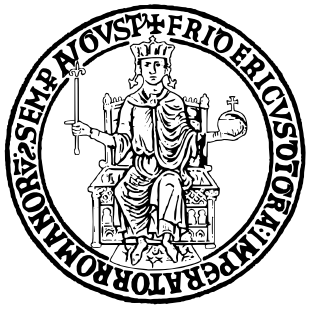
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Field & Service

Dynamic Control of a Ballbot with Integrated Manipulator for Object Handling in Cluttered Environments

Professor:
Fabio Ruggiero

Candidate:
Roberto Rocco P38000274
Chiara Panagrosso P38000272

GitHub Repositories:
https://github.com/Robertorocco/MIAPure-ballbot-ur5e-manipulation.git
https://github.com/chiarapanagrosso/MIAPure-ballbot-ur5e-manipulation.git

Contents

Introduction

Mobile manipulation represents a key frontier in robotics, merging the dexterity of robotic arms with the agility of dynamically balanced mobile platforms. Among such systems, the ballbot stands out for its ability to maintain balance on a single spherical wheel, providing omnidirectional motion and a compact footprint ideal for confined environments. However, the inherent instability of this configuration poses significant challenges for control design, especially when an additional manipulator alters the system's center of mass and introduces dynamic coupling effects.

This work addresses the development of a robust control architecture for a ballbot equipped with an upper-body manipulator, focusing on the stabilization, motion tracking, and disturbance rejection of the combined system. A comprehensive mathematical model of the ballbot dynamics is derived, and advanced control strategies—ranging from cascaded LQR–PI to adaptive model reference control—are implemented to ensure stable operation under varying load conditions. The system's performance is evaluated through simulation, emphasizing balance preservation during manipulation tasks.

The control methodology and the MiaPURE-based hardware architecture adopted in this work are inspired by the framework presented in [2]. In particular, the cascaded LQR–PI control structure and the concept of the compensation module were derived from that study, and their corresponding control diagrams and signal magnitudes were taken as references. Conversely, the generation of velocity references and the parameter tuning procedure were developed independently, as the original work was designed for experiments involving human interaction with the ballbot. Furthermore, the trajectory planning and manipulation aspects—such as the use of MoveIt Task Constructor in ROS 2 for motion generation and control—are further developed within a subsequent project [1].

In the final part of this work, an alternative control strategy for the complete mobile manipulator system is also explored. Instead of implementing a full-body control scheme—which would require the explicit coupling of the manipulator and base dynamics within a unified control law—the proposed approach attempts to approximate such coordination by generating corrective torques in the operational space. These torques are then superimposed onto the joint-space torques produced by the local controllers of the manipulator and the ballbot base. However, as will be shown in the subsequent analysis, this strategy does not fully replicate the performance achievable by a true full-body control framework. A full-body controller would inherently account for the coupled dynamics of the entire structure, enabling the coordinated exploitation of both the base and the arm to accomplish the manipulation task with higher stability and precision.

Chapter 1

Mobile Manipulator System

The MIA Pure ballbot is a dynamically stable mobile robot characterized by an intrinsically unstable structure, similar to that of an inverted pendulum. Its working principle relies on actively balancing an upper body on a single spherical wheel. This architecture provides the unique advantages of omnidirectional maneuverability and a minimal ground footprint, making the platform exceptionally agile in constrained environments. The system is composed of two fundamental parts: a spherical wheel, which allows the robot to move by rotating with a minimal contact point on the ground, and an upper body.

In the specific configuration discussed in this documentation, a UR5e robotic arm is mounted on this upper body, serving as the primary payload to perform manipulation tasks. This chapter will delve into the detailed hardware architecture and the mathematical model that describes the dynamics of this complex mobile manipulator.

1.1 BallBot Hardware Architecture

The ballbot's mechanical and electrical hardware is designed to create a compact and agile mobile base, engineered to support and dynamically balance a significant payload like a robotic manipulator. The architecture is composed of several key subsystems that work in concert to achieve stable, omnidirectional motion. The following sections describe each of these fundamental components in detail, referencing the hardware of the MiaPURE prototype.

Spherical Wheel The foundational component of the robot's locomotion is the spherical wheel (SW), which serves as the single point of contact with the ground. It is constructed from a lightweight bowling ball core, coated with a high-traction surface made from 6.35 mm thick Styrene-Butadiene Rubber (SBR) sheets. The final assembly has a diameter of 22.9 cm and a total weight of 3.6 kg.

Omni-wheels Torque is transmitted from the motors to the spherical wheel via a set of three 125 mm single-plate omni-wheels (OWs). These specialized wheels are positioned to support the entire weight of the upper body while efficiently transferring motive force. The small rollers along the circumference of each OW are coated with Thermoplastic Polyurethane (TPU) to ensure a high coefficient of friction, which is essential for slip-free torque transmission.

Chassis and Actuation System The chassis provides the structural integrity for the upper body and serves as the mounting platform for the powerful actuation system. Fabricated from aluminum sheet and extruded stock, it provides a rigid frame for all other components. It houses the three primary actuator units and includes a set of spring-loaded ball arrestors that apply a preload to the spherical wheel, ensuring it remains firmly in contact with the omni-wheels. Each actuator unit is a quasi-direct drive (QDD) assembly, consisting of a high-torque brushless DC (BLDC)

motor paired with a custom planetary gearbox. The mass and inertia of these actuators, along with all onboard electronics—including the main control unit, motor drivers, Inertial Measurement Unit (IMU), and power systems—are accounted for in the dynamic model as part of the total lumped mass and inertia of the upper body.

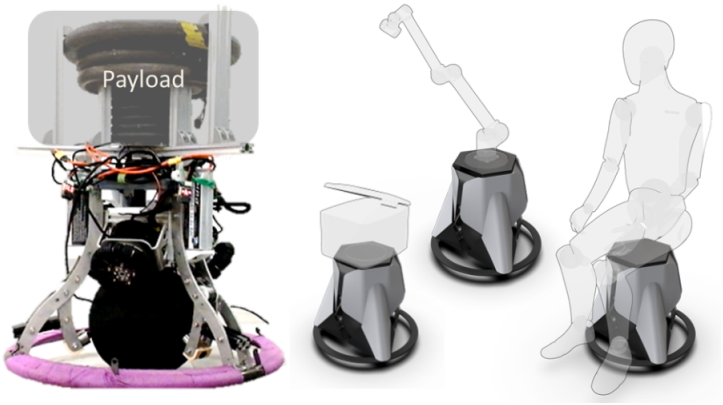


Figure 1.1: MiaPURE drivetrain hardware assembly.

1.2 BallBot Dynamical Model

To develop the mathematical model, several key assumptions are made. The system is treated as two rigid bodies: the main body of the robot and the spherical wheel it balances on. The complex omnidirectional motion is simplified by decomposing it into independent dynamics within two perpendicular vertical planes. A critical assumption is the no-slip condition, which posits that there is no slipping between the omni-wheels and the spherical wheel, nor between the spherical wheel and the ground. It is also assumed that the spherical wheel does not spin about its vertical axis relative to the ground. This requires that the motor torques remain within the limits of static friction to ensure proper torque transmission. Furthermore, it is assumed that the spherical wheel maintains continuous contact with the floor at all times. All other forms of friction, such as rolling or kinetic friction, are considered negligible.

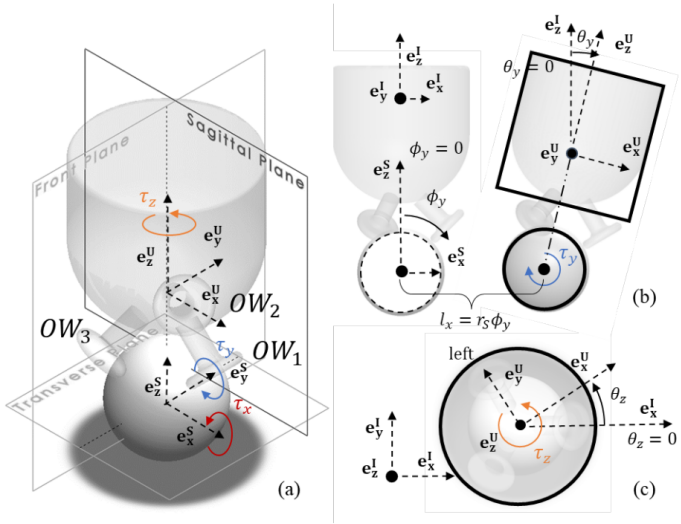


Figure 1.2: (a) Three individual planes defined for planar models and input torque applied to each model. (b) Translation model of the ballbot in the sagittal plane. (c) Spin model in the transverse plane, the black circle represents the upper body of the ballbot from the top view.

The model idealizes the robot's physical structure into two primary **rigid bodies** :

1. The **upper body**, which is a lumped mass that includes the chassis, omni-wheels, motors, and any payload. To account for the non-uniform mass distribution, this body is characterized by a specific Center of Mass (COM) height (l_U) and a moment of inertia matrix (I_U).
2. The **spherical wheel** (SW), which acts as the robot's mobile base.

To simplify the analysis of its complex 3D motion, the system's dynamics are decomposed into three independent 2D planar models: the **Sagittal**, **Frontal**, and **Transverse planes**: in each plane model, the complex interactions between the multiple omni-wheels and the spherical wheel are simplified to a single virtual torque (τ_j) applied at the center of the sphere. The mathematical relationship between this virtual torque and the torques of the actual motors is established later through a torque conversion matrix.

For the planar models, two generalized coordinates are used to describe the system's state in the sagittal ($j = y$) and frontal ($j = x$) planes:

- $\boldsymbol{\theta_j}$: The tilt angle of the upper body relative to the vertical axis.
- $\boldsymbol{\phi_j}$: The angular position of the spherical wheel. The translational displacement of the ballbot along an axis is directly derived from this angle (e.g., $l_x = r_s\phi_y$).

For the transverse plane, there's only one generalized coordinate, the yaw of the upper body described by the angle $\boldsymbol{\theta_z}$, due to the no-spin assumption.

The equations of motion (EOM) for the ballbot are derived using the **Euler-Lagrange energetic approach**. This method is particularly powerful as it operates on scalar energy quantities (kinetic and potential energy) rather than vector forces and accelerations, simplifying the derivation. The Lagrangian, $\mathcal{L}$, of the system is defined as the difference between its total kinetic energy, T, and total potential energy, V:

$$\mathcal{L} = T - V$$

Given the assumption of decoupled dynamics, the model for each plane can be derived independently.

1.2.1 Energy Formulation for the Sagittal Plane

The derivation for the sagittal (pitch) plane, with generalized coordinates $q_y = [\theta_y, \phi_y]^T$, is presented here.

Kinetic Energy (T_y) The total kinetic energy is the sum of the kinetic energies of the spherical wheel (T_{SW}) and the upper body (T_{UB}).

- **Spherical Wheel Kinetic Energy (T_{SW}):** This term includes both the translational energy from the movement of its center of mass and the rotational energy about its center.

$$T_{SW} = \frac{1}{2}m_S(r_S\dot{\phi}_y)^2 + \frac{1}{2}I_S\dot{\phi}_y^2$$

  where m_S, I_S, and r_S are the mass, moment of inertia, and radius of the wheel, respectively. The term $r_S\dot{\phi}_y$ represents the linear velocity of the wheel's center.

- **Upper Body Kinetic Energy (T_{UB}):** This term is more complex, as the motion of the upper body's COM is a combination of the translation of the spherical wheel and the rotation of the body itself, similar to a pendulum on a moving cart.

$$T_{UB} = \frac{1}{2}m_U(v_{Ux}^2 + v_{Uz}^2) + \frac{1}{2}I_{Uy}\dot{\theta}_y^2$$

The velocity components of the COM, located at a distance l_U from the wheel's center, are:

$$v_{Ux} = r_S\dot{\phi}_y + l_U\dot{\theta}_y\cos(\theta_y), \quad v_{Uz} = -l_U\dot{\theta}_y\sin(\theta_y)$$

Here, m_U is the upper body mass and I_{Uy} is its moment of inertia for pitching motion.

Potential Energy (V_y) The potential energy of the system is determined by the gravitational potential of the upper body, assuming the potential energy of the spherical wheel at ground level is constant and thus can be set to zero.

$$V_y = m_U g l_U \cos(\theta_y)$$

This equation defines the potential energy based on the vertical height of the upper body's COM, which changes with the tilt angle θ_y.

Generalized Forces (Q) The non-conservative forces acting on the system include the virtual torque τ_y applied to the spherical wheel and the dissipative effects due to viscous friction. Specifically, the generalized force associated with the upper body's tilt coordinate θ_y is the reaction torque $Q_{\theta_y} = -\tau_y$, while that corresponding to the wheel's rotation ϕ_y is given by $Q_{\phi_y} = \tau_y - b_\phi\dot{\phi}_y$.

1.2.2 Frontal Plane Dynamics

The derivation for the frontal (roll) plane is analogous to the sagittal plane. The same energy formulations are used, substituting the corresponding coordinates ($q_x = [\theta_x, \phi_x]^T$) and using the moment of inertia for roll, I_{Ux}.

1.2.3 Transverse Plane Dynamics

The dynamic model for the robot's motion in the transverse plane describes the spin, or yaw, of the upper body about its vertical axis. This is modeled as a simple, single degree-of-freedom (DOF) system. A critical assumption is made to ensure full control authority over the robot's orientation: the spherical wheel is considered non-spinning with respect to the ground. Consequently, any yaw motion, denoted as θ_z, results solely from the rotation of the upper body relative to the stationary wheel. Under this condition, applying Newton's Second Law for rotational motion yields the following equation of motion:

$$I_z\ddot{\theta}_z + D_z\left(\dot{\theta}_z\right) = \tau_z \tag{1.1}$$

where I_z represents the lumped moment of inertia of the upper body and omni-wheels about the vertical spin axis, $D_z(\dot{\theta}_z)$ denotes the viscous friction torque opposing the spin, and τ_z is the net input torque applied to the chassis to generate the yaw motion.

1.3 Manipulator Hardware Architecture

The manipulator used is the **Universal Robots UR5e**, a versatile collaborative robotic arm widely adopted in industrial and research settings. It features six revolute joints, providing it with six degrees of freedom and a high level of dexterity. The UR5e offers a payload capacity of up to 5 kg and a reach of approximately 850 mm, making it well-suited for pick-and-place tasks within defined workspaces. To enable grasping operations, the manipulator's standard configuration was extended by integrating a realistic gripper, the **Robotiq 2-Finger 85**. This gripper was chosen for its high mechanical compatibility, as it mounts directly to the robot's flange, and for its 5 kg load capacity, which matches that of the manipulator itself.

Chapter 2

System analysis

The following chapter introduces and analyzes the dynamic behavior of the complete robotic system, which consists of a ballbot platform equipped with a manipulator mounted on top. The analysis is divided into two main parts: the modeling of the mobile base dynamics and the study of the manipulator subsystem. Particular attention is devoted to the physical constraints imposed by the omniwheel actuation and to the interactions between the manipulator's motion and the base's balance. Subsequently, the nonlinear dynamic equations are linearized around an equilibrium configuration — corresponding to the upright and static posture of the system — in order to obtain a state-space representation suitable for control design.

2.1 Ballbot System

This section focuses on the analysis and modeling of the mobile base, which serves as the fundamental component responsible for maintaining the balance and mobility of the entire system. The dynamic equations of the ballbot are derived and analyzed in open-loop conditions to understand the intrinsic behavior of the platform prior to controller design. The derivation of the equations of motion is performed through a symbolic formulation based on the **Lagrangian method**, as introduced in the previous chapter. This approach considers both the kinetic and potential energy contributions of the moving bodies and includes the effects of **external forces** applied to the platform as well as **dissipative phenomena**, such as motor friction and rolling resistance. The resulting nonlinear system of equations is expressed in a general form suitable for numerical simulation, linearization, and subsequent control design. This symbolic modeling approach ensures high accuracy in the derivation process and offers flexibility for parameter variation, model scaling, and detailed analysis of the system's open-loop dynamics.

2.1.1 Linearization for Control Design

To analyze the local stability dynamics of the ballbot system in its open-loop configuration the nonlinear model $f(\mathbf{x}, \mathbf{u})$ is linearized around its upright equilibrium point. This equilibrium, $\mathbf{x}_0$, is defined as the state where the ballbot is perfectly vertical and stationary and all inputs are zero:

$$\mathbf{x}_0 = [\theta_x, \theta_y, \theta_z, \phi_x, \phi_y, \dot{\theta_x}, \dots]^T = \mathbf{0} \quad \mathbf{u_0} = [\underbrace{\tau_x, \tau_y, \tau_z,}_{\text{Control Torques}} \underbrace{F_{p,x}, F_{p,y}, F_{p,z}, \tau_{p,x}, \tau_{p,y}, \tau_{p,z}}_{\text{External Platform Wrench}}]^T = \mathbf{0}$$

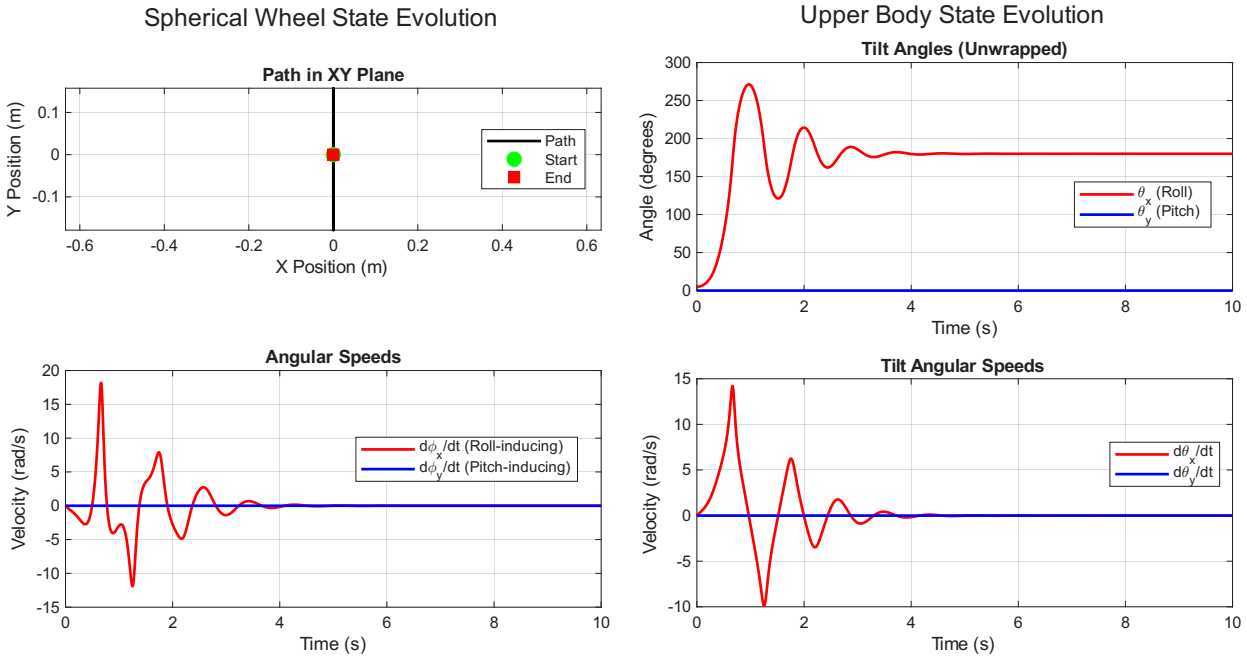
To the linear time-invariant (LTI) state-space model, the Jacobian matrices evaluated at the equilibrium point. The resulting state matrix, $\mathbf{A}$, is a 10×10 matrix, while the input matrix, $\mathbf{B}$, is a 10×9 matrix. A critical insight from the linearization is the (near) block-diagonal structure of these matrices. As we expected, This structure reveals that the system's dynamics are decoupled into the three independent planes of motion: **Saggital, Frontal and Transverse**. This decoupling implies that the full $\mathbf{A}$ matrix can be analyzed as three smaller, independent sub-matrices

($\mathbf{A}_{\text{sagittal}}$, $\mathbf{A}_{\text{frontal}}$, $\mathbf{A}_{\text{transverse}}$). Likewise, the $\mathbf{B}$ matrix shows that inputs (both external or control) intended for one plane only affect the states of that specific plane. This property is highly advantageous as it allows for the design of independent, simplified controllers for each plane of motion. Of course, this clean decoupling is the result of the simplifying hypotheses made during the modeling phase. The real-world physical system is inevitably more complex, featuring unmodeled dynamics and inherent cross-couplings. However, this simplified model allows for the development of straightforward control techniques.

Open-Loop Stability and Simulation

The local stability of the system around the upright equilibrium point determined by the eigenvalues of the $\mathbf{A}$ matrix. An analysis of these eigenvalues reveals the presence of two real positive eigenvalues, belonging to the sagittal and frontal dynamics plus one non-hyperbolic eigenvalue for the transverse plane. This mathematically confirms that the upright equilibrium point is globally unstable. The system was initialized at its upright equilibrium point $\mathbf{x}_0$ with a small initial perturbation in tilt angle (e.g., $\theta_{x,0} = 0.5°$) and zero control input and external perturbation ($\mathbf{u}(t) = 0$).

-



(a) Open-loop evolution of spherical wheel states

(b) Open-loop evolution of upper body states

Figure 2.1: Open-loop simulation results following a small initial roll perturbation. The system is clearly unstable.

As clearly demonstrated by the simulation (Figure 2.1), the system is unable to maintain its balance. The tilt angles (Figure 2.1b) initially grow exponentially and then start to oscillate around a different value, causing the ballbot to tip over to a stable equilibrium. This behavior is highly characteristic of a classic inverted pendulum, to which the ballbot's dynamics are analogous. The simulation also highlights the decoupled nature of the dynamic model we derived, as the initial roll perturbation only excites the roll-inducing states while the pitch-inducing states remain at zero (Figure 2.1a).

2.1.2 Omniwheel Motor Torque Transformation Matrix

The ballbot achieves its signature omnidirectional maneuverability through a unique actuation system consisting of three omniwheels arranged in a specific configuration around the spherical wheel: they are spaced symmetrically around the vertical axis, 120° apart from each other and each omniwheel is tilted with its plane making a 45° angle with the vertical axis. An omniwheel has passive rollers mounted along its circumference. This design allows it to transmit active torque around its main axis of rotation while allowing passive, low-friction motion along its axial direction. This design leads to the problem of **underactuation**: the complete ballbot system has 5 DOF, but its controlled only by three motors, however this strategical arrangement of the three omniwheels gives us the possibility to impart a net torque on the system in any 3D direction, allowing the system to be fully controllable in the plane, i.e. the ballbot can move instantaneously in any planar direction. This can be ultimately proven by formalizing the torque transformation matrix

$$\begin{bmatrix} \tau_x \\ \tau_y \\ -\tau_z \end{bmatrix} = T_{3D} \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$$

that relates the motor torques (τ_1, τ_2, τ_3) to the resulting torques in the decoupled planar models (τ_x for frontal, τ_y for sagittal, and τ_z for transverse). Here, $-\tau_z$ is used to represent the reaction torque applied to the spherical wheel when the motors generate a yaw torque τ_z on the upper body.

The derivation can be summarized in four steps:

- Motor Torque → Force: Each motor's torque (τ_i) is converted into a traction force ($\mathbf{F}_{t,i}$) at the omniwheel's contact point.
- Force → Sphere Torque: This traction force is applied at a distance from the sphere's center, creating a torque on the sphere ($\tau_{s,i}$) via a cross product.
- Summation: The total torque on the sphere ($\tau_S = [\tau_x, \tau_y, -\tau_z]^T$) is the vector sum of the torques from all three wheels.
- Matrix Form: This linear sum is expressed as a matrix multiplication, where T_{3D} contains the geometric terms from the previous steps.

The resulting s T_{3D} matrix is configuration independent and invertible leading to fully controllability in the plane (position and orientation): this ensures that a unique combination of motor torques exists for any desired net torque vector. The inverse of this matrix, T_{3D}^{-1}, is used in the control system to calculate the required individual motor torques for a desired motion.

2.2 Manipulator System

This section details the properties of the UR5e robotic arm that is mounted on the ballbot's chassis. Understanding the manipulator's influence is critical, as its mass, inertia, and configuration directly impact the stability and control of the entire mobile platform. To ease the ballbot system to balance upright, it is important to define a "home position" for the manipulator to mantain during the motion of the base. The "home position" is therefore defined by two primary objectives: to obtain vertical alignment of the manipulator's combined center of mass (CoM) as closely as possible with the vertical axis of the ballbot and also to have the lowest possible aligned center of mass. This reduces the magnitude of the destabilizing effect due to the presence of the arm, requiring less torque to maintain balance.

The precise location of the arm CoM in this home pose was computed using a custom MATLAB script. This script imports the mass, inertia, and link geometry from the robot's URDF file and calculates the aggregated CoM and the corresponding Inertia Tensor for this specific joint

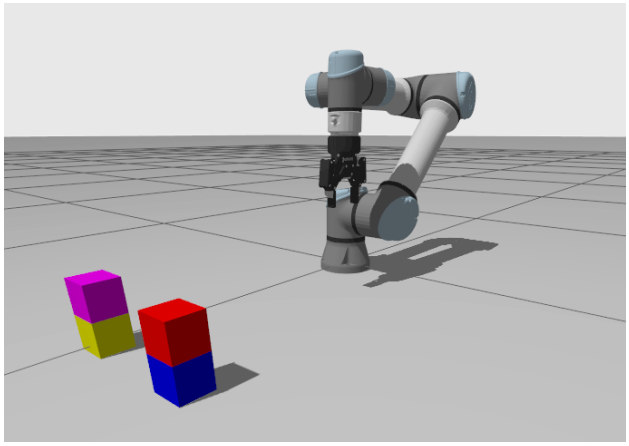


Figure 2.2: Manipulator Home Position

configuration. It is important to identify these parameters to have a more accurate description of the whole system that has to be controlled. Based on this analysis, the calculated Center of Mass of the manipulator, expressed in the frame relative to the spherical wheel, is:

$$CoM_{arm} = \begin{bmatrix} -0.0325 & 0.0672 & 0.7649 \end{bmatrix} m$$

2.2.1 From URDF to SIMSCAPE model

To create a high-fidelity simulation of the complete system, the manipulator's design is imported from its Unified Robot Description Format (URDF) file into the Simscape Multibody environment. This import is accomplished using the MATLAB `smimport` function, which parses the URDF file and automatically generates a Simscape Multibody block diagram. This diagram represents the manipulator with individual blocks for each body (link) and joint.

Once the manipulator model is imported and validated in isolation, some relevant modification where made:

1. **Gripper Joint Correction**: "Mimic" joints present in the gripper URDF file were manually build in the Simscape model. They were originally imported as normal, independent joints; however, their kinematic motion is related to a "master" joint (the only gripper joint actively controlled).

2. **Ballbot Base Integration**:The manipulator base was linked to the platform of the ballbot. Due to the non-existence of a Simscape model for the ballbot itself, the platform was modeled as a Cartesian + Spherical (called Gimbal in Simscape) joint. Their motion is commanded by external inputs provided by the ballbot model's evolution: in particular the Cartesian Joint X-Y positions are governed by the positon of the spherical wheel in the plane, while the Spherical Joint RPY rotations are governed by the tilt angles of the chassis. This allows the manipulator's base position and rotation to reflect the behavior of the mobile base.

Chapter 3

Trajectory generation

3.1 BallBot Trajectory

To enable autonomous navigation, a reference trajectory must be generated for the BallBot to follow. Specifically, the trajectory is defined for the contact point of the spherical wheel with the ground. This path must guide the robot from a designated start position to a goal position within a known 2D environment containing static obstacles. The primary requirement for this trajectory is to be completely **collision-free**, ensuring the robot can safely traverse the workspace. To achieve this, a **path planning algorithm** is employed to compute an optimal and safe route, which is then provided as a series of waypoints to the robot's control system.

3.1.1 Geometric Path Planning

The geometric path planning problem is addressed using the RRT* (Rapidly-exploring Random Tree Star) algorithm, a state-of-the-art sampling-based planner known for its efficiency in complex spaces and its ability to find high-quality, near-optimal paths.

Formally, the path planning problem is defined in the robot's **configuration space** (C-space), which is the set of all possible configurations (positions and orientations) the robot can assume. For our 2D navigation task, the C-space is simplified to the set of all (x, y) positions on the plane. This space is partitioned into two disjoint subsets: the free space (C_{free}), where the robot can move without colliding, and the obstacle space (C_{obs}). The objective of the planner is to find a continuous path, τ, that connects a start configuration q_{start} to a goal configuration q_{goal} such that the entire path lies within C_{free}. To implement this, the continuous environment is first discretized into a `binaryOccupancyMap`. This data structure represents the workspace as a grid, where each cell is assigned a binary value: '0' for free space and '1' for occupied space. This representation provides an efficient computational model for collision checking, which is a fundamental operation for any path planning algorithm.

Given the map, RRT* constructs a tree of collision-free paths, rooted at the starting position, and incrementally expands it throughout the free space. At each iteration, a random point is sampled from C_{free}. The algorithm then finds the nearest node in the existing tree and steers a new branch towards the sampled point. The key feature that distinguishes RRT* is its **optimality**. While the standard RRT algorithm finds a feasible path quickly, it does not guarantee its quality. RRT*, on the other hand, incorporates two crucial steps to converge towards the optimal (shortest) path:

1. *Best Parent Selection*: When a new node is considered for addition to the tree, RRT* examines a neighborhood of existing nodes around it. It then selects as its "parent" the node that results in the lowest-cost path from the root of the tree (q_{start}), rather than simply connecting to the geometrically nearest node.

2. *Tree Rewiring*: After a new node is added, the algorithm re-examines the nodes in its neighborhood. If connecting any of these neighboring nodes through the newly added node results in a shorter path from the start, the algorithm "rewires" the tree by changing their parent-child connections.

These two processes ensure that the quality of the path continuously improves as the number of iterations increases, resulting in a trajectory that is not only collision-free but also smooth and efficient for the BallBot to follow.

Key Search Parameters

The performance and behavior of the RRT* planner are tuned through several key parameters:

- **Obstacle Inflation**: This is a critical parameter for ensuring the physical safety of the robot. Since the planner computes a path for a single point (the robot's center), it does not inherently account for the robot's physical volume. Inflation addresses this by artificially expanding the boundaries of all obstacles in the `binaryOccupancyMap`. The inflation radius is set to be the radius of the robot's spherical base plus a user-defined safety margin ($R_{\text{inflation}} = r_{\text{robot}} + \delta_{\text{safety}}$).
- `MaxConnectionDistance` **and** `MaxIterations`: These parameters collectively control the exploration strategy and the quality of the final path. The `MaxConnectionDistance` defines the maximum length of a single branch that can be added to the tree in one iteration. A smaller value forces the algorithm to generate a denser, more detailed tree, which is effective for navigating through narrow passages. The `MaxIterations` set an upper limit on the number of samples, acting as a termination condition.

The final output of the planning process is a sequence of **waypoints**. The number of these points is determined by the algorithm's exploration parameters and the overall path length. These discrete waypoints define the computed route and serve as the fundamental reference for the control system, which is tasked with generating the continuous **velocity trajectory** needed for the robot to navigate smoothly and accurately along the path.

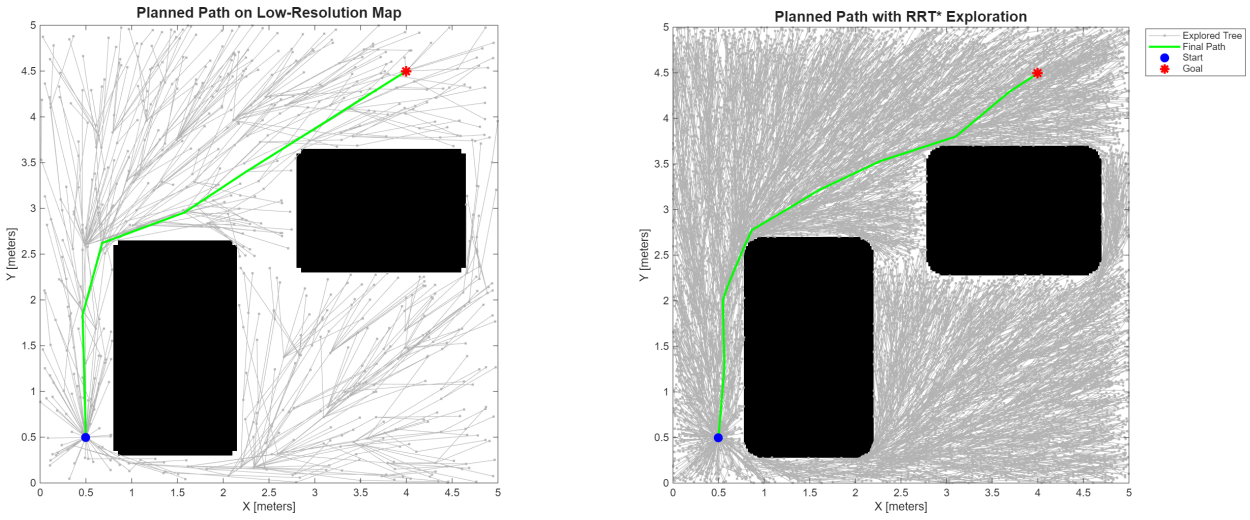


Figure 3.1: Comparison of RRT* performance on low- and high-resolution maps, showing the effect of map discretization on exploration density, obstacle accuracy, and overall path optimality.

Velocity Trajectory Generation

Since the geometric path produced by the planner provides only spatial waypoints without any timing information, a **velocity trajectory generation** stage is required to assign temporal evolution to the motion. The goal is to produce, for each path segment, a reference velocity profile with trapezoidal or triangular shape, defined by the limits of maximum velocity (v_{max}) and maximum acceleration (a_{max}).

The algorithm proceeds through the following steps:

1) **Segment extraction.** Starting from the ordered list of waypoints, the algorithm constructs a list of trajectory segments. Each segment is represented as a structure containing the start and end positions, direction vector, and total length.

2) **Computation of transition velocities.** Considering two adjacent segments, the direction change between them is used to determine an optimal transition velocity. This value depends on the turning angle and on v_{max}, ensuring smooth transitions between consecutive trajectory portions.

3) **Assignment of boundary velocities.** Once the transition velocities are known, they are projected onto each segment direction to compute the initial and final velocities (v_{start} and v_{end}) for that segment.

4) **Generation of trapezoidal/triangular profiles.** Using standard kinematic relations and the limit a_{max}, a velocity profile is generated for each segment. Depending on the segment length—variable because the path is generated by RRT*—the profile assumes a trapezoidal or triangular form. This step produces a reference velocity at each time instant along the segment.

5) **Feedback correction term.** A proportional feedback component is integrated into the velocity generation process to enhance trajectory tracking accuracy. Its purpose is to locally correct the reference velocity whenever a deviation from the planned path occurs, due to modeling inaccuracies or dynamic effects. Specifically, the commanded velocity is adjusted by adding a term proportional to the instantaneous positional error with respect to the current waypoint, effectively guiding the ballbot back toward the nominal path while maintaining smooth motion continuity.

6) **Boundary segment handling.** A stabilizing commanded velocity is generated at the beginning of the trajectory by applying a proportional gain K_p to the position error for the first 3 seconds of motion. For the penultimate segment, both v_{max} and a_{max} are halved to smooth the deceleration phase. Finally, for the last segment, the trapezoidal generation is replaced with a proportional control law on the position error, ensuring accurate convergence to the final waypoint.

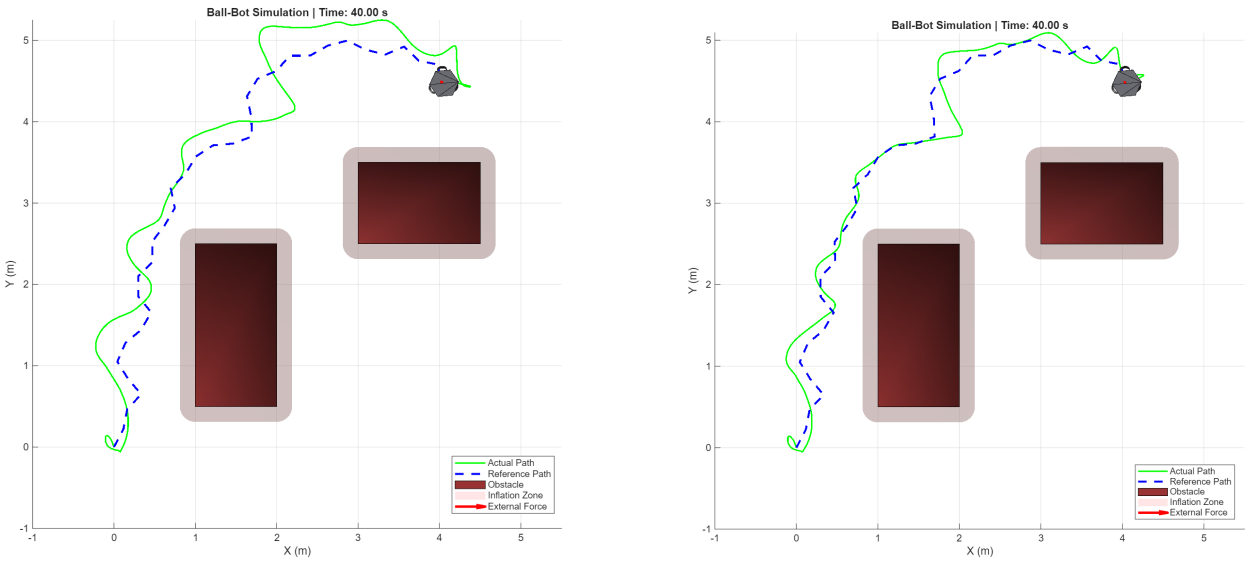
A further improvement in the trajectory tracking performance can be obtained by introducing an additional parameter, the *lookahead distance*. This parameter modifies only the feedback contribution in the velocity generation process, allowing the system to anticipate the motion by following a point that lies ahead along the trajectory instead of the current waypoint. Geometrically, it can be interpreted as if a circle of radius equal to the lookahead distance were centered on the moving ballbot, and the intersection between this circle and the planned path defines the instantaneous target to follow. To quantitatively assess the effect of the lookahead distance parameter, three simulations were conducted using different values for a given controller. The performance was evaluated through two sets of metrics: **tracking performance metrics** and **tilt angle metrics**. The tracking metrics include the Root Mean Square Error (RMSE) and the Maximum Absolute Error (MAE), which respectively measure the average and the peak deviation between the commanded and actual positions during the navigation phase. The tilt angle metrics, instead, provide

an estimate of the robot's body inclination in both the sagittal and transversal planes, reporting the mean and maximum absolute tilt angles during motion. These quantities indirectly describe the effort required by the controller to maintain balance while following the trajectory.

The comparison reveals that the configurations without lookahead and with a lookahead distance of 0.3 m yield very similar performance in both position tracking and body tilt. However, when the parameter is set to 0, the feedback contribution no longer depends on a forward point along the trajectory but instead acts directly on the orthogonal projection of the current position onto the path. This effectively makes the feedback correction more reactive to lateral deviations, leading to a drastic improvement in all performance metrics at the cost of a slightly longer trajectory execution time. The following table summarizes the obtained results.

Case	RMSE	MAE	Mean Roll	Max Roll	Mean Pitch	Max Pitch
No LH	0.22	0.53	2.78	6.58	2.66	7.73
LH = 0.3	0.21	0.53	2.99	7.01	2.51	6.93
LH = 0.0	**0.1360**	**0.3445**	2.86	6.18	**2.33**	**6.92**

Table 3.1: Comparison of trajectory tracking and tilt performance for different lookahead distance configurations. All position-related metrics (RMSE and MAE) are expressed in meters, while tilt angles are expressed in degrees.



(a) No lookahead

(b) Lookahead = 0.0

Figure 3.2: Comparison between two feedback correction strategies.

3.2 Manipulator Trajectory

Regarding the trajectory generation for the **Universal Robots UR5e** manipulator, the system relies on a complex software architecture that abstracts high-level symbolic commands into precise and safe physical movements. The entire execution process is managed by the **MoveIt Task Constructor (MTC)** framework, which decomposes complex robotic tasks—such as pick-and-place operations—into a hierarchy of interdependent sub-tasks called "stages". This stage-based structure defines the complete motion logic, including approaching, grasping, lifting, transferring, placing, and retreating phases.

The sequence of actions (which block to pick and where to place it) is not pre-programmed but generated through a two-level planning process. First, a **Hierarchical Task Network (HTN)** symbolic planner analyzes the initial world state, derived from Gazebo, and computes a high-level

plan to reach the desired goal. This plan, expressed as a list of primitive actions (e.g., `('unstack', 'block_a', 'block_c')`), is published on a ROS 2 topic and received by a dedicated MTC node. The node maps symbolic actions into executable MTC tasks, pairing each "pick" operation (such as `unstack` or `pickup`) with a corresponding "place" action (like `stack` or `putdown`). Within these tasks, motion planning is carried out inside specific MTC stages—such as the `Connect` stages—using the standard MoveIt pipeline based on *sampling-based* algorithms like **RRT** to compute **collision-free trajectories** in the joint space. The framework maintains an updated **Planning Scene** with the real poses of the blocks and employs an adaptive **Allowed Collision Matrix (ACM)** to dynamically define which entities can interact during each stage.

After execution, the trajectories generated in ROS 2 are exported via **ROSBAGs** and post-processed in MATLAB. This step corrects simulation artifacts due to Gazebo's low **Real-Time Factor (RTF)** during contact-rich phases, which causes irregular and non-smooth motion. Each joint trajectory is filtered and resampled through spline interpolation to recover a continuous profile. Moreover, since the planning process operates **online**—computing each step based on the current scene state—temporary "stall phases" appear while waiting for new plans. These intervals are detected and removed, resulting in a smooth and continuous trajectory suitable for controller validation.

The final output consists of seven reference trajectories: six for the manipulator joints and one for the Robotiq gripper, whose internal joints mimic the motion of the primary actuated finger. Additional details on the symbolic planning structure and ROS 2 integration can be found in [1].

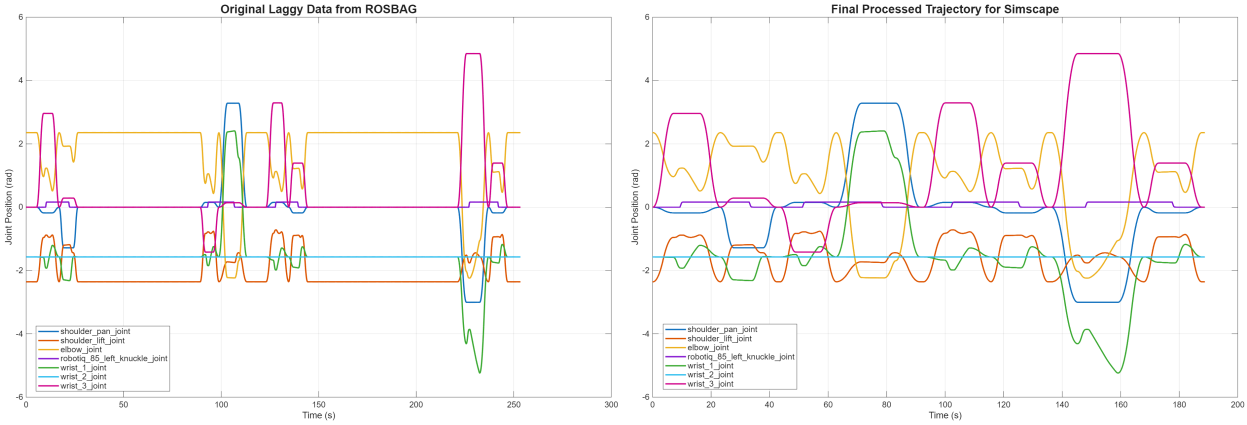


Figure 3.3: Comparison between raw and processed joints' trajectories.

A final remark concerns the execution mode of this trajectory. The execution speed is not fixed; a temporal scaling factor can be applied to the entire sequence. This factor is tuned according to the performance of the control system, particularly its ability to stabilize the BallBot and to react to the disturbances induced by the manipulator's motion. Consequently, the time profiles shown in the processed figure 3.3 should be regarded as indicative of the motion shape rather than its actual duration.

Moreover, the trajectory progression itself is conditional. As discussed in the control section, the "flow of time" for the manipulation advances only if the BallBot remains sufficiently balanced. Otherwise, the execution of the trajectory is paused until stability is restored, effectively decoupling the manipulation time from the simulation time.

Chapter 4

Controller Synthesis

The control synthesis presented in this chapter addresses the stabilization and motion control of the BallBot as a single rigid body. The dynamic coupling and interactions arising from the manipulator will be addressed in the subsequent chapter; in this chapter's formulation, any forces or torques exerted by the manipulator on the main body are treated merely as external disturbances to the system. The fundamental control challenge is analogous to that of an inverted pendulum, where the primary objective is to dynamically stabilize the inherently unstable system around its upright equilibrium point.

During the synthesis of these controllers, the effects of tunable parameters on the BallBot's performance were investigated. However, a detailed formal analysis of these effects is beyond the scope of this chapter. Such an analysis will be presented later when the dynamic coupling with the manipulator is explicitly taken into account.

4.1 Cascade LQR-PI

To address the control problem, we implement a **model-based** control methodology. Inspired by the approach presented by Xiao, which seeks to combine the benefits of stiction-compensation found in cascaded controllers with the optimality and intuitive tuning of model-based methods, we adopt a similar cascaded architecture. This structure is particularly suitable for high-payload systems where unmodeled dynamics, such as friction and stiction from elastomeric components, can degrade the performance of purely model-based controllers.

For the remainder of this analysis, it is assumed that any required velocity reference is generated by an outer-loop controller. Specifically, for point-to-point regulation tasks, a simple Proportional (P) controller is employed. This outer loop calculates a position error between the robot's current location and a desired target, generating a corresponding velocity command that acts like a virtual spring, attracting the BallBot's base towards the target position.

4.1.1 LQR Controller

The core of the inner-loop controller relies on the Linear Quadratic Regulator (LQR) to achieve dynamic stabilization. LQR is an optimal control technique that computes a state-feedback gain matrix, K, which minimizes a quadratic cost function trading off between state deviation and control effort. Its primary role here is to generate the necessary feedback torque to stabilize the system around its unstable upright equilibrium, where the BallBot is perfectly vertical ($\theta = 0$).

The design process begins by linearizing the nonlinear system dynamics around this equilibrium point, where all states and control inputs are zero. Based on the common assumption that the robot's dynamics can be effectively decoupled, the control design is performed independently for the sagittal (pitch) and frontal (roll) planes. For a single plane, the state is defined as $\mathbf{x} = [\theta, \phi, \dot{\theta}, \dot{\phi}]^T$, and the LQR gain K is computed by solving the algebraic Riccati equation for a given linearized

system $(\mathbf{A}, \mathbf{B})$ and user-defined weighting matrices, $\mathbf{Q}$ and $\mathbf{R}$. The matrix $\mathbf{Q}$ penalizes state errors, and its diagonal elements are tuned to prioritize stabilization; a significant weight is placed on the tilt angle θ to ensure the robot remains upright. The scalar $\mathbf{R}$ penalizes the control input, i.e., the motor torque τ. The resulting control law takes the form:

$$\tau = -K(\mathbf{x} - \mathbf{x}_{ref}) \tag{4.1}$$

where $\mathbf{x}$ is the current measured state and $\mathbf{x}_{ref}$ is the reference state vector. For velocity tracking, this reference is defined as $\mathbf{x}_{ref} = [0, \phi, 0, \dot{\phi}_{des}]^T$. This formulation commands the controller to drive the tilt angle θ and tilt rate $\dot{\theta}$ to zero, while ensuring the sphere's angular velocity $\dot{\phi}$ tracks the desired velocity $\dot{\phi}_{des}$ provided by the outer position loop.

4.1.2 LQR-PI Controller

While the LQR controller provides optimal stabilization based on the linearized model, its effectiveness relies heavily on the accuracy of that model. Real-world systems inevitably possess unmodeled dynamics, such as friction, stiction, and parameter variations, which can degrade the performance achieved by LQR alone. To enhance robustness against these discrepancies, a cascaded control architecture combining LQR with a Proportional-Integral (PI) controller is employed.

The cascaded structure operates by having the LQR feed a reference model, whose output velocity is then used as the target for the inner PI loop. The process unfolds as follows:

1. **LQR Reference Torque (τ_r):** The LQR component calculates a reference torque, τ_r, using the optimal gain K and the error between the current measured state $\mathbf{x}$ and a commanded reference state $\mathbf{x}_{ref} = [0, \phi, 0, \dot{\phi}_{des}]^T$. This reference state includes the desired velocity $\dot{\phi}_{des}$ generated by the outer position controller. The resulting τ_r represents the optimal torque according to the nominal linearized model.

2. **Reference Model:** This reference torque τ_r is fed into a reference model, which is essentially the linearized dynamics equation ($\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\tau$) derived from the nominal system parameters. Specifically, the model calculates the reference sphere angular acceleration that should result from τ_r: $\ddot{\phi}_r = \mathbf{A}(4,:)\mathbf{x} + \mathbf{B}(4)\tau_r$, where the index 4 corresponds to the $\dot{\phi}$ state.

3. **Inner PI Tracking Loop (τ_e):** The calculated reference acceleration $\ddot{\phi}_r$ is integrated over time to yield the reference sphere angular velocity $\dot{\phi}_r$. A Proportional-Integral (PI) controller forms the inner loop: it calculates a tracking error between the reference velocity $\dot{\phi}_r$ (now a state variable) and the actual measured sphere velocity $\dot{\phi}$. Based on this error and its integral, the PI controller generates a corrective tracking torque $\tau_e = K_p(\dot{\phi}_r - \dot{\phi}) + K_i \int (\dot{\phi}_r - \dot{\phi})dt$. This loop actively works to eliminate the speed tracking error, thereby compensating for the effects of model uncertainty and disturbances.

The final torque applied to the actual (potentially uncertain) system is the sum of the LQR's reference torque and the PI's tracking torque: $\tau = \tau_r + \tau_e$.

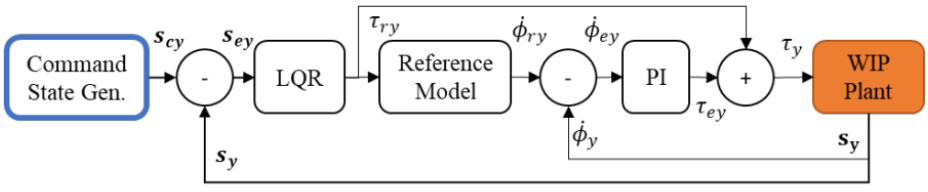
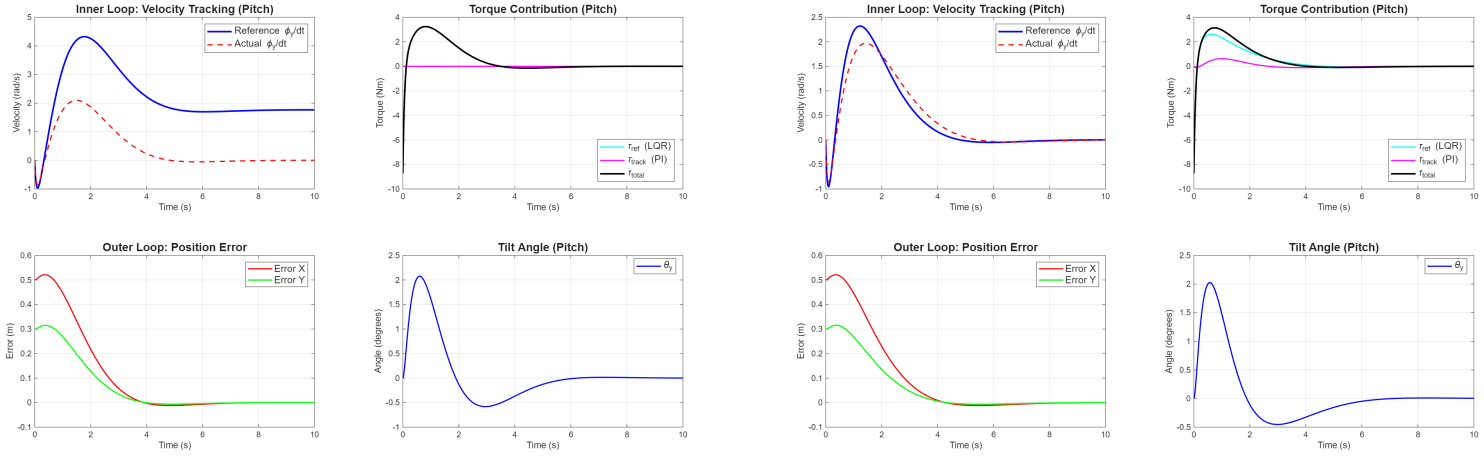


Figure 4.1: Cascaded LQR-PI control architecture. LQR computes reference torque τ_{ry} based on state error s_{ey}. A reference model predicts desired speed $\dot{\phi}_{ry}$, which an inner PI loop tracks by generating corrective torque τ_{ey} based on speed error $\dot{\phi}_{ey}$. Total applied torque is $\tau_y = \tau_{ry} + \tau_{ey}$.

As shown in Fig. 4.2, when only the LQR controller is employed, the system successfully reaches the desired target position, resulting in a vanishing steady-state position error. However, the internally generated velocity reference does not converge to zero but rather settles at a non-zero constant value. This occurs because the reference model evolves according to the nominal linearized dynamics: without integral compensation, the LQR controller assumes that maintaining equilibrium requires a constant rolling velocity, even though the real plant naturally stabilizes at rest.

When the PI compensation is activated in the inner loop, part of the total torque is provided by the PI term τ_{track}, which effectively compensates for modeling uncertainties and deviations between the nominal and real systems. Consequently, the internally generated velocity reference $\dot{\phi}_r$ (derived from the LQR reference torque) evolves coherently with the actual plant behavior, correctly converging to zero along with the measured velocity.

It is worth noting that, despite the modification in the internal reference dynamics and the redistribution of the total torque between the LQR and PI components (τ_{ref} and τ_{track}), both control configurations achieve a comparable position-tracking performance. In both cases, the position tracking error remains approximately $\text{RMSE}_{pos} = 0.36\text{m}$. This indicates that the main contribution of the PI action is not to enhance steady-state accuracy in position, but rather to ensure dynamic consistency between the nominal reference model and the real system's response. These comparative analyses were conducted through simulations where the system was tasked with moving from the origin to a target position of $\mathbf{p}_{target} = [0.5, 0.3]^T$ meters. To assess robustness against real-world variations, these simulations incorporated **parameter uncertainties** of up to $\pm 20\%$ applied randomly to all physical parameters of the ballbot model (masses, inertias, dimensions).



(a) LQR (b) LQR-PI

Figure 4.2: System response comparison: LQR vs. LQR-PI.

4.2 Model Reference Adpative Control Strategy

The previous section has shown that model-based strategies can effectively stabilize the ballbot and achieve desired task performance but their efficacy can decrease under certain conditions such as unmodeled dynamics, variable payloads, and external disturbances—which cause the physical system to deviate from its mathematical model. In the presence of real-world parametric uncertainties or during more complex tasks, the combination of strong non-linearities and system underactuation can lead to significant performance degradation. This requires the investigation of more sophisticated control strategies capable of overcoming these limitations.

For this reason, an advanced control strategy was developed to allow the controller to adapt

to such uncertainties. In this work, a Model Reference Adaptive Control (MRAC) strategy is proposed with the dual goal of stabilizing the system and ensuring precise trajectory tracking. This approach involves defining a reference model that has the desired dynamic behavior and implementing an adaptive law to continuously adjust the controller's internal parameters to minimize the error between the robot's actual behavior and this ideal response. This ensures that the actual system's response tracks the behavior of the ideal reference model, even in the presence of modeling inaccuracies and disturbances.

This section details the design of the MRAC controller, its implementation, and simulation results that validate its effectiveness in enhancing the robustness and adaptability of the ballbot control system.

4.2.1 Control Architecture

The control architecture is designed based on the simplification that the ballbot's dynamics can be decoupled into the three perpendicular vertical planes: the xz-plane (pitch) and the yz-plane (roll), along with the rotational yaw dynamics around the vertical axis. This semplification allows the complex multi-input, multi-output (MIMO) control problem to be treated as three independent single-input, single-output (SISO) control problems. Overall the control system is structured into three separate, dedicated adaptive controllers each of them with their distinct reference model. This modular approach significantly simplifies the design, implementation, and tuning process. Each adaptive controller can be developed and optimized independently for its specific plane of motion, enhancing the robustness of the overall system.

Reference Model Design

The reference model serves as the leader within the control architecture, as it defines the desired mathematical behavior that the actual system is intended to emulate. Following the modular control approach, three distinct linear reference models are designed, one for each plane of motion.

The design process for each model begins with the ballbot's full nonlinear dynamics, which are linearized around the unstable upright equilibrium point. From this linearized model, the dynamic matrices corresponding to the decoupled motion in each plane are extracted. This procedure yields three independent, linear single-input, single-output (SISO) systems:

- **Pitch and Roll Planes:** For the sagittal (xz) and frontal (yz) planes, the resulting models are fourth-order. The state variables for each plane encompass the robot's tilt angle tilt angular rate, and wheel's angular position and angular velocity (e.g., $[\theta,\ \dot{\theta},\psi,\ \dot{\psi}]$).

- **Yaw Plane:** For the transversal (yaw) plane, the dynamics are simplified to a second-order model. The state variables are the yaw angle and yaw angular rate $[\theta,\ \dot{\theta}]$. This simplification is justified by the assumption that the ball's spinning motion relative to the ground is negligible.

For each of these linear SISO systems, a stabilizing state-feedback controller is designed. Pole placement is used to assign the closed-loop poles of each plane model to achieve a desired dynamic response. The specifications for pole locations are driven by the two primary control objectives:

1. **Positioning:** This involves tracking reference trajectories and can be specified with a slower smoother response.

2. **Balancing:** This requires a faster response to ensure stability while tracking the reference trajectory.

We deliberately design the feedback to obtain the reference model to act as two decoupled second-order system, each corresponding to one of the two behaviors. This leads to a characteristic polynomial with the following form:

$$(s^2 + 2\zeta_1\omega_1 + \omega_1^2)(s^2 + 2\zeta_1\omega_2 + \omega_2^2)$$

This approach allows for intuitive specification of the natural frequency (through the desired settling time) and damping ratio of each mode. In particular, we selected a settling time of 3 seconds for the position dynamics and deliberately set the balancing dynamics to be three times faster to ensure rapid stabilization of the tilt angle.

After computing the feedback gain, a feedforward gain is incorporated into the reference model to ensure proper tracking of the external commands $r(t)$, which correspond to the desired sphere wheel angular position (for x and y axes) and upper body orientation (for the z-axis). The final structure of each reference model is given by:

$$\dot{x}_m = (A - BK)x_m + BK_r r \tag{4.2}$$

where (A, B) are the matrices of the linearized SISO system for a given plane, K is the state-feedback gain from pole placement, and K_r is the feedforward gain ensuring that the model output correctly follows the reference r.

Adaptation Law

The adaptation law represents the core of the Model Reference Adaptive Control (MRAC) strategy. It enables the controller to dynamically adjust its internal parameters in real time to minimize the tracking error between the actual system and the reference model. In the proposed architecture, the adaptive law computes three adaptive gains: $K(t)$, $K_r(t)$, and $K_e(t)$, which respectively correspond to the state feedback gain(controller reaction to current state), reference feedforward (controller compensation to track the current reference input) and a robustifying term (adds a switching action to compensate unmodelled dynamics to ensure asymptotic convergence) . These adaptive parameters continuously evolve to ensure that the system output accurately follows the reference model output, even when the system experiences large disturbances or unmodeled effects.

Control law formulation:
The adaptive control input in each plane is defined as:

$$u(t) = K(t)x(t) + K_r(t)r(t) + K_E(t)sgn(y_e(t)) \tag{4.3}$$

where:

- $x(t)$ is the state vector of the plant,
- $r(t)$ is the reference input of the plane,
- $y_e(t) = |B^T P(x_m(t) - x(t))|$ is the projection of the state error along the input direction.

Each gain evolves according to an update rule based on the system's instantaneous error. For example for K and K_r, we have a PI-type adaptive law,

$$K(t) = \alpha \int_0^t (y_e(\tau)x^T(\tau))d\tau + \beta(y_e(t)x^T(t))$$

where the integral part accumulates the adaptation effect over time in order to minimize steady-state error, while the proportional part provide immediate reaction to current error. The control law also includes a robustifying switching, whose corrective action is dominated by the K_E gain, with the following adaptation law:

$$K(t) = \gamma \int_0^t |y_e(\tau)|d\tau$$

Here, the gain grows with the magnitude of the error, resulting in stronger corrective action for larger deviations. To improve numerical stability and prevent slowdowns in the Simulink

simulation, the discontinuous term $sgn(y_e(t))$ was replaced by a smooth approximation using the hyperbolic tangent function, $tanh(y_e(t)$.

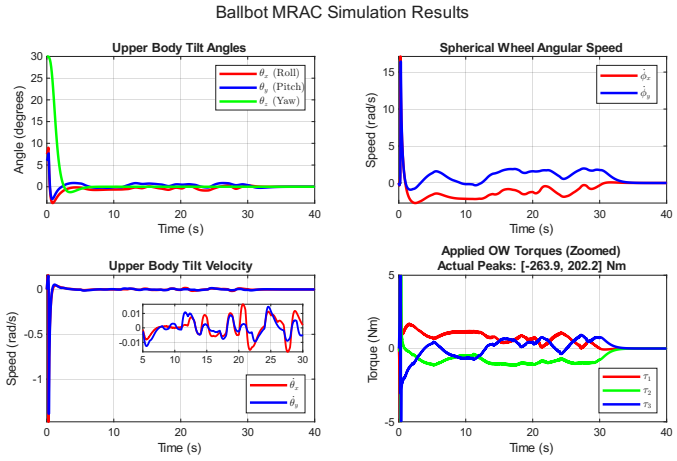
A further enhancement of this control scheme involves the use of a **gain-scheduled reference model**, in which the state-space matrices depend on the desired tilt angle around which the ballbot is required to stabilize, not necessarily the upright position. This approach is particularly useful when compensating for external disturbances while the ballbot follows a trajectory that requires nonzero tilt angles, as we will see in the following chapter. The objective is to construct a set of reference models around different values of tilt configuration and compute the corresponding state-feedback controllers to stabilize the system around those points. By interpolating the resulting closed-loop controllers corresponding to different tilt angles, the MRAC controller can determine the desired system behavior for the real ballbot even when the tilt angle θ is not zero.

Tuning Gains Parameters

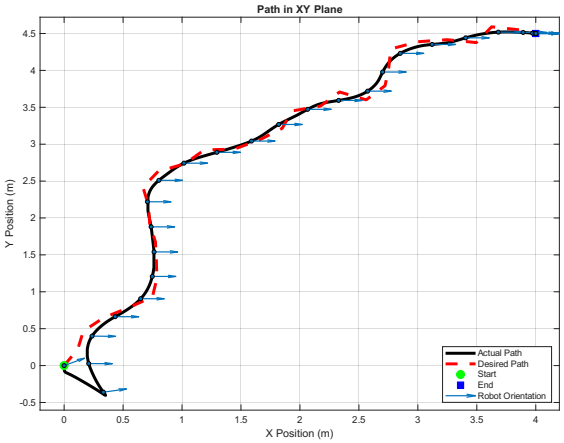
The α, β, γ constants are adaptation gains that determine the speed and aggressiveness of the learning process. These gains need to be carefully tuned to balance fast adaptation with system stability. To determine the optimal adaptation gains for the MRAC controller, we employed a systematic grid search methodology. This approach evaluates the controller's performance over a predefined discrete grid of parameter values. For each combination of (α, β, γ) in the search space, a full simulation is executed, and a composite cost J is calculated. This cost function is a weighted sum of normalized performance metrics, designed to quantify tracking accuracy, adaptation speed, ballbot stability, and physical feasibility. The metrics include the Integral of Time-weighted Absolute Error (ITAE) for tracking performance, the settling time T_s of the adaptive gains (calculated as the last instant among all gains the derivative norm exceeds 5% of its peak), the maximum upper body tilt angle as a measure of stability, the total energy consumed via the L2-norm of the control input, and a term to enforce actuator constraints, which imposes a high cost on any input surpassing the 20 Nm torque limit. The parameter set $(\alpha = 100, \beta = 2, \gamma = 10)$ yielding the minimum composite cost J is then selected as the optimal tuning for the controller.

Simulation Results

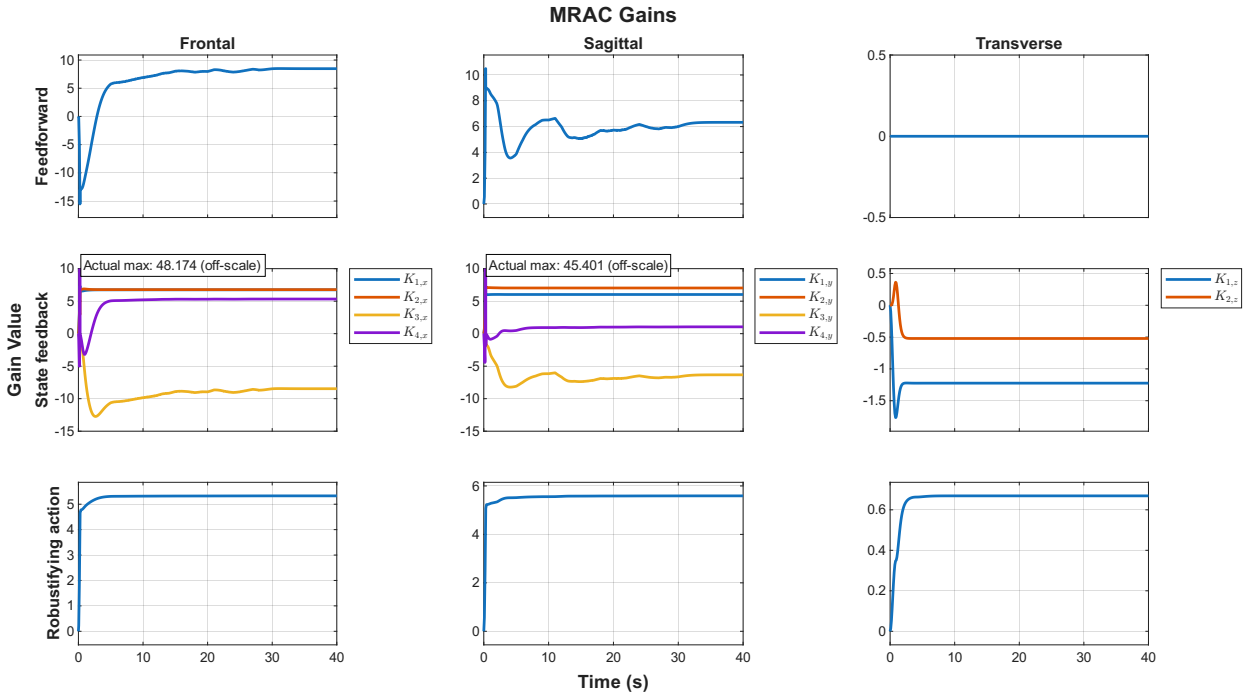
The simulation results demonstrate that the MRAC controller is highly effective in achieving the primary control objectives of balancing and trajectory tracking; however, they also reveal a critical issue with initial control saturation that would be problematic in real-world implementation. As shown in Fig. 4.3a, the controller rapidly stabilizes the ballbot, driving both the pitch (θ_y) and roll (θ_x) angles to zero (upright position) within approximately 5 seconds, while the desired orientation is simultaneously corrected and maintained. The small residual oscillations in tilt angles and velocities indicate a well-damped transient response and confirm that the MRAC adaptation effectively compensates for model uncertainties. The XY-plane trajectory in Fig. 4.3b shows that the actual path closely follows the desired reference, with only minor deviations during curvature transitions, validating accurate path tracking performance. The MRAC gain evolution (Fig. 4.3c) highlights convergence of all adaptive parameters—state feedback, feedforward, and robustifying action—each exhibiting distinct and stable adaptation dynamics. Feedforward gains converge rapidly once the tracking error diminishes, reflecting early reference model matching, while state-feedback gains adapt more actively and dominate during the balancing phase before settling to quasi-steady values. The robustifying action gains increase quickly and saturate at moderate levels, ensuring robustness without excessive control effort. Overall, the gains exhibit bounded adaptation, absence of drift, and clear convergence trends. Regarding control effort, the applied omni-wheel torques remain within ± 5 Nm during steady navigation, but large transient spikes up to ten times the physical motor limit occur initially (see Fig. 4.3a), caused by the controller aggressively compensating for large initial errors and the fast reference model dynamics. Consequently, further refinement—such as moderating the reference model or smoothing the initial command signals—is required to limit these transients and ensure operation within physical constraints.



(a) Ballbot simulation results: state variables and applied torques.



(b) Path tracking in the XY plane.



(c) MRAC controller adaptive gains evolution.

Figure 4.3: Simulation results for the Ballbot Navigation with MRAC controller.

Chapter 5

Dynamic Interaction Control

The integration of a robotic manipulator onto a dynamically stable platform like the ballbot introduces a complex set of control challenges, transforming the system into a mobile manipulator whose base and arm dynamics are **intrinsically coupled**. This chapter addresses the control problem of this complex system by decomposing it into two distinct but related viewpoints:

1. **The Ballbot Perspective: Stability and Disturbance Rejection.** From the viewpoint of the ballbot, the manipulator is treated as a time-varying source of dynamic disturbances. The shift in the system's combined center of mass, due to both the static posture and the motion of the arm, introduces moments that must be actively compensated by the base controller to maintain stability. The primary objective from this perspective is to ensure the platform remains robustly balanced, rejecting the disturbances generated by any manipulation task.

2. **The Manipulator Perspective: Task Execution on a Mobile Base.** Dually, from the viewpoint of the manipulator, its base is no longer a fixed inertial reference but a mobile and unstable platform. To successfully execute a task and position the end-effector with the required precision, the arm's controller must be aware of and account for the base's motion.

(a) (b)

Figure 5.1: Comparison of sagittal plane models: (a) The coupled dynamic model showing the manipulator mounted on the ballbot base. (b) The simplified model for the base, where the manipulator's dynamic effects are represented as an equivalent external wrench (forces F_{px}, F_{pz} and torque τ_{py}) applied at point P.

5.1 Modeling External Interactions

To analyze the effect of the manipulator on the ballbot's stability, the standard dynamic model is extended. The core of this approach is to represent the complex forces and moments generated by the manipulator as a simplified, equivalent **external wrench** applied to the ballbot's chassis. The provided image illustrates this modeling simplification.

Figure 5.2 depicts the fully coupled physical system, where the multi-link manipulator is mounted on the tilting upper body of the ballbot. For the purpose of controlling the base, the approach is simplified : the physical manipulator is not considered, and its dynamic effects—resulting from the weight, inertia, and motion of its links—are lumped into an equivalent set of forces (F_{px}, F_{pz}) and a torque (τ_{py}) applied at a single, fixed point **P** on the chassis.

This simplification is incorporated into the **Euler-Lagrange formulation**. The Lagrangian of the ballbot base, $\mathcal{L} = T - V$, remains unchanged. However, the external wrench from the manipulator is added to the **generalized forces** (Q_i) on the right-hand side of the Lagrange equations. This method effectively captures the influence of the manipulator on the ballbot's balancing dynamics without needing to solve the full, combined kinematics and dynamics of the entire mobile manipulator in the base controller. In the formulation of the external disturbance model, a strategic simplification is made. The vertical force component, F_{pz}, which corresponds to the weight of the manipulator, is **deliberately excluded**. This choice is made to isolate the controller's response from the significant destabilizing moment known as the **P-Δ effect**, which arises when the vertical load is displaced horizontally during the ballbot's tilt.

Conversely, the lateral force components, F_{px} and F_{py}, are **retained** in the model. These forces represent the static imbalance caused by the manipulator's overhanging center of mass. By adopting this simplified wrench, the analysis focuses on validating the controller's capability to counteract the persistent lateral disturbances inherent to the system's geometry, without the compounding instability from the gravitational load.

Modeling Base Motion Effects on the Manipulator

Just as the manipulator's movements create disturbances for the ballbot, the base's dynamic motion induces significant inertial forces on the manipulator itself. As the ballbot translates and tilts to maintain balance, it imparts accelerations to the base of the manipulator. These accelerations generate inertial forces (including Coriolis and centrifugal effects) that propagate through the arm's links and are felt as disturbance torques at each joint. Accurately modeling these effects is critical for precise end-effector control. Within the Simscape model, the manipulator's base is rigidly connected to the ballbot's upper body frame. Consequently, the software automatically computes the complex inertial forces arising from the base's translation and tilting.

5.2 Hybrid Compensated LQR-PI Control

The hybrid compensated LQR-PI controller provides robust stabilization and velocity tracking in the presence of **external forces and torques**, such as those generated by manipulator interactions. The control architecture combines an LQR–PI cascade with a feedforward compensation term derived from equilibrium conditions, ensuring accurate motion control under dynamically varying loads.

At each control step, the steady-state tilt angle (θ_{EQ}) and motor torque (τ_{EQ}) corresponding to the desired wheel velocity ($\dot{\phi}_{cmd}$) are derived from these equilibrium conditions. This formulation centers the control action around a dynamically consistent operating point, enhancing disturbance rejection and steady-state accuracy.

The controller is termed **hybrid** due to its context-dependent strategy. For instance, the yaw control component switches its reference from a trajectory-dependent desired yaw angle to a zero-orientation target once the final segment of motion is reached.

5.2.1 Compensation Module

The compensation module enables the controller to proactively counteract steady external forces and torques while maintaining a desired velocity. To sustain constant translational motion, the BallBot must lean at a specific angle (θ_{EQ}) and apply a constant torque (τ_{EQ}) to balance gravity, inertial effects, and any external wrench acting on it. This module computes the corresponding *dynamic equilibrium conditions*—specifically, the tilt angle and motor torque that yield zero angular acceleration for both the body and the spherical wheel ($\ddot{\theta} = 0, \ddot{\phi} = 0$)—given the current velocity command $\dot{\phi}_{cmd}$ and the measured external wrench. Using the system's nonlinear dynamic model, a numerical solver determines these equilibrium values, which are then applied as feedforward terms: the equilibrium angle adjusts the LQR's reference orientation, while the equilibrium torque is added to the output of the feedback controllers. Evaluating the LQR around the estimated equilibrium angle θ_{EQ} effectively shifts the controller's operating point to the dynamically balanced state. This approach allows the feedback loops to regulate only local deviations around equilibrium rather than compensating for the full external disturbance. The computation is carried out independently along the pitch and roll axes, assuming near-decoupled dynamics around the upright configuration.

5.2.2 Yaw-Control Module

Control of the BallBot's orientation about the vertical axis (yaw, θ_z) is handled independently from the pitch and roll stabilization loops. The yaw motion is largely decoupled from the inverted pendulum dynamics that govern translational stability, behaving instead as an inertial disk driven by motor torque and affected by friction. Consequently, a conventional Proportional-Integral-Derivative (PID) controller is adopted to ensure effective yaw angle tracking.

The determination of the desired yaw angle ($\theta_{z,des}$) is less straightforward, especially under the influence of external forces generated by the manipulator. A useful strategy is to orient the BallBot so that the main component of the external force aligns with the direction of motion along the planned trajectory. In principle, such alignment allows the external force to assist forward motion or braking, reducing the effort demanded from the actuators. The desired yaw is therefore computed dynamically by comparing the direction of the external force vector, expressed in the robot's frame, with the direction of the next trajectory segment.

Despite its conceptual advantage, this approach introduces practical difficulties. Instantaneous reorientation is not feasible, and any rapid yaw rotation produces reaction torques that disturb the balance. Moreover, rotating the manipulator relative to the base alters the external wrench applied to the BallBot, resulting in time-varying disturbances. The Compensation Module mitigates these effects by continuously updating the equilibrium conditions in response to the changing wrench, though its effectiveness is limited by the system's dynamic response.

To ensure smooth reference evolution, the yaw reference computation includes an unwrapping mechanism that preserves continuity across $\pm 180°$, allowing for seamless multi-turn rotations without discontinuities in the control signal. The PID controller then applies the appropriate yaw torque (τ_z) based on the error between this continuous desired angle and the measured yaw, driving the orientation error toward zero.

5.2.3 Hybrid Control Logic and Operational Phases

The control architecture is implemented as a hybrid system composed of four sequential operational phases, each governing a specific stage of navigation, stabilization, and manipulation. The control logic defining the transitions and objectives of each phase is outlined below.

1. **Phase 1: Initialization Phase**
   The system begins with a 4-second initialization phase, during which the manipulator remains in its home configuration, exerting a static external wrench on the BallBot. The base controller aims to regulate the platform's position around the origin $[0, 0]$ and converge to

the corresponding dynamic equilibrium posture (tilt angle θ_{eq}) that compensates for this initial wrench. A proportional (P) controller generates reference velocities from the Cartesian position error, while the cascaded LQR–PI controller computes the resulting motor torques. To ensure smooth startup behavior, the feedforward compensation module remains disabled for the first 0.1 seconds of simulation.

2. **Phase 2: Trajectory Tracking Phase**
   After initialization, the BallBot enters the tracking phase, following the pre-defined trajectory defined by successive waypoints. Reference velocities are generated by the trapezoidal velocity profiling logic (see Section 3.1.1). During this phase, the yaw-control module actively adjusts the robot's orientation while the base translates along the path. As the BallBot approaches the final segment of the trajectory, the reference velocity is progressively reduced for braking, and yaw control is suspended to prepare for the transition to precise stabilization.

3. **Phase 3: Stabilization Phase**
   Upon reaching the vicinity of the goal position, the controller switches from tracking to regulation. The velocity reference is now generated by a Proportional–Integral (PI) position controller, which eliminates steady-state error and ensures accurate parking at the target. Simultaneously, the yaw angle is regulated to zero to align the manipulator for task execution. The system is considered stable once the BallBot remains within a 0.05 m radius of the goal position for at least one second while maintaining zero yaw error. Only under these conditions does the system transition to the manipulation phase.

4. **Phase 4: Manipulation Phase**
   In the final phase, the manipulator executes the pick-and-place motion, generating time-varying external wrenches on the BallBot. The controller compensates these disturbances in real time through the equilibrium compensation module, which adjusts the steady-state tilt angles (θ_{eq}) in both sagittal and frontal planes to maintain balance. A hybrid-state safety mechanism continuously monitors the BallBot's stability: if the base drifts beyond acceptable bounds, the system automatically reverts to Phase 3. During this fallback, manipulator motion is paused until balance is reestablished, after which the manipulation resumes seamlessly.

5.2.4 Parameter Tuning

Tuning the parameters involved in the BallBot's control architecture represents a fundamental step to ensure stable and accurate motion. The procedure focused on the parameters directly affecting the **balancing and motion control tasks**, considering both regulation and disturbance rejection performance.

Two distinct experimental campaigns were carried out to analyze the controller's behavior under different operating conditions:

- **Experiment 1 – Static Manipulator:** The manipulator was kept stationary while the BallBot started from random initial conditions and was required to regulate its position to the origin. This setup isolates the intrinsic balancing capability of the base.
- **Experiment 2 – Active Manipulator:** The manipulator executed a planned pick-and-place trajectory while the BallBot simultaneously compensated for the resulting interaction forces and moments. This configuration assesses the controller's ability to reject dynamic disturbances.

Each simulation lasted 10 seconds and started from randomized initial conditions: planar offsets between -0.3 m and 0.3 m, and initial tilt angles (roll and pitch combined) between -0.07 rad ($\approx \pm 4^\circ$). Controller parameters were sampled within predefined ranges for each run, with a total of 150 simulations per experiment.

The parameters considered for tuning were:

- K_p^{pos}, K_i^{pos}: Gains of the outer position control loop, with K_i^{pos} fixed to 0.05 to correct steady-state errors without significantly affecting the transient response.
- K_p^{inner}, K_i^{inner}: Gains of the inner PI velocity control loop.
- Q_1, Q_2, Q_3, Q_4: Diagonal elements of the **Q** weighting matrix in the LQR cost function. These weights penalize deviations in tilt angle (θ), sphere angle (ϕ), and their respective angular rates ($\dot{\theta}$, $\dot{\phi}$).
- R: Weight for the control input, fixed at $R = 1$.

Several **performance metrics** were extracted from each simulation to quantitatively evaluate the system behavior:

- **RMSE Metrics:** Root-mean-square errors were computed for the BallBot's planar position (*Position RMSE*), for the body tilt angle representing balance precision (*Tilt RMSE*), and for the end-effector position (*Operational-Space RMSE*, relevant only during manipulator motion). These indicators collectively quantify tracking accuracy and overall stability.
- **Maximum Deviation:** Largest instantaneous deviation from the upright position recorded during the simulation.
- **Convergence Time:** Defined only for the first experiment (static manipulator). It represents the earliest instant when the BallBot's distance from the origin remains below a threshold $\epsilon = 0.01\,\text{m}$ for a continuous duration of 1 s, i.e. $d(t) < 0.01\,\text{m} \;\; \forall\, t \in [t_i,\, t_i + 1.0\,\text{s}]$.

All results were aggregated into a structured dataset pairing each parameter configuration with the corresponding performance outcomes. Statistical analyses were performed using both **Pearson** and **Spearman** correlation coefficients to investigate dependencies between control parameters and performance indices. The two analyses yielded qualitatively consistent results, indicating similar trends in parameter influence. Consequently, only the **Spearman correlation tables** are presented and discussed, as they better capture the monotonic relationships between parameters and system performance.

The resulting correlation heatmaps enable a direct interpretation of how parameter variations affect regulation quality and disturbance rejection, providing a data-driven foundation for final parameter selection.

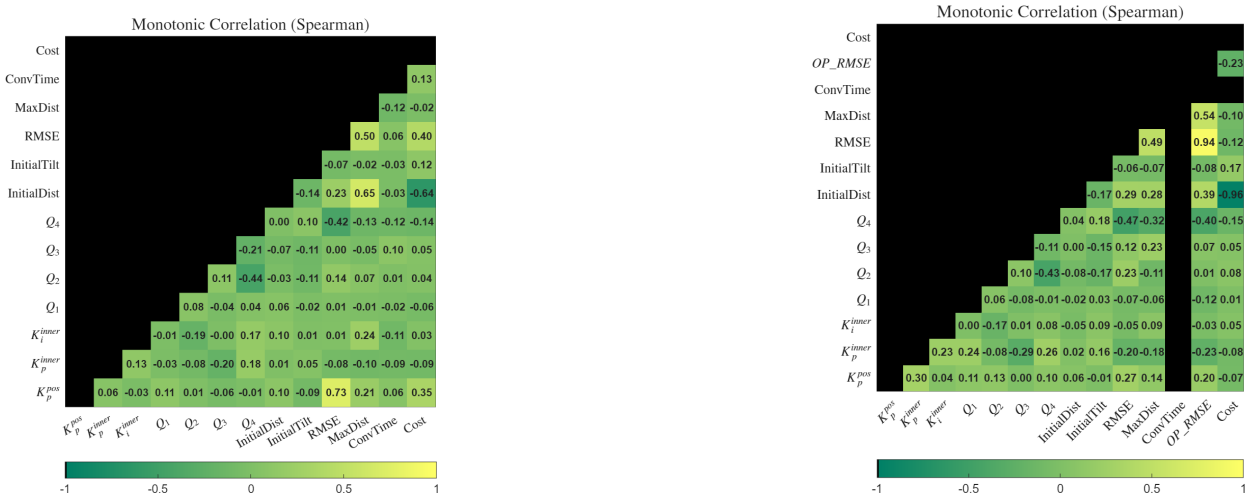


Figure 5.2: Spearman correlation matrices: (left) static manipulator, (right) active manipulator.

5.2.5 Simulation Results

During the *Navigation Phase*, the BallBot exhibits stable dynamic behavior while tracking the precomputed trajectory. The upper body tilt angles oscillate within acceptable limits, approximately between $\pm 5°$, indicating that the controller successfully maintains balance while in motion. The oscillatory behavior, also evident in the peaks of the Omni-Wheel Torques, originates from the waypoint-following mechanism: each waypoint update introduces a sudden change in the position error. The controller immediately reacts to this discontinuity by applying a sharp torque peak to compensate for the increased tracking error, a behavior clearly visible in the torque plots. Throughout the trajectory, the yaw angle evolves with its own dynamics, rotating up to nearly $-180°$ before returning toward zero. The base velocity remains below 1.5 km/h: this is relatively slow for an inherently unstable platform, higher velocities could be achieved by increasing the maximum velocity and acceleration parameters (v_{max} and a) used to generate the reference $\dot{\phi}_c$. With the current tuning, the system favors stability and accuracy over speed, achieving a mean tilt angle of 2.28°, a trajectory RMSE of 0.072 m, and a maximum deviation from the reference path of 0.247 m, thus allowing the use of a smaller inflation radius in the planner. In the final seconds of this phase, all dynamic variables gradually settle as the BallBot approaches the goal position, preparing for the stabilization and subsequent manipulation phases. Results of this phase can be seen in Figure 5.3.

Once stabilized, the *Manipulation Phase* begins. During this stage, the manipulator executes the pick-and-place task while the BallBot compensates for the induced dynamic loads. The tilt angle of the upper body is not maintained at zero but varies, reaching values up to approximately 10°. This implies that the manipulator's base is inclined while performing the task, an effect that is not explicitly compensated in a joint-space control scheme and can therefore introduce positional errors in the manipulation. The applied torques and external disturbances exhibit high-frequency oscillations, while the overall wrench transmitted by the manipulator to the BallBot increases over time. Results of this phase can be seen in Figure 5.4.

The operational-space error is defined as the Cartesian distance between the actual end-effector position and a fixed virtual reference at the goal. This error arises from two main sources: the non-zero equilibrium tilt ($\theta_{eq} \neq 0$) imposed by the compensation module, and small oscillations of the BallBot's base caused by active disturbance rejection.

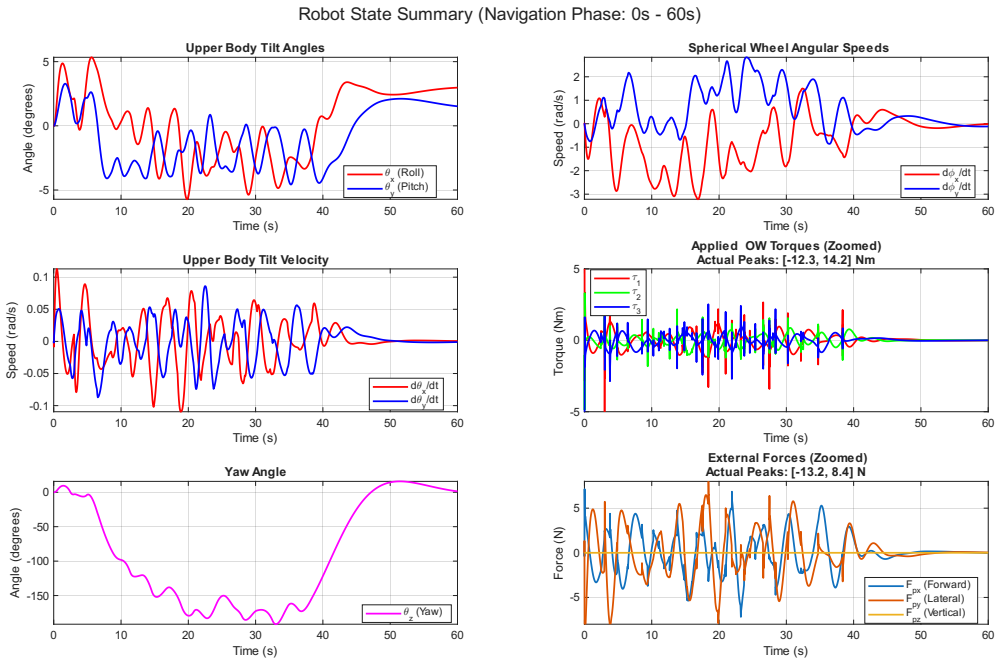


Figure 5.3: Robot state summary during the navigation phase, showing path tracking dynamic.

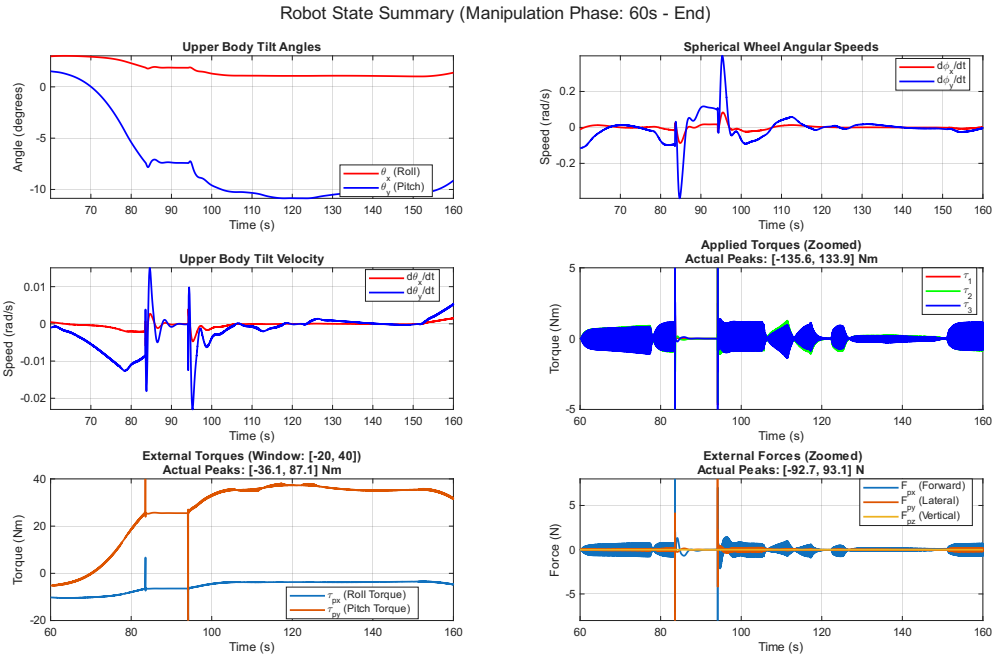


Figure 5.4: Robot state summary during the manipulation phase.

5.2.6 Operational-Space Feedback Compensation Attempt

The definition of the operational-space error enables the synthesis of a corrective control action that exploits this feedback information to actively compensate for deviations induced by the platform motion. An operational-space feedback correction strategy was implemented to enhance manipulation accuracy during dynamic balancing. The approach consisted in augmenting the manipulator's nominal joint-space trajectory with a corrective term computed from the operational-space error and mapped back to the joint space and scaling this term by a coefficient $k_{op} \in [0, 1]$, with the intent of compensating for base inclination and residual coupling effects between the Ballbot and the manipulator.

However, this approach did not yield the desired improvement in performance. The corrective action required to minimize the operational-space error necessarily involved motion of the manipulator, which in turn produced a dynamic wrench on the Ballbot's base. The stability controller of the Ballbot then reacted to this disturbance by modifying its tilt and adjusting its position to restore balance. This induced motion of the base altered the manipulator's reference frame, reintroducing or even amplifying the same operational-space error the correction was designed to eliminate. As a result, the interaction between the two control objectives—manipulation accuracy and base stability—became antagonistic, ultimately preventing convergence and demonstrating the inherent challenge of simultaneous disturbance rejection and end-effector precision on a dynamically balancing platform.

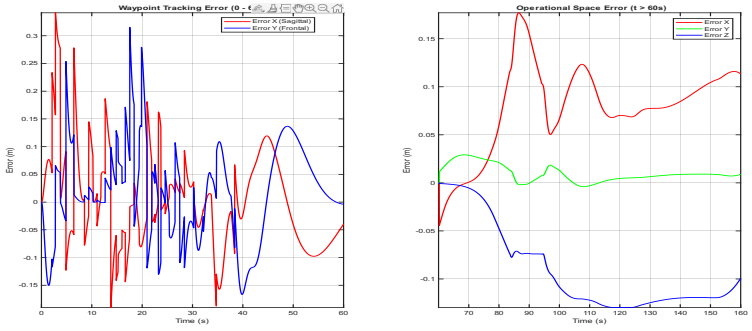


Figure 5.5: Tracking error (nav phase), operational space error (man phase)

Bibliography

[1] Roberto Rocco and Chiara Panagrosso. Pick & place in a blocksworld environment using htn planning and moveit. `https://github.com/Robertorocco/Pick_Place_Blocksworld_Environment`, 2025.

[2] Chenzhang Xiao, Seung Yun Song, Yu Chen, Mahshid Mansouri, João Ramos, Adam W Bleakney, William R Norris, and Elizabeth T Hsiao-Wecksler. Exploiting physical human-robot interaction to provide a unique rolling experience with a riding ballbot. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*. IEEE, 2025.