\sin\theta_2 + 158.8\cos\theta_3 \\ &+ 56.5)\sin(\theta_1+\psi_R) + 50.7\cos\psi_R + {}^{W}x_R \end{aligned} \tag{23}$$

$$\begin{aligned} ^{W}y_E = &-(13.2 - 142\sin\theta_2 + 158.8\cos\theta_3 \\ &+ 56.5)\cos(\theta_1+\psi_R) + 50.7\sin\psi_R + {}^{W}y_R \end{aligned} \tag{24}$$

$$^{W}z_E = -108 - 142\cos\theta_2 - 158.8\sin\theta_3 + 72.2 - d_{\text{base}} \tag{25}$$

The Jacobian of the end-effector position with respect to both arm and base variables can be obtained by differentiating equations (23) to (25) with respect to each configuration variable:

$$\mathbf{J}_i(q) = \begin{bmatrix} \frac{\partial ^{W}x_E}{\partial q_1} & \frac{\partial ^{W}x_E}{\partial q_2} & \frac{\partial ^{W}x_E}{\partial q_3} & \frac{\partial ^{W}x_E}{\partial q_4} & \frac{\partial ^{W}x_E}{\partial q_b} & \frac{\partial ^{W}x_E}{\partial \psi_b} \\ \frac{\partial ^{W}y_E}{\partial q_1} & \frac{\partial ^{W}y_E}{\partial q_2} & \frac{\partial ^{W}y_E}{\partial q_3} & \frac{\partial ^{W}y_E}{\partial q_4} & \frac{\partial ^{W}y_E}{\partial q_b} & \frac{\partial ^{W}y_E}{\partial \psi_b} \\ \frac{\partial ^{W}z_E}{\partial q_1} & \frac{\partial ^{W}z_E}{\partial q_2} & \frac{\partial ^{W}z_E}{\partial q_3} & \frac{\partial ^{W}z_E}{\partial q_4} & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

All quantities are expressed in millimeters and should be converted to meters for calculations.

3 TASK PRIORITY CONTROL

A recursive kinematic controller is implemented using the task-priority redundancy resolution framework [4]. Tasks are assigned strict priorities, with higher-priority constraints satisfied first. Lower-priority tasks are projected into the null space of those above them.

Let $\zeta \in \mathbb{R}^n$ represent the quasi-velocity vector. Each task i is defined by its Jacobian $J_i(q)$, its desired evolution $\dot{\sigma}_i$, and its error $\tilde{\sigma}_i = \sigma_{i,d} - \sigma_i(q)$. Task activation is governed by a binary function $a_i(q) \in \{0,1\}$.

Algorithm 1 Recursive Task Priority Redundancy Resolution

```
Require: Task list $\{J_i, \tilde{\sigma}_i, a_i\}$
  Initialize: $\zeta_0 \leftarrow 0^n$, $P_0 \leftarrow I^{n\times n}$
  for $i = 1$ to $k$ do
    if $a_i \neq 0$ then
      $\bar{J}_i \leftarrow J_i P_{i-1}$
      $\zeta_i \leftarrow \zeta_{i-1} + \bar{J}_i^{\dagger}(\dot{\sigma}_i + K_i\tilde{\sigma}_i - J_i\zeta_{i-1})$
      $P_i \leftarrow P_{i-1} - \bar{J}_i^{\dagger}\bar{J}_i$
    else
      $\zeta_i \leftarrow \zeta_{i-1}$, $P_i \leftarrow P_{i-1}$
    end if
  end for
  return $\zeta_k$
```

3.1 Equality Tasks

Equality tasks encode features that the system must track exactly when unconstrained. They define direct mappings from configuration variables to Cartesian or internal robot goals, such as positions, orientations, or joint angles.

Each task is expressed as:

$$\dot{\sigma}_i = K_i(\sigma_{i,d} - \sigma_i(q)) \tag{27}$$

Here, $\sigma_i(q) \in \mathbb{R}^m$ is the current feature, $\sigma_{i,d}$ the desired value, and $J_i(q) \in \mathbb{R}^{m\times n}$ the task Jacobian. Equality tasks are solved recursively in priority order using the algorithm from Section III.

Table 2: Task-specific Definitions and Associated Jacobians

Task	**Mathematical Definition**	**Explanation**
Position	$\sigma_p = \eta_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \tilde{\sigma}_p = \eta_{1,d} - \eta_1,$	Tracks the Cartesian position of the end-effector.
Orientation	$\tilde{\sigma}_o = w\varepsilon_d - w_d\varepsilon - \varepsilon \times \varepsilon_d, \quad J_o = J_\omega$	Quaternion-based angular velocity error.
Configuration	$\sigma_c = \begin{bmatrix} \eta_1 \\ \varepsilon \end{bmatrix}, \quad \tilde{\sigma}_c = \begin{bmatrix} \eta_{1,d} - \eta_1 \\ \varepsilon_{\text{err}} \end{bmatrix},$	Tracks full 6D end-effector pose.
Joint Position	$\sigma_{ji} = q_i, \quad \tilde{\sigma}_{ji} = q_{i,d} - q_i, \quad J_{ji} = [0, \ldots, 1, \ldots, 0]$	Regulates individual joint angle.
Base Position	$\sigma_{bp} = \begin{bmatrix} x_b \\ y_b \end{bmatrix}, \quad \tilde{\sigma}_{bp} = \sigma_{bp,d} - \sigma_{bp},$	Tracks robot base's planar position.
Base Orientation	$\sigma_{bo} = \psi_b, \quad \tilde{\sigma}_{bo} = \psi_{b,d} - \psi_b, \quad J_{bo} = [0, \ldots, 0, 1]$	Yaw regulation of the base frame.

3.2 Inequality Tasks

Inequality or set-based tasks maintain a system variable within safe operational bounds. These tasks do not define target values to track, but instead enforce constraints. The most common inequality task is the joint limits task.

Task Variable:

$$\sigma_{li}(q) = q_i \tag{29}$$

Safe Set:

$$\mathscr{S}_{li} = [q_{i,\min}, q_{i,\max}] \tag{30}$$

The goal is to keep the joint position $\sigma_{li}(q) \in \mathscr{S}_{li}$. When the joint approaches the limits, a constant velocity is commanded to move it back toward the safe region:

$$\dot{x}_{li} = 1 \tag{31}$$

Jacobian:

$$J_{li} = [0, \ldots, 1, \ldots, 0] \in \mathbb{R}^{1\times n} \tag{32}$$

Activation Function:

$$a_{li}(q) = \begin{cases} -1, & a_{li} = 0 \wedge q_i \geq q_{i,\max} - \alpha_{li} \\ +1, & a_{li} = 0 \wedge q_i \leq q_{i,\min} + \alpha_{li} \\ 0, & a_{li} = -1 \wedge q_i \leq q_{i,\max} - \delta_{li} \\ 0, & a_{li} = +1 \wedge q_i \geq q_{i,\min} + \delta_{li} \end{cases} \tag{33}$$

Where:
- $\alpha_{li} \in \mathbb{R}$: activation threshold - $\delta_{li} \in \mathbb{R}$: deactivation threshold - $\delta_{li} > \alpha_{li}$: prevents chatter

This task is only triggered when the joint angle approaches the threshold. Once active, it remains enabled until the state moves back into the inner safe zone. These switching mechanisms ensure that joint limits are enforced without interfering with lower-priority control objectives.

3.3 Weighted Damped Least Squares

To regulate the contribution of each degree of freedom (DOF) during velocity control, a weighting matrix $W \in \mathbb{R}^{n\times n}$ is introduced in the damped least squares (DLS) solver.

Weighting Matrix

$$W = \text{diag}(w_1, w_2, \ldots, w_n)$$

Each diagonal entry w_i penalizes the corresponding DOF. A higher w_i reduces the influence of the i-th actuator. This allows the controller to prioritize arm motion over base motion, or vice versa.

Weighted DLS Formulation

Let $J(q) \in \mathbb{R}^{m\times n}$ be the Jacobian matrix and $\dot{x}_E \in \mathbb{R}^m$ the desired end-effector velocity. The weighted damped least squares solution is:

$$\zeta = W^{-1}J^T(JW^{-1}J^T + \lambda^2 I)^{-1}\dot{x}_E$$

where $\lambda \in \mathbb{R}$ is the damping factor and I is the identity matrix. This formulation penalizes costly DOFs and ensures numerical stability even when J is ill-conditioned.

This solver is applied at each control step in the task-priority hierarchy to compute the velocity command ζ.

4 IMPLEMENTATION

The system runs on ROS 1 with all modules built in Python. Each controller node implements a velocity-level task-priority stack, resolved recursively at runtime.

Behavior logic is structured as a tree, where nodes perform base and joint actions under hard constraints (joint limits) and soft goals (Cartesian targets). Motion is informed either by predefined coordinates or visual feedback.

4.1 Task D: End-Effector Navigation

This task sends the manipulator to predefined poses using full 6D tracking:

$$\tilde{\sigma}_c = \begin{bmatrix} \eta_{1,d} - \eta_1 \\ \varepsilon_{\text{err}} \end{bmatrix}, \quad \dot{\sigma}_c = K_c \tilde{\sigma}_c \qquad (34\text{–}35)$$

Two configurations alternate:

- Arm-only: $W = \text{diag}(10^8, 10^8, 1, 1, 1, 1)$
- Base-favored: $W = \text{diag}(100, 100, 1000, 1000, 1000, 1000)$

Each node runs until:

$$\|\tilde{\sigma}_c\|_2 < 0.01 \qquad (36)$$

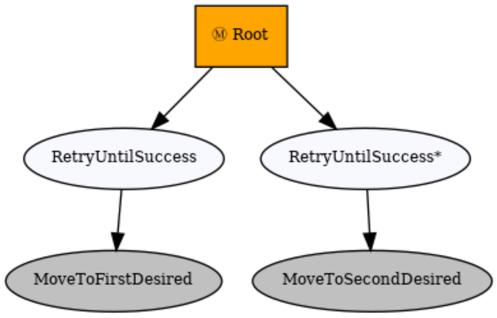

Figure 5: EE navigation sequence.

4.2 Task E: Pick Object

A single node handles both approach and lift using two stacked targets:

$$\sigma_a = \begin{bmatrix} x \\ y \\ z + z_{\text{offset}} \end{bmatrix}, \quad \sigma_l = \begin{bmatrix} x \\ y \\ z + z_{\text{offset}} + z_{\text{lift}} \end{bmatrix} \qquad (37\text{–}38)$$

Once aligned, a ROS service triggers suction. The final pose is saved to `lifted_pos`.

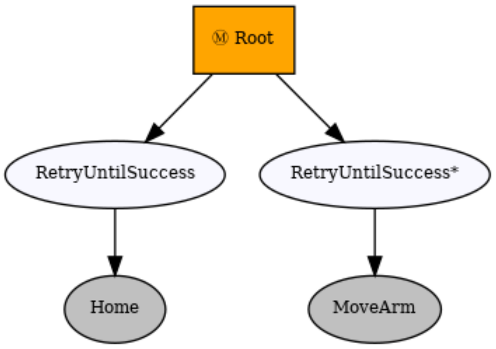

Figure 6: Pick sequence.

4.3 Task F: Pick and Place Object

This combines previous motions into a linear pipeline:

(i) MoveToHome

(ii) PickObject

(iii) MoveToDrop

(iv) DropObject

(v) HomeArm

All use position tracking:

$$\dot{\sigma} = K(\sigma_d - \sigma(q)), \quad J = J_v$$

Blackboard keys `lifted_pos` and `start_pos` maintain inter-task state.

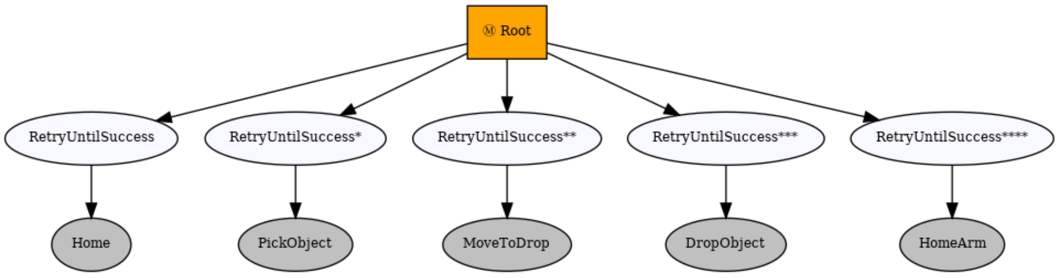

Figure 7: Pick and Place Object Sequence.

4.4 Task G: Pick-Transport-Place with Dead-reckoning

This variant drops sensing and memory. Pick and drop locations are hardcoded, and the robot navigates using velocity integration:

$$\begin{aligned} x_k &= x_{k-1} + V \cos\theta_{k-1}\Delta t \\ y_k &= y_{k-1} + V \sin\theta_{k-1}\Delta t \\ \theta_k &= \theta_{k-1} + W\Delta t \end{aligned} \tag{25}$$

The controller reuses the same task stack. Weight toggling suppresses base motion during manipulation.

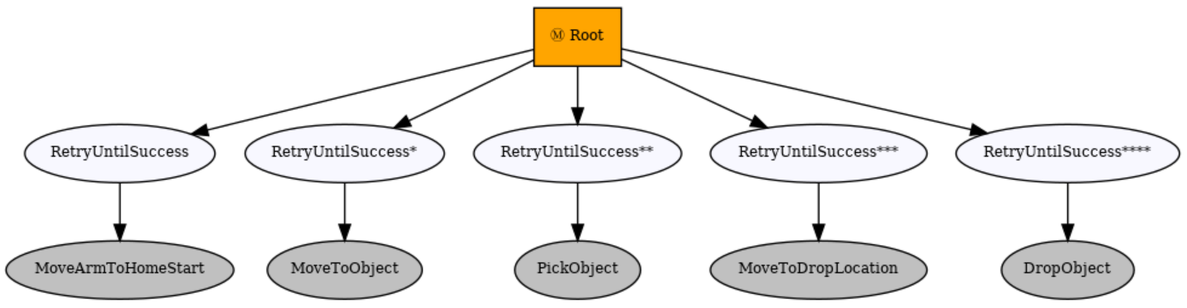

Figure 8: Pick, Transport and Place with Dead-reckoning Sequence

4.5 Task H: Pick-Transport-Place with Dead-reckoning and ArUco

This task adapts to runtime changes using ArUco detection. The marker pose is computed via solvePnP and transformed into the world frame:

$$T_{wm} = T_{wc} \cdot T_{cm}$$

The result is written to `aruco_goal`. From there, the base moves under the marker, and the arm executes a pick using the same phased control as Task E.

The robot retreats using a partial overwrite of `lifted_pos`. Drop execution is gated by `WaitForDropGoal`, which ensures the destination is known. The arm finally returns to home.

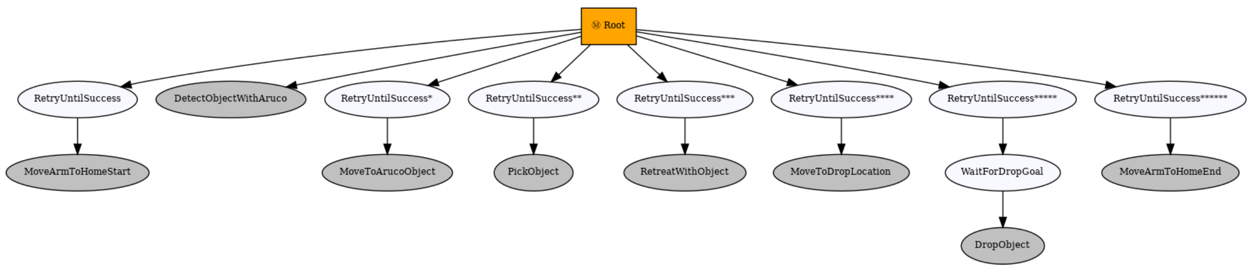

Figure 9: Pick, Transport and Place with Dead-reckoning and ArUco Sequence

5 RESULTS

5.1 Task D: End-Effector Position Navigation

This task moves the end-effector to a predefined Cartesian goal using only the arm. Base motion is suppressed by assigning high weights to the first two velocity components.

Figure 10 shows the result. The base trajectory is flat. The end-effector aligns forward in the x-direction with no lateral drift. All motion is resolved through the joints.

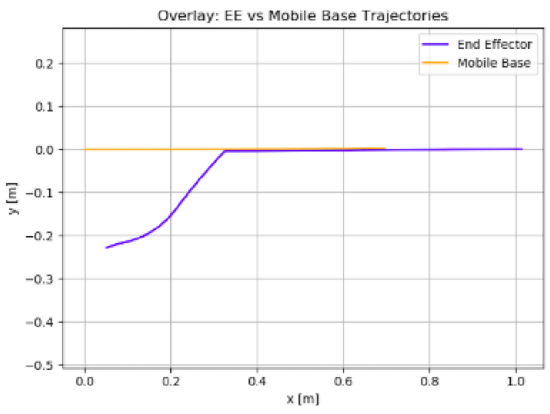

Figure 10: Overlay of end-effector and base trajectories

Figure 11 plots the error norms. Joint limit tasks are active from the start. They remain active through the entire motion. The end-effector configuration error decreases slowly. A sudden drop at $t = 25$ s marks convergence when it reaches home position.

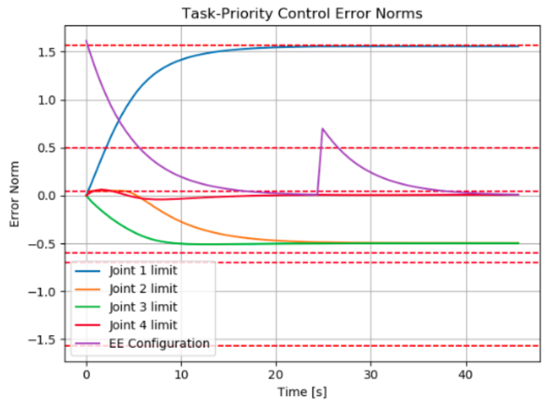

Figure 11: Task error norms. Joint limits stay active. EE error converges near $t = 25$ s.

Figure 12 shows the joint velocities. The base joints remain zero. The arm moves alone. Velocities decay smoothly. A spike in $t = 25$ s matches the drop in EE error and signals the task to move to another distressed position.

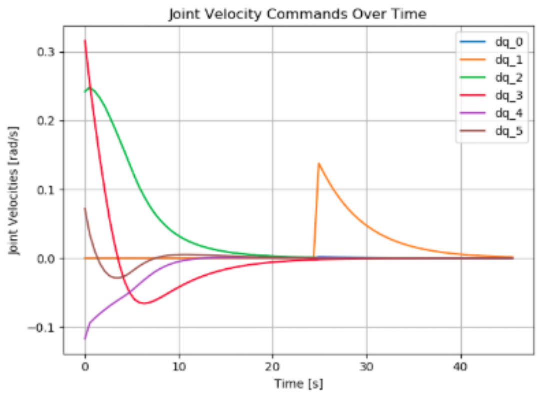

Figure 12: Joint velocities (Only arm joints are active)

The controller maintains safety through joint limit tasks.

5.2 Task E: Pick Object

This task requires a pick motion through two phases—approach and lift—resolved inside a single `MoveArm` controller. Position Figure 13 shows the Cartesian trajectories of the end-effector and base. The motion is fully attributed to the manipulator. The base remains stationary, confirming that the weighting configuration ($W = \mathrm{diag}(10^8, 10^8, 1, 1, 1, 1)$) successfully suppresses redundant base motion during pick execution.

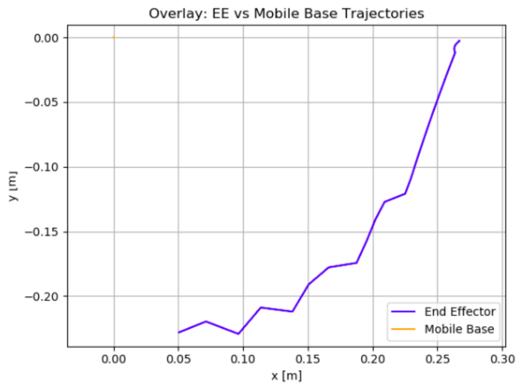

Figure 13: EE and base trajectories during pick.

Figure 14 plots the error norms of the position and joint-limits (first priority) tasks. The end-effector task starts around 0.25, decays steadily, and briefly spikes during lift reinitialization. This marks the transition from approach to lift phase.

Joint 1 gradually saturates its upper bound. The activation triggers as it crosses the 1.5 rad threshold. The system then enforces the constraint to prevent violation. Joint 2 follows a similar path, dipping toward the lower limit.

This dynamic interplay between the configuration objective and inequality tasks shows clearly in the switching events. These transitions are short and bounded, confirming stability of the null space projection.

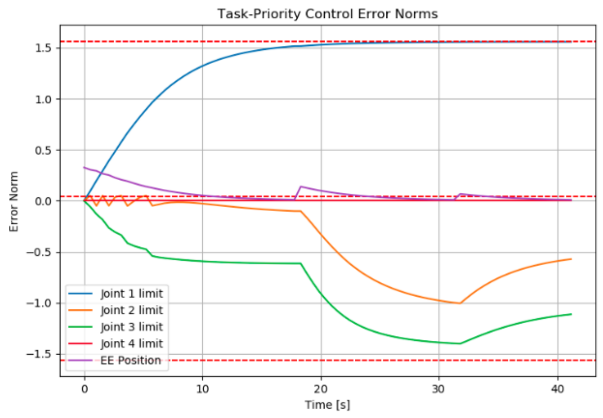

Figure 14: Error norms for joint limits and EE tracking.

The commanded velocities in Figure 15 reflect the transitions in task activation. At first, dq_3 fluctuates while dq_2 and dq_4 normalize gradually to keep the end-effector in the home position. Spikes are observed in joints 3 and 4 during lift reinitialization, aligning with a temporary increase in the error norm.

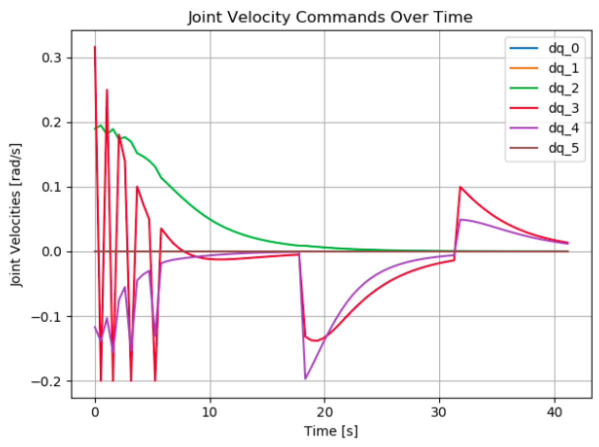

Figure 15: Joint and base velocity commands during pick.

The results confirm that the task-priority structure preserves the hierarchy: joint safety constraints are enforced while the Cartesian pick motion is tracked. The observed saturation and velocity reversals are characteristic of the activation policy described in Equation (33), validating the switching logic and gain tuning.

5.3 Task F: Pick and Place (Predefined Targets)

The robot picks an object at the front and drops it at the back using five sequential nodes. The arm executes this transfer under strict joint constraints. The base remains fixed (with more weighted DLS). All motion is resolved through the task-priority controller.

Trajectory

Figure 16 shows the end-effector path. The arc is formed as the arm repositions the object from the front to the rear. Since the robot has to move the end-effector from the front to the rear, the motion avoids colliding with the body by exploiting redundancy.

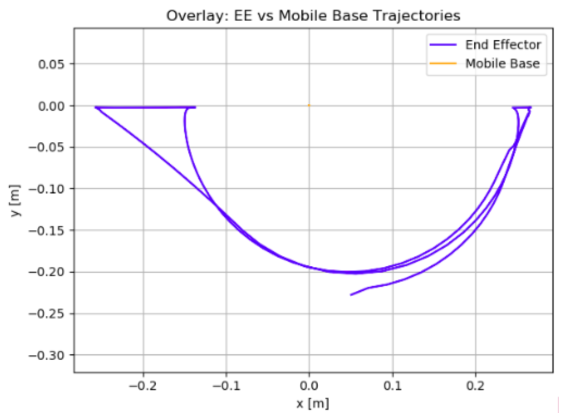

Figure 16: Arc-shaped EE trajectory from front pick to rear drop.

Error Norms

In Figure 17, Joint 1 limit tasks activate early and remain saturated. The joint hits both bounds at the two ends of the trajectory. This is expected: Joint 1 controls lateral reach and is critical for spanning the full arc. Joints 2 and 3 remain within bounds. The end-effector task is tracked throughout. Constraint satisfaction and task convergence co-exist under the recursive projection scheme.

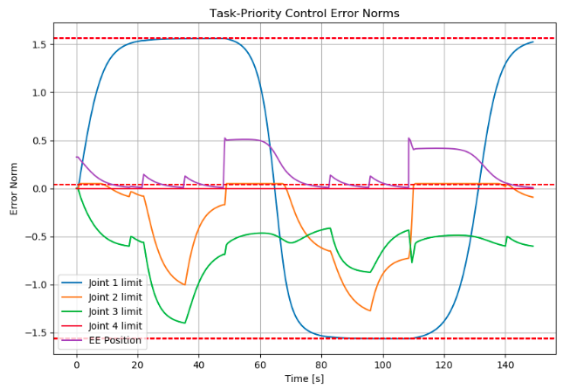

Figure 17: Joint 1 dominates constraint activity due to arc traversal.

Velocity Commands

Figure 18 shows the velocity response. Spikes on Joint 1 reflect limit avoidance and switching. The solver preserves continuity by shifting the load to other joints. Joints 3 and 4 show brief activation. No oscillations are observed. Velocity profiles are smooth between transitions. The controller regulates activation without instability.

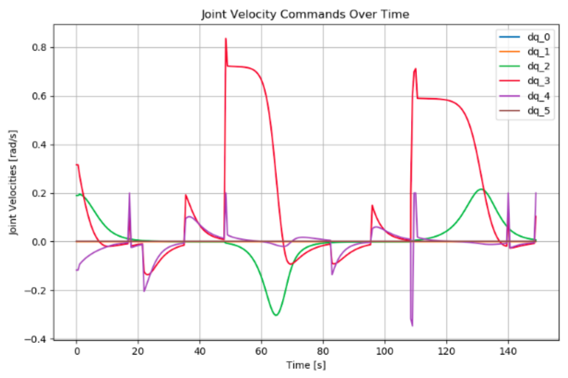

Figure 18: Joint velocities adjust to constraint activations.

The arc is shaped entirely by task hierarchy and joint limits. No trajectory was predefined. The end-effector tracks its goal by solving redundancy with safety as the primary constraint.

5.4 Task G: Pick-Transport-Place without ArUco

This experiment evaluates a full manipulation pipeline using only predefined targets. No ArUco detection was used. The robot followed a fixed sequence based on dead reckoning.

Figure 19 shows the overlay of end-effector and base positions. The arm executed pickup at the front, followed by a lift and a full backward reach to the rear. The base remained static for most of the trajectory, moving only during transport to the pick-up and drop locations. The end-effector followed the desired path without oscillation or deviation.

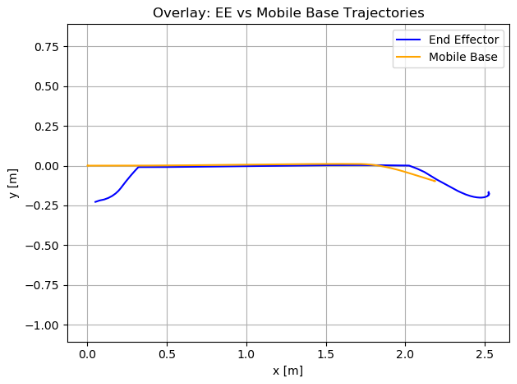

Figure 19: End-effector and base trajectory during Task G.

Figure 20 shows the task-space error evolution. Joint 1 operated close to its upper bound across the motion. This was expected, as the robot moved the arm from front where the upper limit is.

Joint 2 and Joint 3 enforced vertical lift and kept the arm within a safe space. End effector position error decayed to zero at each phase, confirming accurate convergence. Spikes in joint errors mark transitions between segments, where the active constraints switched.

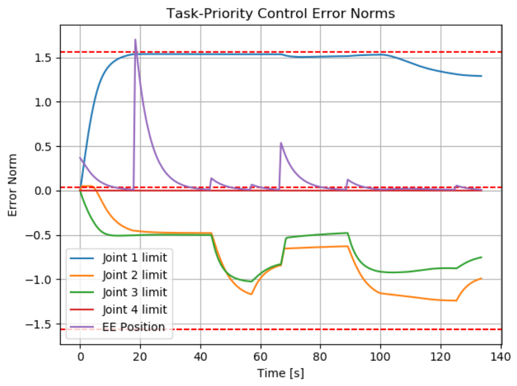

Figure 20: Error norms for joint limit tasks and end-effector position.

The joint velocities in Figure 21 show how the joints moved to achieve the task sequence. Peaks in dq_1 and dq_2 correspond to the pickup and drop phases, where lifting and lowering were dominant. dq_0 peaked only during transport.

dq_3 and dq_4 remained mostly inactive. Their small amplitudes show they were needed only for maintaining pose stability or resolving redundancy during transitions. The switching behavior is consistent with task prioritization.

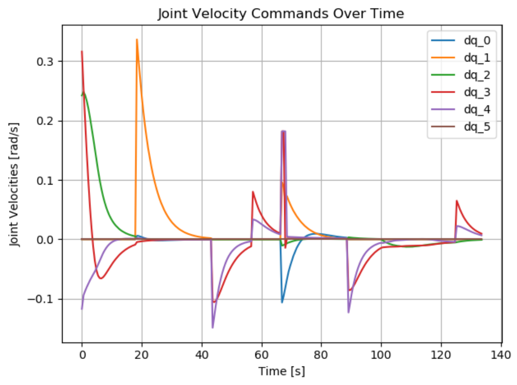

Figure 21: Joint velocity profiles over the course of Task G.

5.5 Task H: Visual Pick and Place Using ArUco Feedback

In this task, the mobile manipulator detects an object using ArUco marker. The robot starts at a known home pose, detects the goal pose from the image frame, transforms it to the world frame, and stores it on the blackboard. After picking the object, it checks for a drop ArUco pose. If none is provided, it returns to the previously saved start pose and drops the object.

Figure 22 shows the full trajectory. The end-effector path includes two curved segments corresponding to forward (home position) and backward (safe position) motions. The base trajectory is mostly linear but shows small deviations from the visual estimate due to ArUco detection error and control priority.

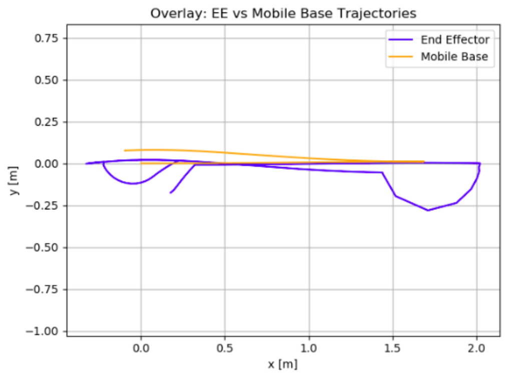

Figure 22: Task H trajectory: EE and base path during visual pick and place.

The joint error norms in Figure 23 show how joint 1 repeatedly approached the upper and lower thresholds during forward and backward base motions. This was caused by joint limit tasks suppressing base translation when the arm reached its kinematic range. Joint 4 spiked twice due to fast task switching around suction control. The EE position task converged reliably across each phase. The final drop phase showed a clean decay below the threshold.

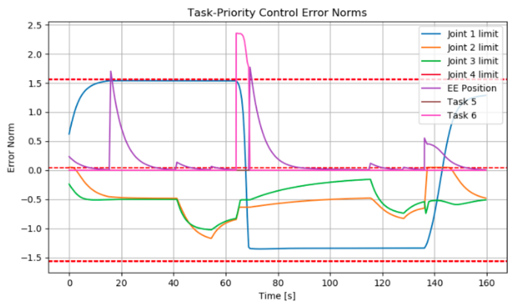

Figure 23: Error norms of EE and joint limit tasks.

Figure 24 presents the commanded velocities. Sharp velocity spikes occurred during suction activation, followed by a fast transition to the lift pose. The base velocity (dq_0 and dq_1) was suppressed while the end-effector tracked the ArUco-derived target. During return, the velocity profile flattened, with near-zero oscillations.

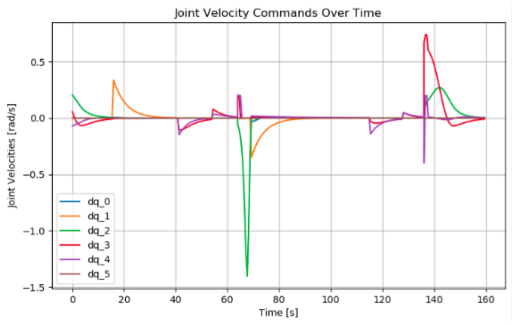

Figure 24: Joint and base velocity commands during Task H.

These results confirm that the system handled ArUco goal detection, real-time path correction, and task composition using recursive priority logic. Base and joint limits were respected, and the object was dropped at a dynamically inferred location, showing the integration of visual perception and task-level planning.

6 CONCLUSION

The controller solves constrained manipulation using velocity-level recursion. All motion is regulated through task prioritization, without trajectory preplanning. Joint limits are enforced throughout. Equality tasks guide the end-effector to each goal. Behavior trees modularize execution. ArUco detection enables visual reactivity. Dead reckoning ensures fallback execution. The system maintains feasibility even under saturation and joint reactivation. Results confirm consistent convergence, safe actuation, and dynamic adaptation across all task phases.

References

[1] A. Moe, J. J. Steil, and S. Haddadin, "Task-priority control of constrained mobile manipulators using a unified hierarchical quadratic program," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1772–1779, 2020.

[2] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009.

[3] M. Rocchi, A. Rocchi, A. Settimi, F. Ferri, and D. Nardi, "A flexible behavior-based control framework for mobile manipulators using task-priority inverse kinematics," *Frontiers in Robotics and AI*, vol. 7, p. 27, 2020.

[4] P. Cieslak. Hands-On Intervention Lecture Slides, Spring, 2025.

A HARDWARE LAYOUT

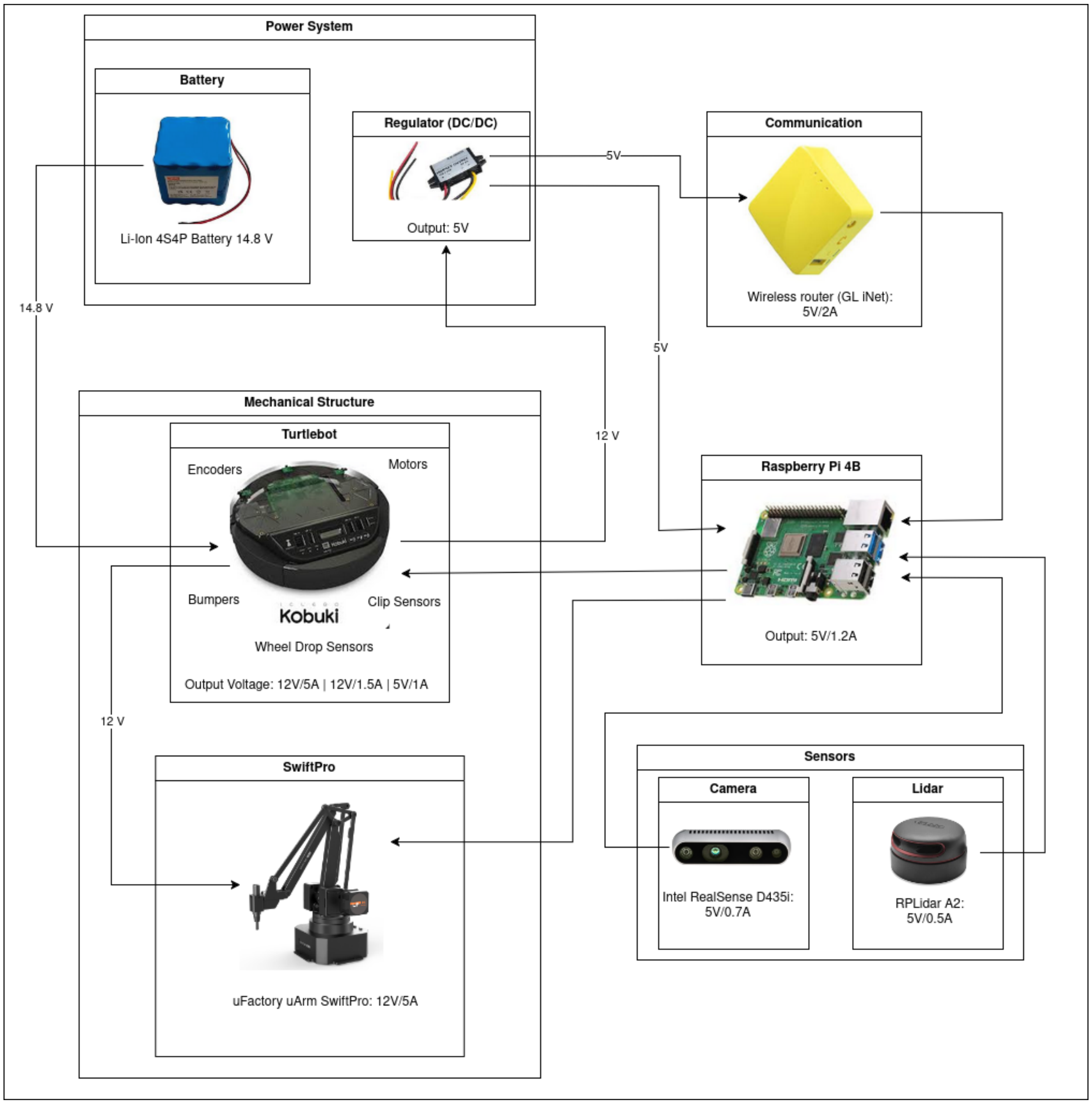

Figure 25: Hardware Architecture

B SOFTWARE LAYOUT

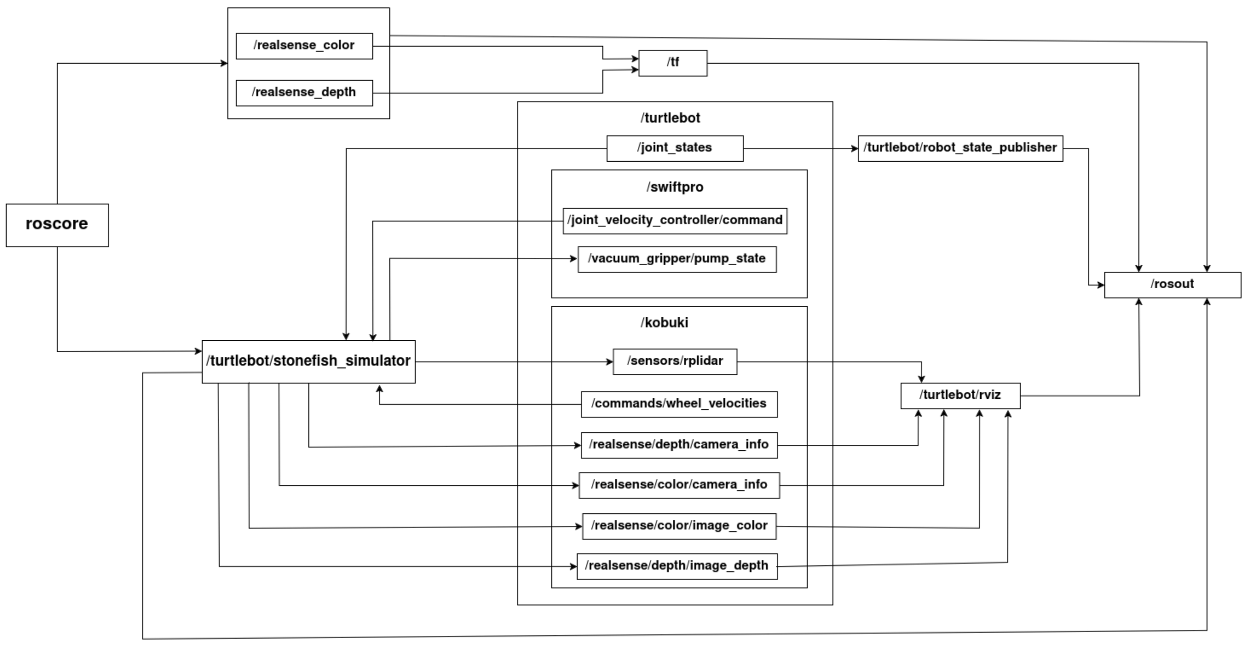
/realsense_color
/realsense_depth
/tf
/turtlebot
/joint_states
/turtlebot/robot_state_publisher
roscore
/swiftpro
/joint_velocity_controller/command
/vacuum_gripper/pump_state
/rosout
/kobuki
/turtlebot/stonefish_simulator
/sensors/rplidar
/turtlebot/rviz
/commands/wheel_velocities
/realsense/depth/camera_info
/realsense/color/camera_info
/realsense/color/image_color
/realsense/depth/image_depth

Figure 26: Simulation Software Architecture

C CONTROL LAYOUT

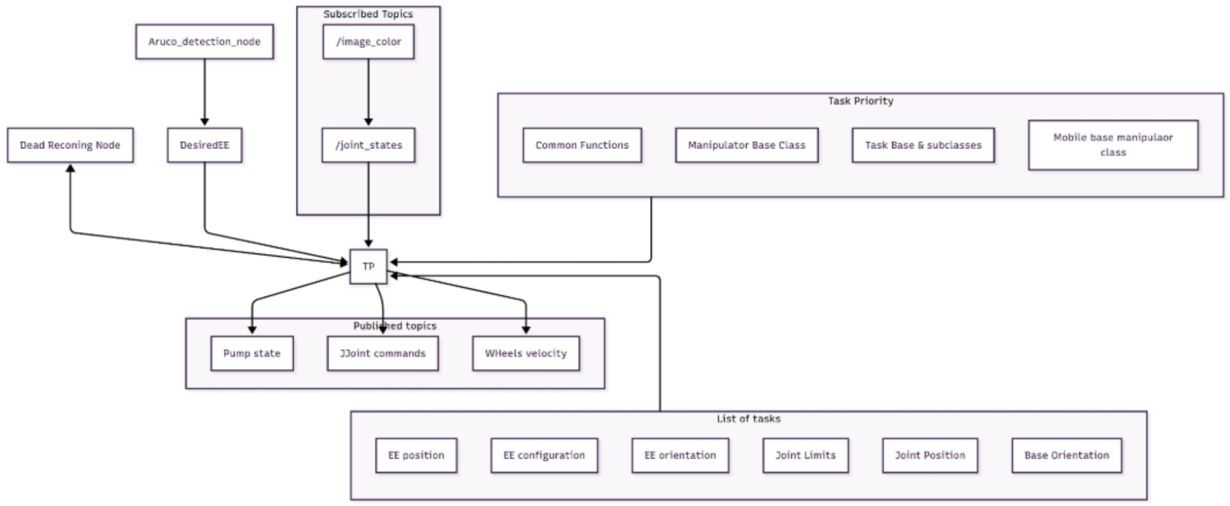

Figure 27: Control Architecture