SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Faculty of Information Engineering, Informatics and Statistics

Department of Computer, Control and Management Engineering (DIAG)

Autonomous & Mobile Robotics Project
Prof: Giuseppe Oriolo
Tutor: Francesco D'Orazio

Grasping on-the-move using a mobile manipulator

Lorenzo Cirillo 1895955
Lorenzo Cirone 1930811
Claudio Schiavella 1884561

June, 2024

Contents

1 Introduction

The rapid advancements in mobile robotics and reactive manipulation have unlocked new potential to enhance the efficiency and reliability of mobile manipulator robots operating in complex environments. Mobile reactive manipulation entails the capability of a manipulator robot to execute manipulation tasks while its base is in continuous motion toward subsequent targets. This report introduces a comprehensive architecture for reactive mobile manipulation on the move, drawing inspiration from the work of Burgess-Limerick [3]. In their paper, *"An Architecture for Reactive Mobile Manipulation On-The-Move"*, they proposed an innovative architecture that integrates and coordinates the control of the mobile base with that of the manipulator, enabling task execution while the robot is in motion. This approach significantly reduces cycle times compared to traditional methodologies, where the base halts during manipulation.

The primary objective of this project is to implement an on-the-move manipulation system for the TIAGo [7] mobile robot, designed to grasp static objects in a responsive and robust manner. The reactive nature of the proposed system enhances robustness against perception errors and environmental perturbations while also improving the accuracy of the robot's movements compared to open-loop trajectory planning approaches. The system's efficacy was validated through grasping and releasing tasks involving static objects, demonstrating superior task execution speed and smoother movements compared to existing techniques.

This report is structured as follows: Section 2 describes the TIAGo robot and its components. Section 3 offers a comprehensive overview of the entire control architecture, which comprises three key modules: the base controller, the arm controller, and the Quadratic Programming (QP) solver responsible for coordinating the base and arm motions. Section 4 reports the implementation details of these three modules. Section 5 presents simulation results that evaluate the performance of the entire architecture, accompanied by a thorough analysis of the findings. Finally, Section 6 summarizes the project's main results and discusses potential future directions for further system enhancements.

By implementing these three modules, the TIAGo robot can grasp objects on the move without halting its base, significantly reducing task completion time. This project underscores the significance of integrated control systems in advancing the capabilities of mobile robots in dynamic and unpredictable environments, paving the way for more efficient and reliable robotic applications.

2 Robot

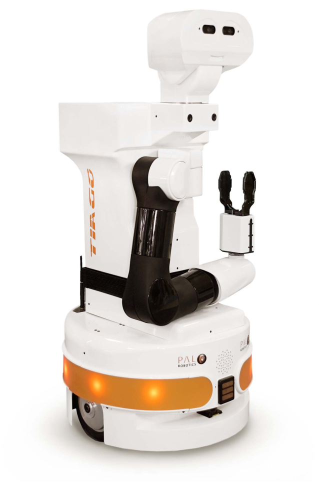

Figure 1: TIAGo robot equipped with the pal gripper: a two degree of freedom hand

The TIAGo robot [7], represented in Figure 1, is a robotic platform developed by PAL Robotics, a company specializing in creating service robots and humanoids. TIAGo, an acronym for Take It and Go, is a versatile robot designed to operate in home, office, and industrial environments. It offers advanced capabilities in manipulation, autonomous navigation, and interaction with humans. It is characterized by a mobile base that enables smooth navigation in complex spaces. Its modular structure includes a telescopic column that can vary the robot's height, facilitating interaction at different heights and manipulating objects in different environments. The robot has an articulated arm with seven degrees of freedom, terminating in a precision gripper or customizable EE. This allows it to perform various manipulation tasks, from gripping delicate objects to handling complex tools. The arm's precision and versatility allow operations in both domestic and industrial environments. The hardware architecture features advanced sensors, including RGB-D cameras, lidar, force and torque sensors, microphones, and voice-motion speakers. These sensors enable the robot to perceive and understand its surroundings, facilitating autonomous navigation tasks, object recognition, and effective human interaction. TIAGo can also exploit advanced artificial intelligence algorithms for different tasks.

The robot software is based on ROS (Robot Operating System), an open-source platform that provides a wide range of tools and libraries for robotic development. This makes the robot highly customizable and integrated with other technologies and systems.

3 Architecture

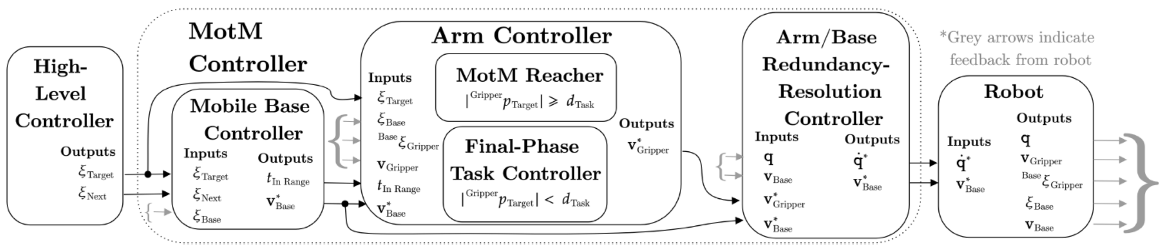
MotM Controller
High-Level Controller
Outputs
ξ_{Target}
ξ_{Next}
Mobile Base Controller
Inputs
ξ_{Target}
ξ_{Next}
ξ_{Base}
Outputs
$t_{\text{In Range}}$
$\mathbf{v}^*_{\text{Base}}$
Arm Controller
Inputs
ξ_{Target}
ξ_{Base}
${}^{\text{Base}}\xi_{\text{Gripper}}$
$\mathbf{v}_{\text{Gripper}}$
$t_{\text{In Range}}$
$\mathbf{v}^*_{\text{Base}}$
MotM Reacher
$|{}^{\text{Gripper}}p_{\text{Target}}| \geqslant d_{\text{Task}}$
Final-Phase Task Controller
$|{}^{\text{Gripper}}p_{\text{Target}}| < d_{\text{Task}}$
Outputs
$\mathbf{v}^*_{\text{Gripper}}$
Arm/Base Redundancy-Resolution Controller
Inputs
$\mathbf{q}$
$\mathbf{v}_{\text{Base}}$
$\mathbf{v}^*_{\text{Gripper}}$
$\mathbf{v}^*_{\text{Base}}$
Outputs
$\dot{\mathbf{q}}^*$
$\mathbf{v}^*_{\text{Base}}$
*Grey arrows indicate feedback from robot
Robot
Inputs
$\dot{\mathbf{q}}^*$
$\mathbf{v}^*_{\text{Base}}$
Outputs
$\mathbf{q}$
$\mathbf{v}_{\text{Gripper}}$
${}^{\text{Base}}\xi_{\text{Gripper}}$
ξ_{Base}
$\mathbf{v}_{\text{Base}}$

Figure 2: Controller full architecture

The modules implemented in the project compose the control architecture shown in the image 2. This approach presented in [3] allows the robot to reach a target to grasp in a reactive manner while its base moves. The redundancy resolution module coordinates grasping the target while the base moves, smoothly linking these tasks. The modules in this architecture have been slightly revised in this project to replicate performances and results as close as possible to those of the paper [3] with the available and limited resources. In particular:

- The mobile base controller was divided into two controllers: the first one takes care of approaching the robot to the grasping target, the second one is activated when the first one has finished its task and approaches the robot to the next target, i.e. the object where the grasping target will be released;
- The arm controller has been deepened by implementing reactive and non-reactive architectures.

- The redundancy resolution controller has a more general form, close to quadratic programming [4]

4 Implementation

In this section, we report the detailed implementation of the three modules introduced in Section 3. In particular, in Section 4.1, we describe the algebraic and geometric implementation that determines the foundation of the mobile base controller; in Section 4.2, we detail the arm controller schema together with different approaches, and in Section 4.3 a detailed mathematical description of the QP solver is reported.

4.1 Mobile Base Controller

The base controller is a critical component in the TIAGo robot's ability to perform tasks involving grasping and placing objects while moving. Its primary function is to navigate the robot to an optimal pose, termed closest approach pose (ξ_C), which places the robot within the manipulation range of the target object, allowing it to efficiently execute both the grasping and placing task.

The computation of the closest approach pose is essential for the grasping phase and involves several geometric parameters, as reported in Figure 3, including:

- Current base pose $\xi_B = (\xi_{B,x}, \xi_{B,y}, \theta)$;
- Grasping target position $\xi_T = (\xi_{T,x}, \xi_{T,y})$;
- Closest approach radius r_C;
- Distance between current base pose and closest approach pose ρ;
- Distance between current base pose and grasping target d
- Phase displacement between the robot's current forward direction α;

The closest approach pose is derived by computing the tangents from the current base pose ξ_B to a circle centered at the target pose ξ_T with radius r_C, with the intersection of these tangents with the circle determining two possible points, and ξ_C being the point closest to the robot's current position.

Given these elements, the values necessary for the base controller grasping phase are computed by geometric inspection.

The Euclidean distance d between the current pose ξ_B and the target pose ξ_T is computed as

$$d = \sqrt{(\xi_{T,x} - \xi_{B,x})^2 + (\xi_{T,y} - \xi_{B,y})^2}.$$

The distance ρ, the error in distance to ξ_C, is given by

$$\rho = \sqrt{d^2 - r_C^2}.$$

The angle α, defining the orientation error, is calculated as

$$\alpha = \mathrm{atan2}(\xi_{B,y} - \xi_{C,y}, \xi_{B,x} - \xi_{C,x}) + \pi - \theta.$$

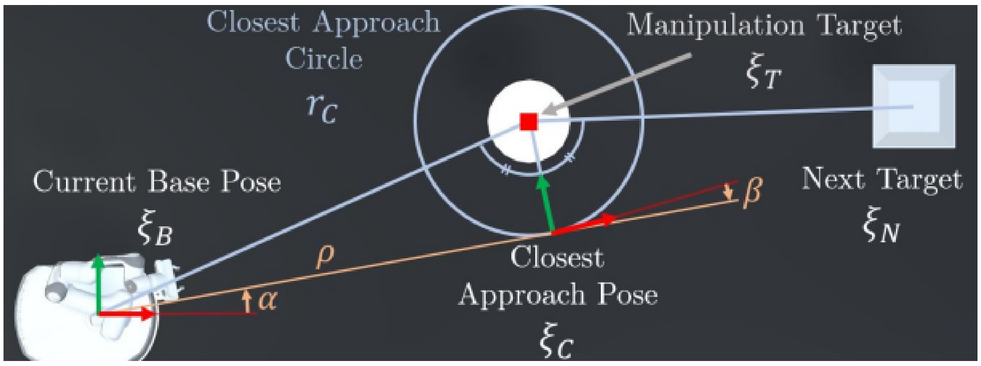

Figure 3: Geometric representation of closest approach target calculation.

From these values, the base controller utilizes the errors in distance (ρ) and orientation (α) to regulate the robot's forward (v_B) and steering (ω_B) velocities, defined respectively as

$$v_B = cost.$$
$$\omega_B = (k_\alpha \alpha)\frac{v_B}{\rho}. \tag{1}$$

where k_α is a gain factor ensuring stability, set to $k_\alpha = 4$ in our experiments, with the closest approach radius $r_C = 0.6m$ and $v_B = 0.3\frac{m}{s}$. Once the robot reaches the closest approach pose ξ_C, the controller shifts its focus to the placing target pose ξ_N, corresponding to the placing phase, based on the same principles used for determining ξ_C, and initiates the transition when the distance ρ falls below a certain threshold. In this case, the steering velocity becomes

$$\omega_B = (k_\beta \beta)\frac{v_B}{\rho_n}.$$

where k_β is the gain factor ensuring stability during the placing phase, set to $k_\beta = 2.2$ in our experiments, and β and ρ_n are respectively the orientation and distance errors between ξ_B and ξ_N.

The implementation of the base controller involves initializing parameters and thresholds, defining the robot's initial and target poses, computing the geometric calculations for distance and orientation, implementing the control laws for forward and steering velocities, continuously monitoring the robot's pose, shifting focus as needed, and tuning parameters to optimize performance. This ensures the stability and efficiency of the base controller, enabling the robot to perform grasping and placing tasks seamlessly while on the move.

4.2 Arm Controller

We decided to implement the arm controller in two ways and compare the behavior and results in them:

- **MoveIt Approach**: we use the inverse kinematic plugin of MoveIt.
- **Quintic Polynomial Approach**: trajectory planning with a quintic polynomial timing law.

4.2.1 MoveIt Approach

MoveIt is a robotic manipulation platform for ROS, which incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control, and navigation.

We use this plugin to move TIAGo's gripper grasping frame to a desired pose in Cartesian space. Since the mobile nature of TIAGo, we prefer to express the cartesian desired position concerning base_footprint frame; thus, we set up a simple transformation using *tf2_ros*.

Given a desired pose (e.g., the pose of the target ball) in Cartesian space, this plugin performs Inverse Kinematics, namely, computes the robot's joint configurations corresponding to that Cartesian pose. Since there is more than one configuration corresponding to the same pose, a planner is used to choose a configuration and find a plan to reach it.

The pipeline is the following:

1. Choose a group of joints (group_arm_torso).
2. Choose a planner and define the reference frame (/base_footprint in this case).
3. Project the grasping target position into the robot workspace
4. Set desired pose of /arm_tool_link encoded in a geometry_msgs.msg PointStamped.
5. Give time to find a plan.
6. Execute the plan if found

In this case, projecting the coordinates of the grasping target into the robot's workspace is necessary due to the MoveIt planners' non-reactivity. Using appropriate transformations, the planner has values that allow the arm's movement to be designed considering the target already reachable.

After transforming the coordinates of the grasping target from the universe frame to the TIAGo moving base frame, the projection takes place as follows

$$x_{proj} = x_B - \rho\cos(\alpha) + r_C.$$
$$y_{proj} = y_B - \rho\sin(\alpha).$$
$$z_{proj} = z_B.$$

Planners All the planners we used come from the same family of planners available in MoveIt, which is Open Motion Planning Library (OMPL).

OMPL is an open-source motion planning library that primarily implements randomized motion planners. MoveIt integrates directly with OMPL and uses the motion planners from that library as its primary default set of planners.

The planners in OMPL are abstract, so they do not know the robot. Instead, MoveIt configures OMPL and provides the back-end for OMPL to work with problems in Robotics.
Within OMPL, planners are divided into three categories:

- Geometric planners (Single-query, Multi-query, Optimizing planners)
- Control-based planners
- Multilevel-based planners

In particular, the planners we used rely on Single-query and Multi-query geometric planners.

Geometric planners only account for the geometric and kinematic constraints of the system. Assuming that any feasible path can be turned into a dynamically feasible trajectory, any geometric planner can be used to plan with geometric constraints.

Single-query planners typically grow a tree of states connected by valid motions. These planners differ in the heuristics they use to control where and how the tree is expanded. Some tree-based planners grow two trees: one from the start and one from the goal. Such planners will attempt to connect a state in the start tree with another in the goal tree.

On the other hand, **Multi-query planners** build a roadmap of the entire environment that can be used for multiple queries.

The planners we use are:

- **SBLK**: Single-query Bi-directional Lazy Collision Checking Planner is essentially a bidirectional version of EST (Expansive Space Tree) planning algorithm with Lazy state validity checking.

The EST algorithm relies on a probability distribution to guide tree growth, associating a weight w(q) with each tree configuration q. w(q) is an important estimate of choosing q as a tree configuration to add a new tree branch.

So the algorithm first selects a q in the tree with probability $\frac{w(q)}{\sum_{q' \in T} w(q')}$, then samples q_{near} a collision-free configuration near q and generates a path from q to q_{near}. If the path is collision-free, then it adds q_{near} and (q, q_{near}) to T [5].

The Bi-Directional EST instead grows two trees rooted at q_{init} and q_{goal} towards each other. Until the solution is found, the algorithm adds a new branch (using simple EST) to T_{init} and T_{goal}.
It then attempts to connect neighboring configurations from the two trees, and if it succeeds, it will return a path from q_{init} to q_{goal} [5].

The images in Figure 4 show how these two algorithms work.

Furthermore, Lazy state validity checking is a technique used in motion planning to delay verifying whether a particular state (position and configuration of the robot) is valid until it becomes necessary.

By valid, we mean if it satisfies all the constraints imposed on the robot's movement. These constraints include staying within the environment's boundaries, avoiding obstacles, and respecting joint limitations (MoveIt knows the workspace and the joint limits).

In the traditional approach, every state generated during the planning process is immediately checked for validity. This can be computationally expensive, especially in complex environments.

Lazy state validity checking instead postpones this verification until absolutely necessary. This can be done by checking validity only when the state needs to be connected to other states in the plan or deferring validation until a final path is generated, focusing on finding a promising path first.

Finally, the SBLK planner comprises a bi-directional EST algorithm with the lazy collision checking method.

- **RRT***: The Basic RRT (Rapidly exploring Random Trees) is a randomized tree that iteratively expands by attempting to make connections between a randomly spawned node q_{rand} in the state space and a node in a tree that is closest to q_{rand} (q_{near}) [6].

  First, initialize a tree rooted at q_{init} and until the solution is not found:

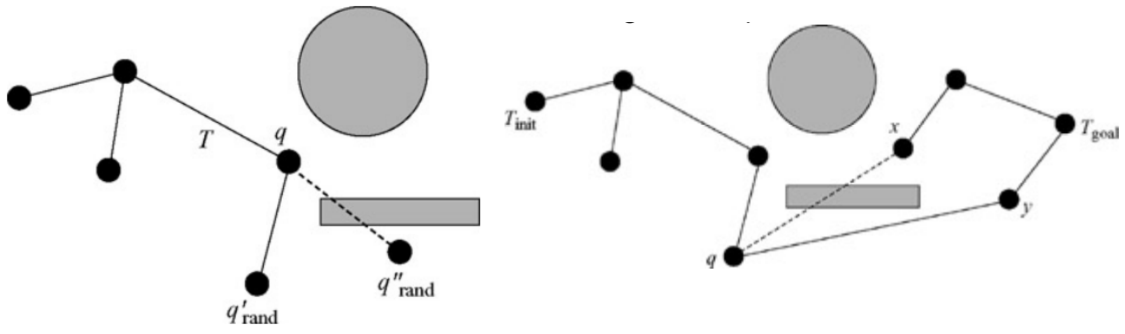

Figure 4: Example of EST (left) and Bidirectional EST (right) algorithm iteration

1. It focuses on a configuration from the tree and generates a random sample q_{rand} with a uniform probability distribution.
2. It finds q_{near}, the nearest configuration to q_{rand} according to a distance function.
3. It generates a path from q_{near} to q_{rand} (not necessarily collision-free)
4. It chooses a q_{new} at a distance of δ from q_{near} along the path to q_{rand}
5. If there is no collision from q_{near} to q_{new} then adds q_{new} to the tree.

The iteratively built tree rapidly covers the configuration space because the expansion is biased toward unexplored areas (Larger Voronoi regions).

Finally, RRT* is an asymptotically optimal version of RRT: the algorithm converges on the optimal path as a function of time. This was the first provably asymptotically planner (together with PRM), as figured out in Figure 5.

The core difference with basic RRT is that RRT* incorporates an additional step after tree expansion called ”rewiring” to improve the path’s optimality. This rewiring process ensures that the tree progressively converges towards a path with minimal cost (distance, time, etc.).

- **PRM***: The Basic PRM (Probabilistic RoadMap) algorithm works by creating a probabilistic roadmap, essentially a network of potential paths, within the robot’s environment. The pipeline is the following:

  1. Extract a sample q of the configuration space with a uniform probability distribution.
  2. Compute $\beta(q)$ (the volume of workspace occupied by the robot in q) and check for collisions.
  3. If $q \in C_{free}$ (the configuration space for no collisions), add q to the PRM; otherwise, discard it.
  4. Search the PRM for sufficiently near configurations q_{near}.

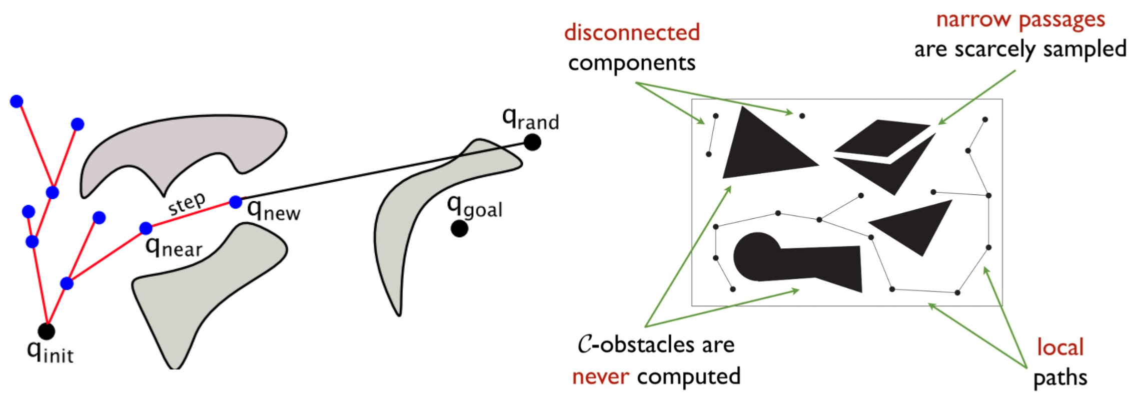

Figure 5: Example of RRT (left) and PRM (right) algorithm iteration

5. If possible, connect q to q_{near} with a free local path (through a local planner).

The construction of the PRM is arrested when:

– Disconnected components are less than a threshold;
– A maximum number of iterations is reached.

If q_{start} and q_{goal} can be connected to the same component, a solution can be found by graph search; else, enhance the PRM by performing more iterations.

Finally, the PRM* is an advanced version of the PRM algorithm. It incorporates additional steps to improve the path's quality.
Similar to RRT, it incorporates a rewiring step. The difference is in the connection strategy because while basic PRM connects each configuration to its nearest neighbors within a fixed distance, PRM* often employs a more sophisticated connection strategy such as K-Nearest Neighbors or Dynamic Radius (the radius used to find neighbors can vary based on the number of existing nodes in the roadmap).

Regular PRM attempts to connect states to a fixed number of neighbors while PRM* gradually increases the number of connection attempts as the roadmap grows in a way that provides convergence to the optimal path [1].

4.2.2 Quintic Polynomial Approach

The TIAGo arm controller plans a trajectory for the EE that will result in arrival at the target as the base enters the manipulation range. We use analytically calculated quintic polynomial trajectories, recalculated at each time step, enabling reactive control that responds to environmental disturbances and inaccurate robot control. The trajectory planning process is crucial for ensuring that the robot's movements are efficient, precise, and safe, avoiding any abrupt motions that could potentially disrupt the task or damage the robot. To this end, we employ a timing law $s(\tau)$ defined as a quintic polynomial, which provides a smooth transition between the initial and final positions. This timing law governs the end-effector (EE) progress along a desired linear, cartesian trajectory ξ_{des}. Using a quintic polynomial for the timing law, we ensure that the trajectory has continuous velocity and acceleration profiles, essential for smooth and controlled arm movements. This section details the mathematical formulation and implementation of the timing law and trajectory planning, highlighting the steps taken to achieve precise arm movements in the TIAGo robot.

First, we need the initial (ξ_i) and final (ξ_f) EE configurations. ξ_i is computed through the forward kinematics, while ξ_f is the target pose. We want the EE to perform a linear cartesian path. For this reason, the desired trajectory is given by

$$\xi_{des}(s(\tau)) = \xi_i + s(\tau)(\xi_f - \xi_i).$$
$$\dot{\xi}_{des}(s(\tau)) = \dot{s}(\tau)(\xi_f - \xi_i).$$
$$\ddot{\xi}_{des}(s(\tau)) = \ddot{s}(\tau)(\xi_f - \xi_i).$$

where $s(\tau) \in [0, 1]$ is the timing law expressed as a quintic polynomial as follows

$$s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3.$$
$$\dot{s}(\tau) = 30\tau^4 - 60\tau^3 + 30\tau^2.$$
$$\ddot{s}(\tau) = 120\tau^3 - 180\tau^2 + 60\tau.$$

Notice that

$$\tau = min(1, \frac{t}{T}) \qquad s(\tau) \in [0, 1].$$

with $T = 5$ and $t = 0$ incremented any iteration of $\Delta t = 0.1$ until $t < T$. Our control strategy is a control loop that updates the joint configuration until t is smaller than T.

The control loop is organized in the following schema by making the following computations:

- Initialize $t = 0, k = diag\{10, 10, 20\}\Delta t$.
- Forward kinematics to get ξ_i.
- Jacobian J, its pseudoinverse $J^\dagger$ and its time derivative $\dot{J}$.
- $s(\tau), \dot{s}(\tau), \ddot{s}(\tau)$ based on the current value of τ.
- $\xi_{des}, \dot{\xi}_{des}, \ddot{\xi}_{des}$ based on the current value of $s(\tau)$.
- Position error as $\Delta\xi = \xi_{des} - \xi_i$
- Control input $\dot{\xi}_r = \dot{\xi}_{des} + k\Delta\xi$.
- $\dot{q} = J^\dagger\dot{\xi}_r$.

- q with the Euler integration [2]. $q_{k+1} = q_k + \dot{q}\Delta t$.

The primary objective of this trajectory planning is to determine the joint positions q_{des} and velocities $\dot{q}_{des}$ that will bring the EE precisely to the desired final position. Calculating these values accurately during the control loop ensures that the EE follows the planned trajectory, achieving the smooth and controlled motion necessary for the task. The quintic polynomial timing law and linear Cartesian trajectory are designed to facilitate the computation of q_{des} and $\dot{q}$, ensuring that the EE reaches its target with the desired precision and stability.

4.3 Redundancy-Resolution Controller

The redundancy resolution controller transforms the desired EE and base velocities into robot joint velocities. This module ensures motion coordination between the arm and base and can exploit the redundant degrees of freedom to achieve secondary goals, such as avoiding joint position and velocity limits, and dynamically prioritizes base or arm velocities [4].

The objective is to obtain a module that allows the mobile base and the arm to act simultaneously, optimizing their movements. The base will tend to slow down when approaching the grasping target.
The optimization problem to achieve this is as follows:

$$\begin{array}{ll} \underset{x}{\text{argmin}} & f(x) = \frac{1}{2}x^T H x \\ \text{subject to} & \mathcal{J}x = \nu \\ & \mathcal{A}x \leq \mathcal{B} \\ & \mathcal{X}^- \leq x \leq \mathcal{X}^+ \end{array} \tag{2}$$

In our case study, the optimization variables in the objective function must take into account both base and joint velocity, as well as slack variables that will be discussed in more detail later in the section

$$x = (\omega_L \quad \omega_R \quad \dot{q}_a \quad \delta)^T \in \mathbb{R}^{n=n_w+n_a+n_\delta} \tag{3}$$

The dimensions of the values involved are

- **Joints:** $n_a = 7$
- **Wheel velocities:** $n_\omega = 2$
- **Slack variables:** $n_\delta = 4$
- **Task dimension:** $t = 3$

The velocity dimension of the wheels considers both in the vector $(\omega_L \quad \omega_L)$, while the task dimension is due to its positioning-only nature. From these considerations, the variable associated with costs is as follows

$$H = diag\{\lambda_q, \lambda_\delta\} \in \mathbb{R}^{n\times n} \tag{4}$$

Where the terms λ are such that

- The values associated with the joints consider a factor inversely proportional to the base error for angular velocities and a constant gain [4] for joint velocities

$$\lambda_q = \left(\frac{1}{\|\rho\|}, \frac{1}{\|\rho\|}, k_a, k_a, k_a, k_a, k_a, k_a, k_a \right), \qquad k_a \in \mathbb{R}. \tag{5}$$

- The values associated with the slack variables are considered a factor inversely proportional to the error of the EE

$$\lambda_\delta = \left(\frac{1}{\|\Delta\xi\|}, \frac{1}{\|\Delta\xi\|}, \frac{1}{\|\Delta\xi\|}, \frac{1}{\|\Delta\xi\|} \right). \tag{6}$$

In the optimization problem, equality constraints consider the following facts:

- The value of the EE velocity comes from direct kinematics and can be sacrificed by introducing a slack variable to improve coordination with base movement

$$J\dot{q}_a = \dot{\xi}_r - \delta_a, \quad \delta_a \in \mathbb{R}^t. \tag{7}$$

In this case, the reference velocity value of the EE is that of the closed system

$$\dot{\xi}_r = \dot{\xi}_{des} + k\Delta\xi \in \mathbb{R}^t. \tag{8}$$

- The value of the moving base speed is linked to the angular velocity of the two wheels, and this can be sacrificed with a new slack variable to improve coordination with the movement of the arm

$$\frac{\omega_L + \omega_R}{2} = \nu_r - \delta_\nu, \quad \delta_\nu \in \mathbb{R}. \tag{9}$$

In this case, the reference velocity value of the moving base ν_r is that constant of the paper [3].

Consequently, the equality constraint matrices are such that

$$\mathcal{J} = \left(\begin{array}{cc:c:c:c} \multicolumn{2}{c:}{\mathbf{0}_{t\times n_\omega}} & J_{t\times n_a} & \mathbb{I}_{3\times 3} & \mathbf{0}_{3\times 1} \\ \hdashline 0.5 & 0.5 & \mathbf{0}_{1\times n_a} & \mathbf{0}_{1\times 3} & 1 \end{array} \right) \in \mathbb{R}^{t+1\times n} \tag{10}$$

$$\nu = \left(\begin{array}{c} \dot{\xi}_r \\ \hdashline \nu_r \end{array} \right) \in \mathbb{R}^{t+1} \tag{11}$$

As far as inequality constraints are concerned, they concern the following considerations

- The future positions of the robotic arm joints, obtained by Euler integration from their velocity values, must fall within the mechanical limits of the system

$$q_{k+1} = q_k + \dot{q}\Delta t \in [q_{a,min}, q_{a,max}]. \tag{12}$$

- The values of the slack variables must be positive

$$\delta = (\delta_a \quad \delta_{\nu_r}) \geq 0. \tag{13}$$

Thus, the matrices for the inequality constraints are such that

$$\mathcal{A} = \left(\begin{array}{c|c|c|c} \mathbf{0}_{n_a \times n_\omega} & \mathbb{I}_{n_a \times n_a} & \mathbf{0}_{n_a \times 3} & \mathbf{0}_{n_a \times 1} \\ \hline \mathbf{0}_{n_a \times n_\omega} & \mathbb{I}_{n_a \times n_a} & \mathbf{0}_{n_a \times 3} & \mathbf{0}_{n_a \times 1} \end{array}\right) \in \mathbb{R}^{2n_a \times n} \tag{14}$$

$$\mathcal{B} = \begin{pmatrix} \frac{q_k - q_{a,min}}{\Delta t} \\ \frac{q_{a,max} - q_k}{\Delta t} \end{pmatrix} \in \mathbb{R}^{2n_a} \tag{15}$$

Finally, the upper $\mathcal{X}^+$ and lower $\mathcal{X}^-$ limits to the components of the optimization problem are placed on the values of the velocity and slack variables.

The problem posed in this way will be the input of a quadratic problem solver, which will return the solution that best satisfies the possibility of dynamically adapting the speed of the base with the movement of the arm [4].

5 Simulations

This section reports the simulations used to test the three mentioned controllers. The environment used for the simulations is very simple: just the target (i.e., a ball) placed on a post for the grasping phase and an empty box for the placing phase.

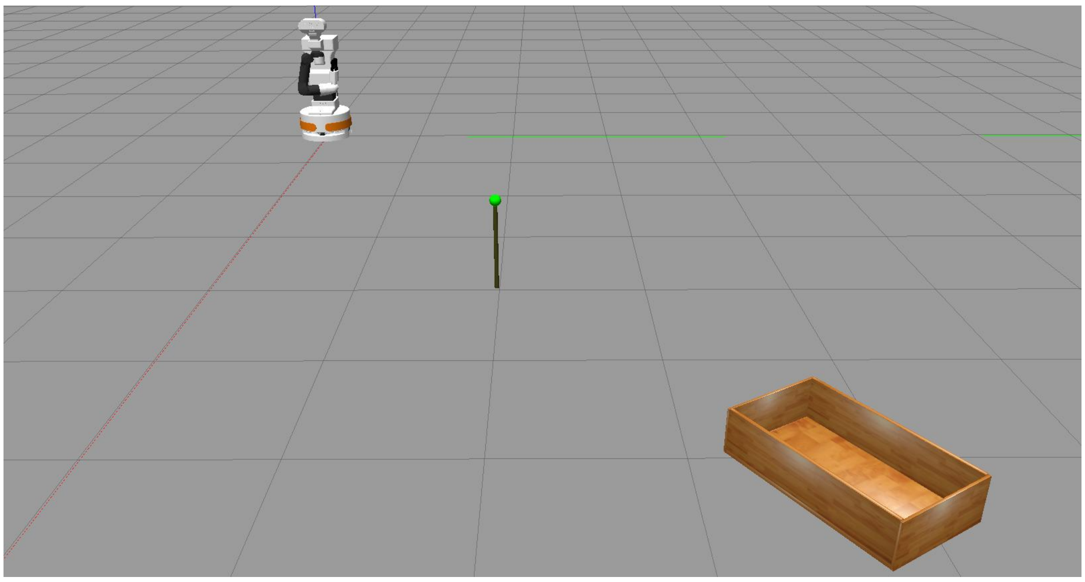

Figure 6: Simulations Gazebo environment

We consider a baseline consisting of the static grasping approach: the robot reaches the grasping zone around the target, stops, grasps the target, and moves towards the box to place it. This is compared with the grasping-on-the-move approach; namely, the robot has to achieve the grasping and placing phase without stopping by coordinating the base and arm movements. In this way, the simulation will be smoother, and its total time will decrease, meaning that the robot spends less time performing the entire task.

Therefore, this section is organized as follows: we analyze the on-the-stop simulations in Section 5.1 and the on-the-move simulations in Section 5.2. In both these sections, we divide our simulations into Mobile Base with MoveIT Planners and Mobile Base with Quintic Polynomial. In particular, we refer to the Quintic Polynomial approach with the QP Solver as QPP. In the end, in Section 5.3, we compare each method's simulation times and success rate between on-the-stop and on-the-move approaches.

5.1 On-The-Stop

This section analyzes the on-the-stop simulations in which the robot reaches the closest approach point, stops, grasps the ball, and moves toward the box.

5.1.1 Mobile Base with MoveIT Planners

In Figure 7 it is reported the mobile base trajectory of the robot. It can be appreciated that the robot can reach the closest approach points of both the ball and the box, making the grasping and placing phases easier to achieve.

We report the mobile base velocities with the error metrics in Figures 8. We can notice that the steering velocity (ω_b) is zero when α is also zero (as described in 1), and then it is again zero when β is also zero. Moreover, the robot reaches ξ_C when ρ is almost zero (not exactly zero because of the threshold), while it reaches ξ_N when ρ_n is almost zero (not exactly zero always because of the threshold). Moreover, ρ_n and β are constant for a specified period (i.e., when the robot stops moving the arm towards the ball). This is because the robot is stationary, and consequently, the error metrics regarding ξ_N (i.e., the closest approach point of the box) cannot change over time.

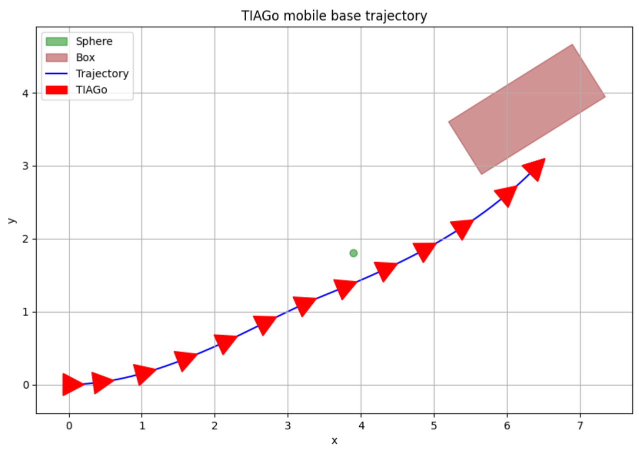

Figure 7: Base controller trajectory

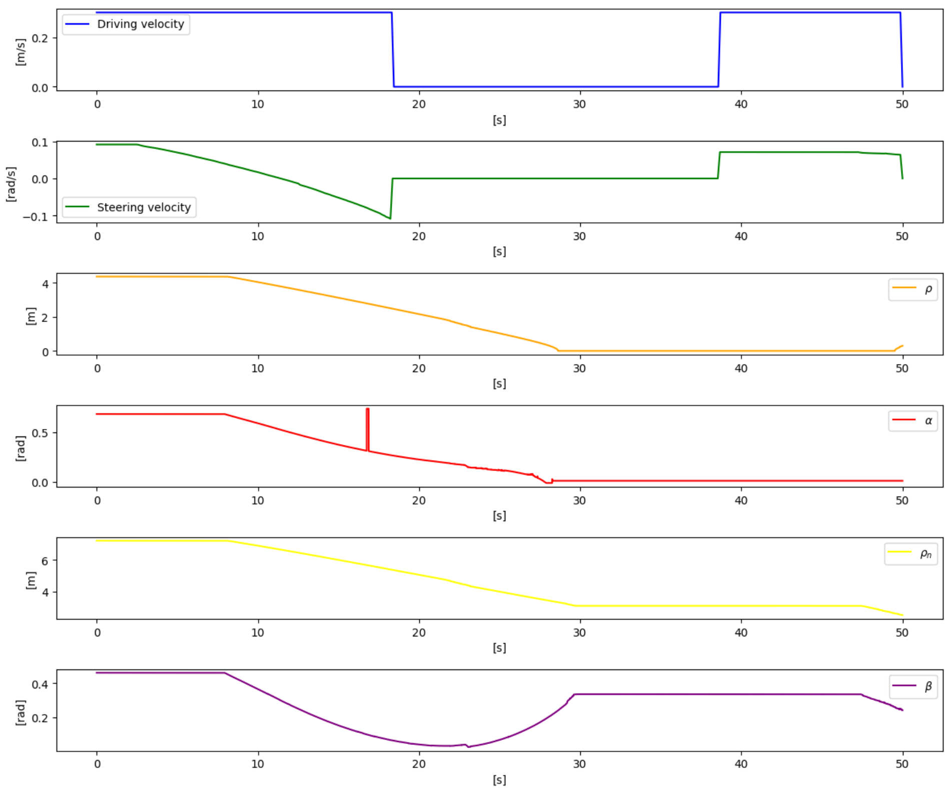

Figure 8: Base controller velocities and error metrics on-the-stop.

Regarding the planners, we report in Figures 9, 10, 11 the EE position error (i.e., the error between the desired position and its actual position) respectively of SBLK, RRT, and PRM. It can be appreciated that, for all three planners, this error decreases along with the number of iterations, meaning that the EE can reach the target ball above the post.

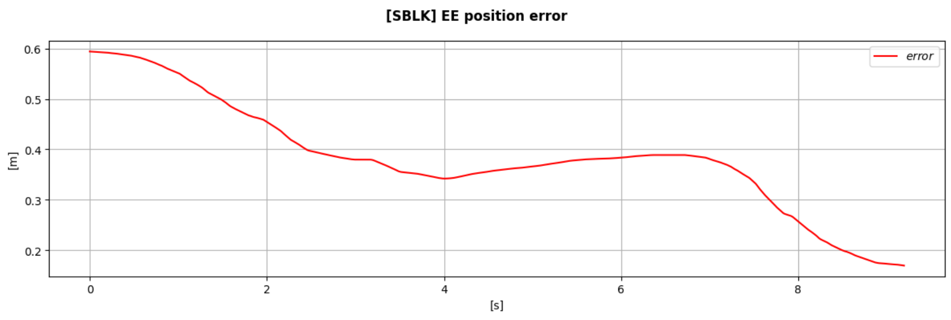

Figure 9: SBLK EE position error on-the-stop.

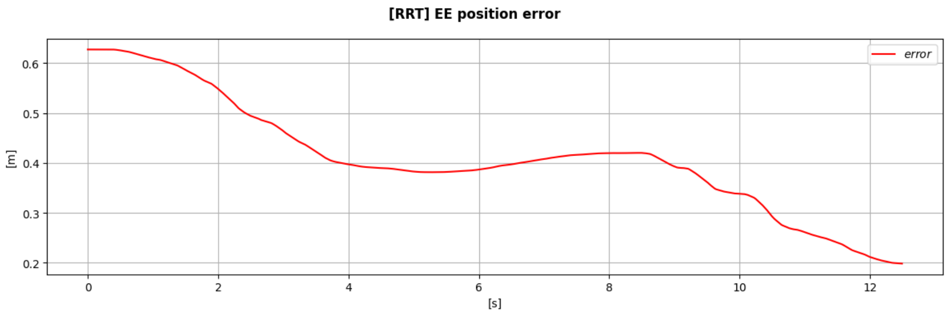

Figure 10: RRT EE position error on-the-stop.

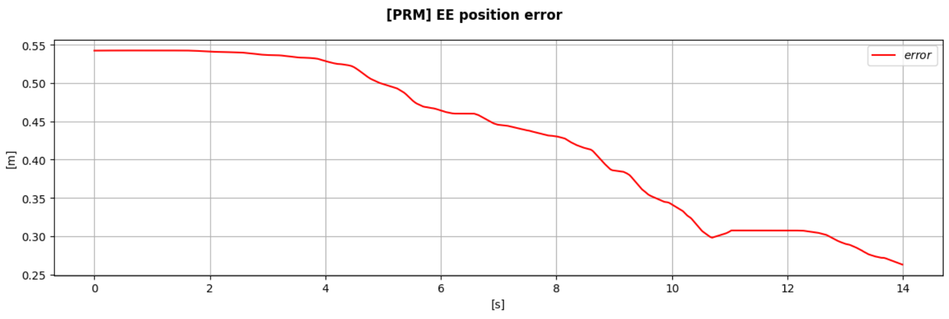

Figure 11: PRM EE position error on-the-stop.

5.1.2 Mobile Base with Quintic Plynomial

The robot reaches the ball in this simulation by following the quintic polynomial approach described in 4.2.2. Regarding the mobile base, the results are the same as reported in Figure 8, so we can make the same considerations. As for the EE position, the error is reported in Figure 12. Also, in this case, it can be noticed that this error decreases, meaning that the gripper can grasp the ball.

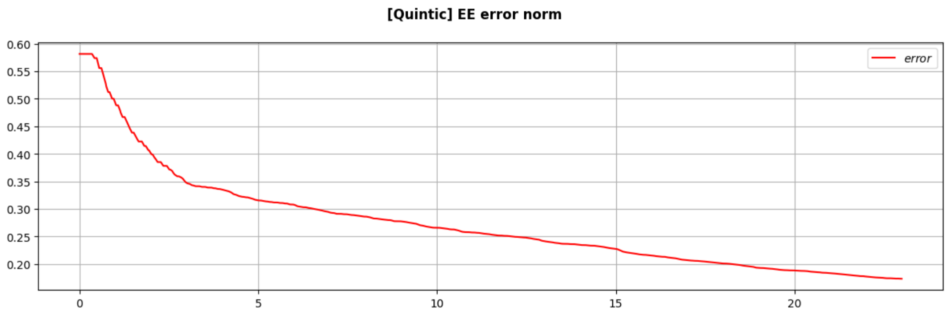

Figure 12: Quintic EE position error on-the-stop.

5.2 On-The-Move

This section analyzes the on-the-move simulations in which the robot reaches the closest approach points of the ball and the box without stopping.

5.2.1 Mobile Base with MoveIT Planners

We report in Figure 13 the velocities and the error metrics of the mobile base. The same considerations for what depicted in Figure 8 can be done, except for the fact that ρ_n and β are not constant when the robot is grasping the ball since it never stops but continues to go towards the box.

The EE position errors for the three planners are figured out in Figures 14, 15, 16.

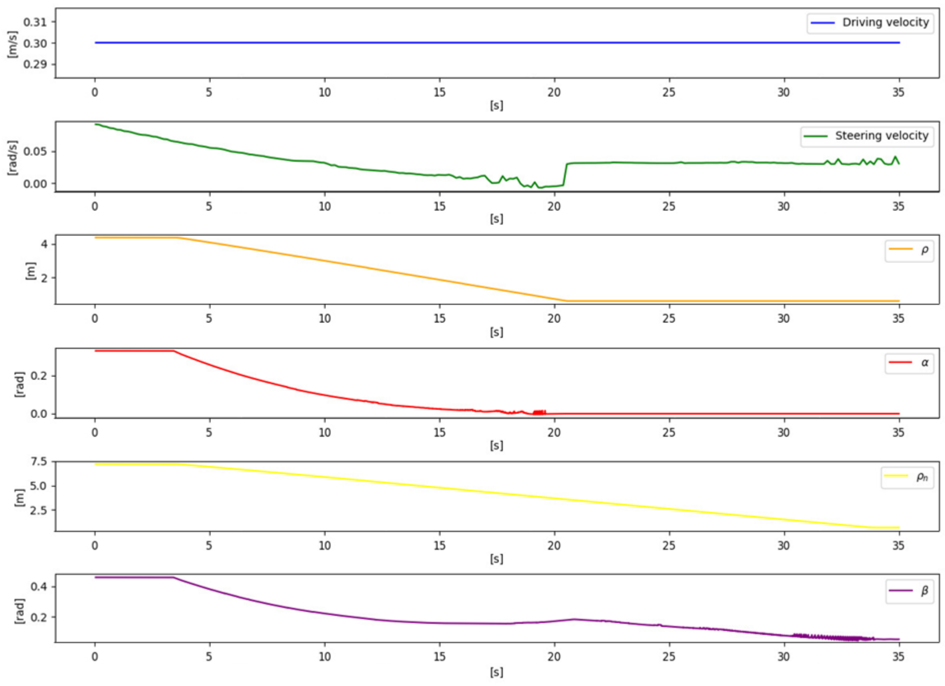

Figure 13: Base controller velocities and error metrics on-the-move.

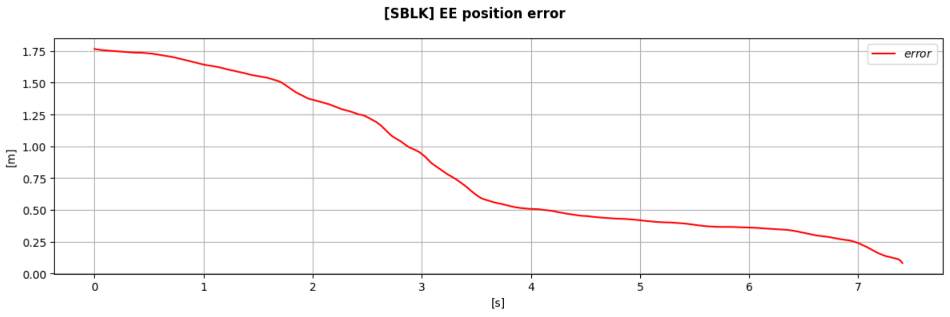

Figure 14: SBLK EE position error on-the-move.

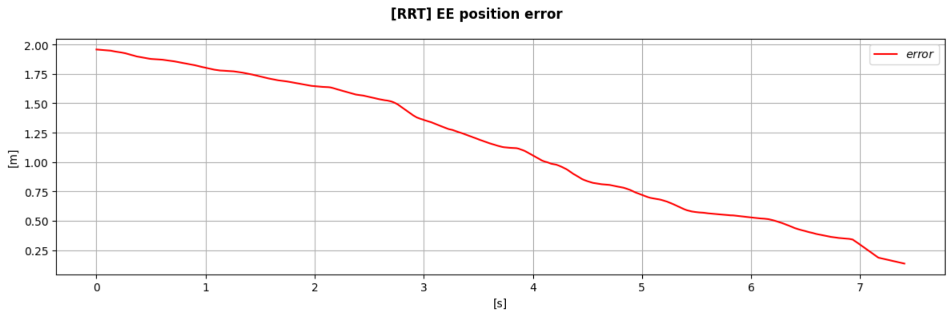

Figure 15: RRT EE position error on-the-move.

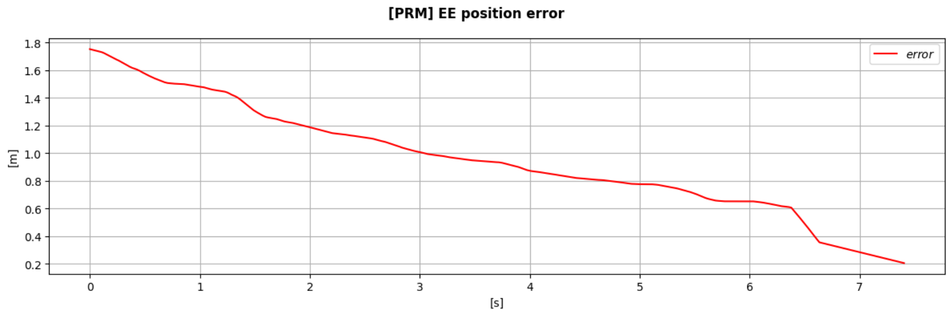

Figure 16: PRM EE position error on-the-move.

5.2.2 Mobile Base with Quintic Polynomial and QP Solver (QQP)

The robot reaches the ball in this simulation by following the quintic polynomial approach described in 4.2.2.

The results of the base are the same as reported in Figure 13, while the EE position error is in Figures 19, 20.

Focusing on the QQP (namely, the QP solver 4.3 combined with the quintic polynomial approach 4.2.2). Figure 17 reports the mobile base velocities and error metrics. We can appreciate that the driving velocity decreases when the robot is near the ball (as we can see better in Figure 18), increasing when the robot goes towards the box. This was expected by adopting the QP solver, which penalized the mobile base velocity to allow the robot to grasp and place the ball in a smoother and more secure way. After grasping, it was decided to revert to a constant but lower speed than the original one to avoid too big speed jerks

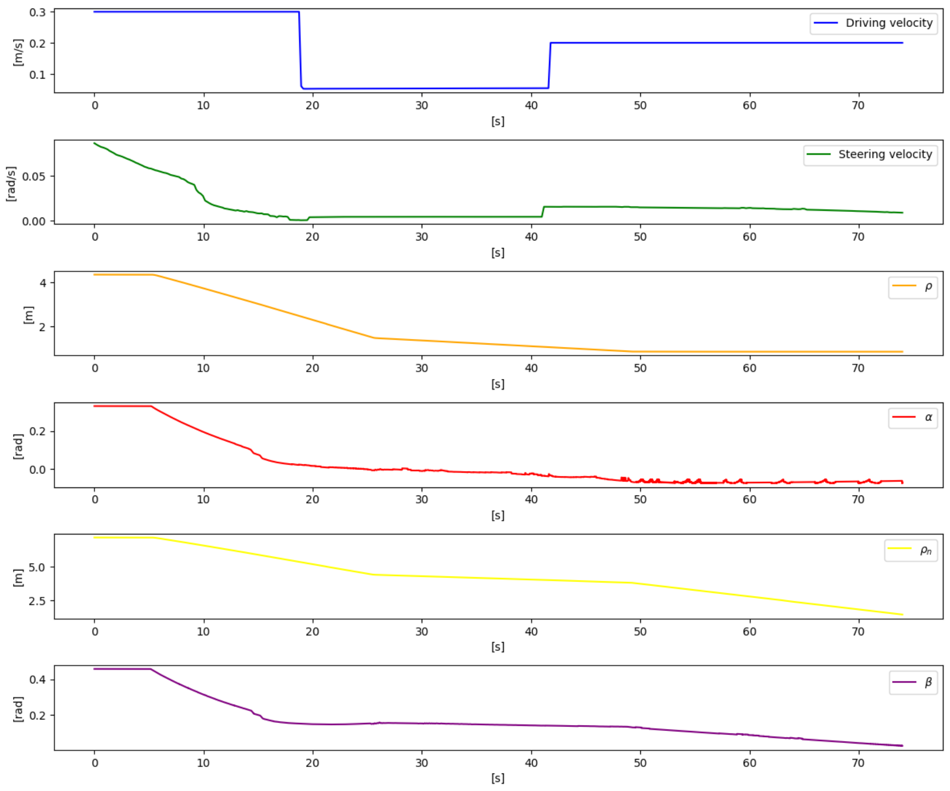

Figure 17: Base controller velocities and error metrics on-the-move by adopting QQP.

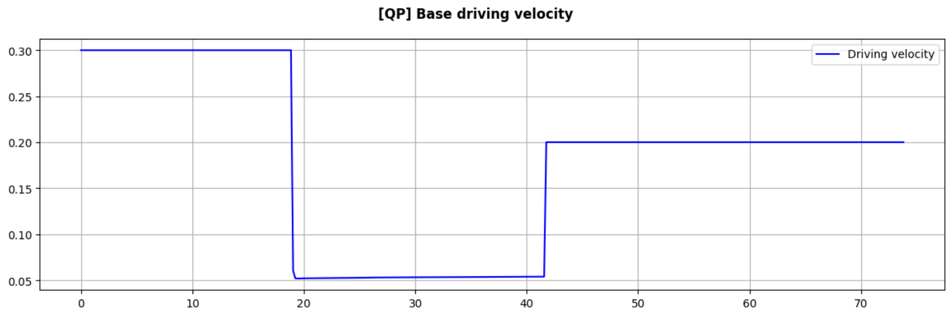

Figure 18: Base controller driving velocity on-the-move by adopting QQP.

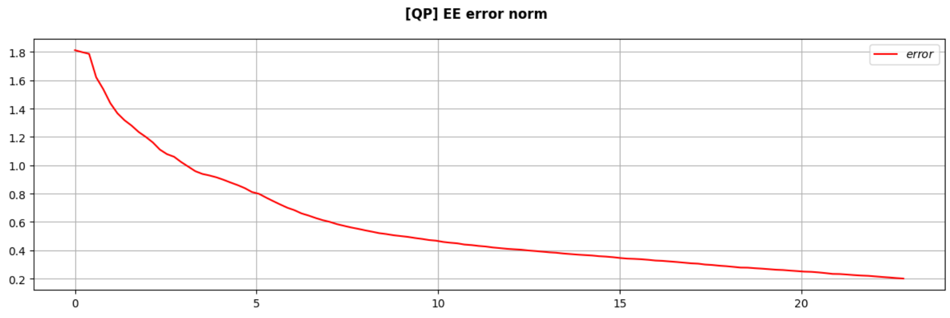

Figure 19: QQP EE position error on-the-move.

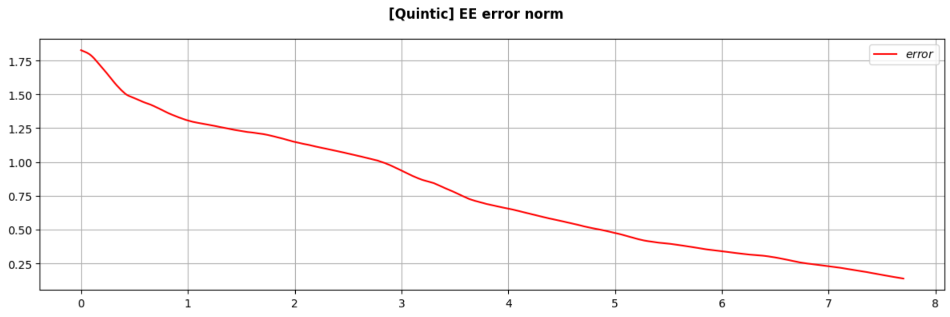

Figure 20: Quintic EE position error on-the-move.

5.3 Times and Success Rate

In Table 1, we report the planning time (T_{plan}), trajectory time (T_{traj}), simulation total time (T_{tot}) for each planner on-the-stop, and the same times with also the success rate (sr) for the on-the-move simulations. These measures refer to a mean value over the trials.

	On-The-Stop			On-The-Move			
Approach	$T_{plan}[s]$	$T_{traj}[s]$	$T_{tot}[s]$	$T_{plan}[s]$	$T_{traj}[s]$	$T_{tot}[s]$	sr
QQP	-	-	-	-	23.00	74.00	**10/10**
Quintic	-	23.02	60.00	-	7.67	37.80	9/10
SBLK	6.29	9.17	49.00	3.38	**3.58**	25.00	3/10
RRT	10.01	8.64	50.00	3.35	4.53	23.00	4/10
PRM	6.39	7.12	42.00	3.73	4.49	**22.00**	2/10

Table 1: Planning time (T_{plan}), trajectory time (T_{traj}), simulation total time (T_{tot}) by adopting each planner on-the-stop, and the same time with and the success rate (sr) for the on-the-move simulations.

It can be noticed that, when adopting the MoveIT planners, T_{tot} is smaller on-the-move concerning on-the-stop (+40%). This is because the robot doesn't stop to grasp the ball. Regarding the quintic polynomial approach, there is no T_{plan} since The trajectory is updated at every time step, allowing for adaptive control that reacts to environmental changes, object movements, new sensory information, and inaccuracies in robot operation. Moreover, the QQP approach doesn't make sense on-the-stop simulations since it aims to coordinate the base and arm motions during the grasping. However, the QQP approach is the best for sr. Indeed, regarding the sr, there are improvements of +200%, +125%, and +350%, respectively, concerning the SBLK, RRT, and PRM. All these improvements are at the expense of T_{tot}.

6 Conclusions

The project has successfully demonstrated the effectiveness of a motion-responsive manipulation architecture for mobile robots. By implementing and rigorously testing the manipulation system on the TIAGo robot, we have obtained significant results that highlight notable improvements in both the speed of task execution and the smoothness of movements compared to traditional methods. Our simulations have shown that this approach offers superior robustness against perception errors and environmental perturbations, confirming the system's reliability in dynamic and complex scenarios.

One of the key findings of this project is the efficacy of the QQP approach, which has proven to be the most successful method in terms of success rate. Even though this method results in a higher overall simulation time, it still outperforms the baseline method, which relies on a quintic approach requiring the robot to stop. The QQP approach ensures continuous motion of the base while the manipulator performs its tasks, leading to a more fluid and efficient operational model.

The reactive nature of our approach allows the robot to adapt to changes in the environment in real time, significantly enhancing the accuracy and coordination between the moving base and the manipulator's arm. This represents a substantial advancement in the field of mobile robotics, offering considerable potential for applications in industrial and domestic settings where interaction with dynamic environments is essential.

Future research directions could focus on integrating more advanced artificial intelligence algorithms for enhanced object recognition and autonomous navigation. Additionally, optimizing the performance of the redundancy resolution controller could lead to even more efficient management of the robot's redundant degrees of freedom, further improving the system's overall efficiency. These advancements could pave the way for more sophisticated and autonomous mobile manipulation systems operating in increasingly complex and unpredictable environments.

References

[1] Saleh Alarabi, Chaomin Luo, and Michael Santora. A prm approach to path planning with obstacle avoidance of an autonomous robot. In *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*, pages 76–80, 2022.

[2] Yuliy Baryshnikov and Robert Ghrist. Euler integration over definable functions. *Proceedings of the National Academy of Sciences*, 107(21):9525–9530, 2010.

[3] Ben Burgess-Limerick, Chris Lehnert, Jurgen Leitner, and Peter Corke. An architecture for reactive mobile manipulation on-the-move, 2022.

[4] Jesse Haviland, Niko Sunderhauf, and Peter Corke. A holistic approach to reactive mobile manipulation. *IEEE Robotics and Automation Letters*, 7(2):3122–3129, April 2022.

[5] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726 vol.3, 1997.

[6] Steven Lavalle and James Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: New directions*, 01 2000.

[7] Jordi Pagès, Luca Marchionni, and Francesco Ferro. Tiago: the modular robot that adapts to different research needs. In *International Conference on Intelligent Robots*, 2016.