Final Project Report: Giraffe Robot for Q&A Sessions

Martina Panini

July 2025

Abstract

This report details the design and control of a "Giraffe" robot, a robotic system aimed at automating microphone handling during Q&A sessions in conference rooms or theaters. The project addresses the tasks outlined in the final project assignment, covering robot modeling, kinematics, dynamics, trajectory planning, and task-space control with redundancy resolution.

1 Robot Description

The Giraffe robot is designed to be attached to the ceiling of a room. The robot is required to reach locations up to 1 meter high within this area. It possesses 5 Degrees of Freedom (DoF) to achieve its tasks:

- 1 spherical joint at the base, modeled as 2 revolute joints with intersecting axes.
- 1 prismatic joint, enabling significant extension for reach.
- 2 revolute joints, for precise orientation of the microphone.

The robot's primary task is 4-dimensional, controlling the `X, Y, Z` position, and the pitch orientation of the microphone. A specific pitch orientation of 30 degrees with respect to the horizontal is required for comfortable speaking. The system inherently has 1 DoF redundancy, which is exploited to perform a secondary task.

2 Workplan and Implementation Details

The project was structured into several incremental steps, each building upon the previous one. The following sections detail the resolution of each task with references to the implemented code.

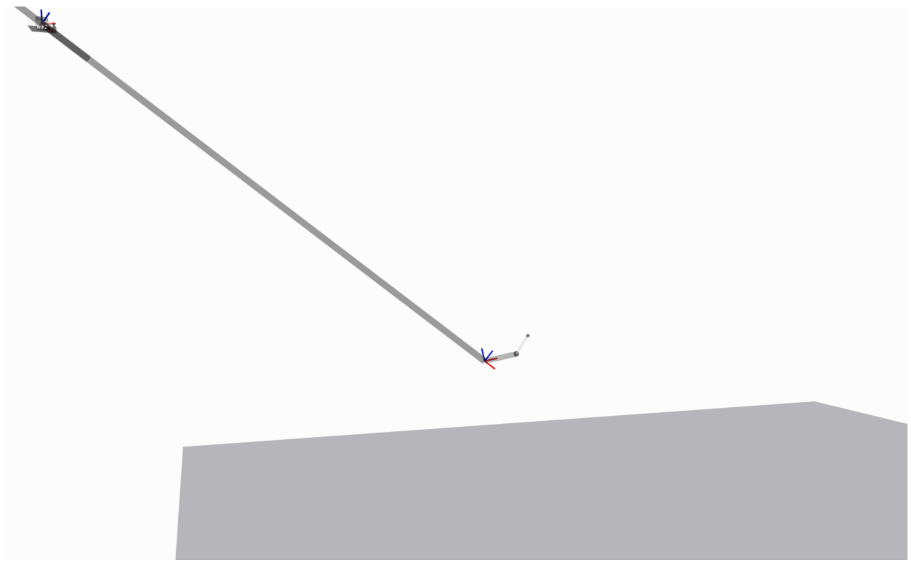


Figure 1: Robot final configuration

2.1 Build the URDF model of the robot

The Unified Robot Description Format (URDF) model of the robot, including link dimensions and frame placements, is defined in `giraffe.urdf.xacro`. The process of generating and loading the URDF model into Pinocchio is handled by the `getRobotModel` function. This function generate the URDF file and then building the Pinocchio robot model from this file. Robot parameters like link lengths, masses, center of masses (COMs), and inertia tensors are explicitly defined in the `setRobotParameters` function.

2.2 Forward and Differential Kinematics

The forward kinematics (FK) and differential kinematics (Jacobian) of the end-effector are implemented and tested.

- **Direct Kinematics:** The `directKinematics` function calculates the homogeneous transformation matrices from the world frame to each robot link, based on the URDF structure and joint values. This function provides the end-effector's position and orientation. A test function, `dk_test` compares the custom direct kinematics implementation with Pinocchio's built-in functions for validation.
- **Differential Kinematics:** The `computeEndEffectorJacobian` function computes the geometric Jacobian of the end-effector. It correctly identifies the joint origins and axes for each joint to form the Jacobian matrix. The `geometric2analyticJacobian` function converts the geometric Jacobian to the analytic Jacobian, which is crucial for task-space control, especially for orientation tasks involving Euler angles.
  The `jacobian_test` function validates the custom Jacobian computation against Pinocchio's internal Jacobian calculation.

2.3 Pinocchio library's RNEA function

Pinocchio's Recursive Newton-Euler Algorithm (RNEA) is used as the core of the robot's dynamic simulator. The `dyn_simulation` function utilizes `pin.rnea` to compute the joint torques required to achieve a given state, and crucially, `pin.aba` (Articulated Body Algorithm) for forward dynamics, which computes joint accelerations from applied torques and non-linear effects. As observed in initial tests, without active control, the robot would yield under gravity, highlighting the necessity for the robust control strategies implemented later.
This simulation is computed for a determined time interval (set in the configuration file).

2.4 Polynomial trajectory in the task space

A fifth-order polynomial trajectory is planned in the task space to move the end-effector from an initial configuration to a desired configuration. The `fifthOrderPolynomialTrajectory` function calculates the coefficients for a fifth-order polynomial, ensuring smooth transitions of position, velocity, and acceleration at the start and end points. This function is used within `generate_trajectory` to produce smooth desired trajectories for the end-effector's position and pitch orientation over a specified duration.

2.5 Inverse-dynamics (computed torque) control action in the task space

An inverse-dynamics (computed torque) control action is implemented in the task space to linearize the system and achieve precise tracking of the desired task. The `simulation` function contains the core control loop. It computes the desired joint accelerations using a task-space inverse dynamics approach. The control law follows the form $\tau = M(q)\ddot{q}_{des} + h(q, \dot{q})$, where $\ddot{q}_{des}$ is derived from the desired task-space acceleration, and M and h are the mass matrix and bias terms, respectively. The desired task-space acceleration is a function of position and velocity errors, as well as desired task accelerations from the trajectory.

2.6 PD gains of the Cartesian controller

The PD (Proportional-Derivative) gains for the Cartesian controller are defined in the configuration file.

- `Kp_pos` and `Kd_pos`: Diagonal matrices for the 3D position control, set to `diag([100., 100., 100.])` and `diag([10., 10., 10.])` respectively.

- `Kp_pitch` and `Kd_pitch`: Scalar gains for the 1D pitch orientation control, set to `10.0` and `2.0` respectively.

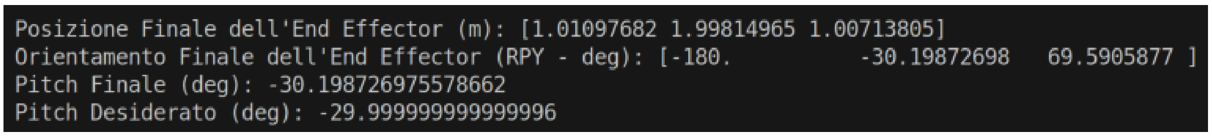
These gains are then used in the `simulation` function to compute the desired task-space accelerations based on the current errors.
Values of gains are chosen after different simulations with different gains.

2.7 Null-space of the task

To exploit the 1 DoF redundancy, a secondary task is defined in the null-space of the primary task, aiming to minimize the distance to a given postural configuration, `q0_calibrated`. The `simulation` function implements null-space projection. The term `N_task @ q_postural` is added to the desired joint accelerations, where `N_task` is the null-space projector of the task Jacobian, and `q_postural` is a PD control law applied to the difference between the current joint configuration and the desired postural configuration (`q0_calibrated`). The postural gains, `Kp_postural` and `Kd_postural`, are defined in the configuration file as `10.0` each. The `q0_calibrated` is computed using an inverse kinematics approach to ensure that the initial posture is achievable and minimizes the task error while also considering the desired end-effector pose.

2.8 Simulate the robot to reach a specific location from the homing configuration

The robot's simulation to reach the target location $p_{des} = [1, 2, 1]$ from the homing configuration $q_{home} = [0, 0, 0, 0]$ is performed by the `simulation` function. The desired end-effector position `pdes` and the desired pitch orientation `pitch_des_deg` (which is -30 degrees) are defined in the configuration file. The simulation loop iteratively updates the robot's state using the computed torque control law and integrates the dynamics, while publishing the robot's state to RViz for visualization and logging data for plotting. The results include plots of end-effector position and pitch over time, demonstrating the robot's ability to track the desired trajectory and reach the target, as is shown in Figure 3.



```
Posizione Finale dell'End Effector (m): [1.01097682 1.99814965 1.00713805]
Orientamento Finale dell'End Effector (RPY - deg): [-180.         -30.19872698   69.5905877 ]
Pitch Finale (deg): -30.198726975578662
Pitch Desiderato (deg): -29.999999999999996
```

Figure 2: Raw logged data of end-effector position and orientation during the simulation.

3 Conclusion

This project successfully developed and implemented a control system for the Giraffe robot, addressing key aspects of robot kinematics, dynamics, and task-space control. The implemented control strategies, including inverse dynamics control with null-space projection, demonstrate the robot's ability to perform complex tasks such as precise end-effector positioning and orientation while managing kinematic redundancy.

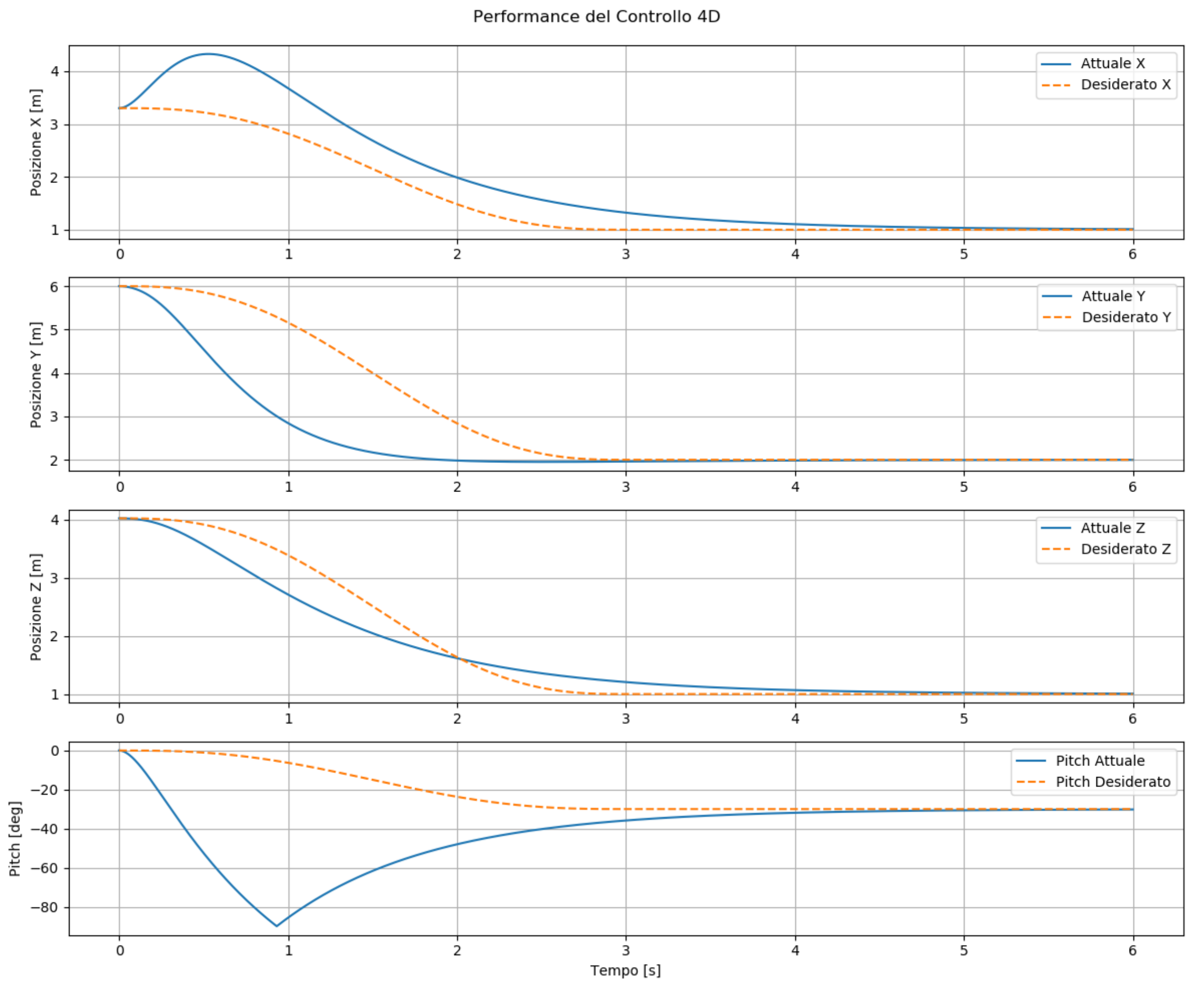


Figure 3: Tracking performance of the end-effector's X, Y, Z position and Pitch orientation (actual vs. desired) over time during the task-space simulation.