Giraffe Robot Report

Alessandra Benassi

Artificial Intelligence Systems Master's Degree
Introduction to Robotics course

University of Trento
Department of Engineering and Computer Science

Introduction

The project focuses on the design and control of a giraffe robot to automate microphone handling during Q&A sessions at talks in conference rooms. Following the specifications given in the pdf, the instructions were implemented in different simulations of kinematics, dynamics, robot visualization, trajectory planning and control in task space.

URDF design

The design of the robot was done in the `giraffe_robot.xacro` file and then the real urdf file was generated with the designated command. 1 The robot is attached to the ceiling of the room, at a 4m height, to a fixed box and is located in the middle. The floor of the room was modelled as requested with a 5×12m area. The robot has 5 degrees of freedom:

- 1 spherical joint at the base (2 revolute joints with intersecting axes). They were designed as 2 overlapping full spheres rotating in the assigned direction.
- 1 prismatic joint able to achieve a long extension
- 2 revolute joints to properly orient the microphone. Designed at the extremities of the wrist link

It was also added a microphone, modelled with a cylinder and a sphere, to simulate more realistically the end effector behaviour and trajectory tracking.

After the links, joints and dimensions modelling, in the urdf design was also involved the robot's material choice to compute the masses and inertia. After some research, the chosen material was aluminium. Due to the nature of this robot, having a ceiling attachment, the robot is supposed to be as light as possible and this material is a classic choice for its weight to strength ratio, in addition to a limited cost. For the end effector the material choice was ABS plastic, the most common for electronics and daily manufactured objects for its strength, durability, and impact resistance.

Kinematics

In the `kinematics.py` file the forward and differential kinematics are tested: the end-effector pose computed in the custom directKinematics function is compared with the result obtained using the Pinocchio library.

The *direct kinematics* of the robot was calculated from the geometric description provided by the URDF model. Each joint transformation was modelled using the homogeneous transformation matrices, considering the rotations, translations and the joint type (revolute or prismatic).

The *differential kinematics* firstly takes in input the joint configurations and computes the geometric Jacobian by considering the contribution of each joint to the linear and angular velocity of the end-effector. The result is then converted to the analytic Jacobian, useful for the following task space operations.

In the test functions, the comparison between the native and the custom functions, implemented in `kin_utils.py` showed negligible numerical differences, confirming the correctness of the kinematics model.

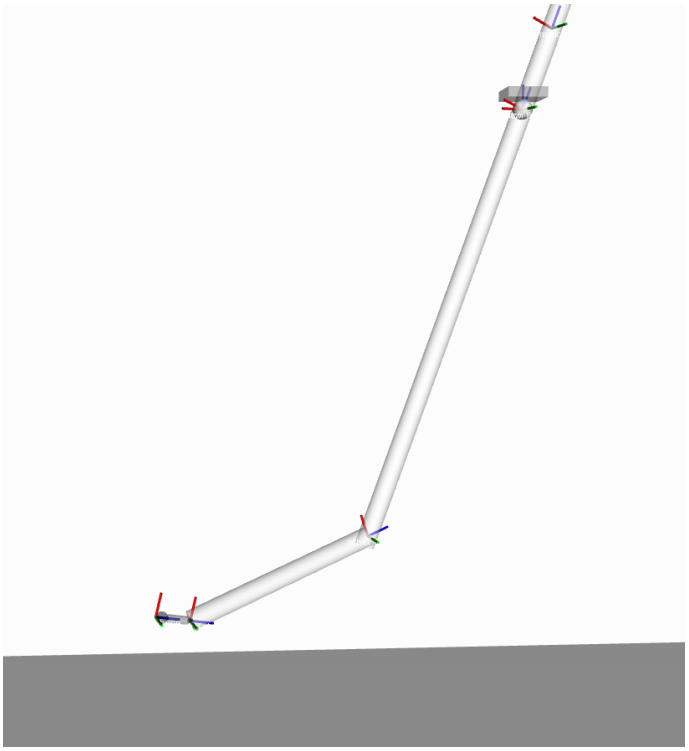

Figure 1: **Giraffe urdf showed in rviz**

Polynomial trajectory

A fifth order polynomial trajectory was implemented to obtain the robot's motion between the initial configuration and a randomized desired final configuration. Starting from the initial joint configuration, the custom inverse kinematics algorithm computes a joint configuration that satisfies the desired end-effector pose, reaching the task-space position given in input. During the simulation, the trajectory is executed by publishing the computed joint positions, velocities and accelerations (q, qd, qdd) at each time step. A visual marker was added to indicate the desired end-effector position in the task space.

The choice was a quintic trajectory because it allows the control of position, velocity, and also acceleration, while with a cubic one only the first two can be handled. The fifth order is preferred in robot control for its smoother transitions, higher stability and a better overall motion.

Dynamics

The robot dynamics is modelled using the equation: $M(q)\ddot{q} + C(q, \dot{q}) + g(q) = \tau$. These quantities were computed using the functions from the Pinocchio library. In particular, the RNEA is used to compute the inverse dynamics, that given $q, \dot{q}, \ddot{q}$ as inputs, returns the torques τ required to produce the desired motion.

To simulate the robot motion, forward dynamics is computed using the equation:

$$\ddot{q} = M(q)^{-1}(\tau - C(q, \dot{q} - g(q))$$

.

To test the correctness of the code, the computed acceleration was compared with the one given by the Pinocchio native function implementing the ABA algorithm, mentioned in theory classes. Finally, the Newton Euler integration is applied to update the joint velocities and position using the computed acceleration.

As shown in the laboratories, to add physical plausibility and feasibility, some additional torque terms were introduced: viscous damping, proportional to the joint velocity; joint limit torques, implemented as spring–damper elements near the set joint boundaries. These terms prevent unbounded and unwanted motion and stabilize the simulation when no control input is applied.

The simulation shows the giraffe robot not moving, avoiding gravity collapse or unrealistic movements thanks to the integration of these dynamics equations.

Task space

In `task_space_trajectory.py` are implemented the last steps specified in the workplan, combining all the previously seen elements. The trajectory duration is set to 7s and the goal is to reach the point $p_{des} = [1, 2, 1]$ with an angle of 30°. The control law has PD gains and follows the equations above, using the joint space acceleration $\ddot{q}$ to compute τ. To obtain $\ddot{q}$:

$$\ddot{q} = J^{\#}(\ddot{p} - \dot{J}\dot{q})$$

with

$$\ddot{p} = \ddot{p}_{des} + K_d\dot{e} + K_p e$$

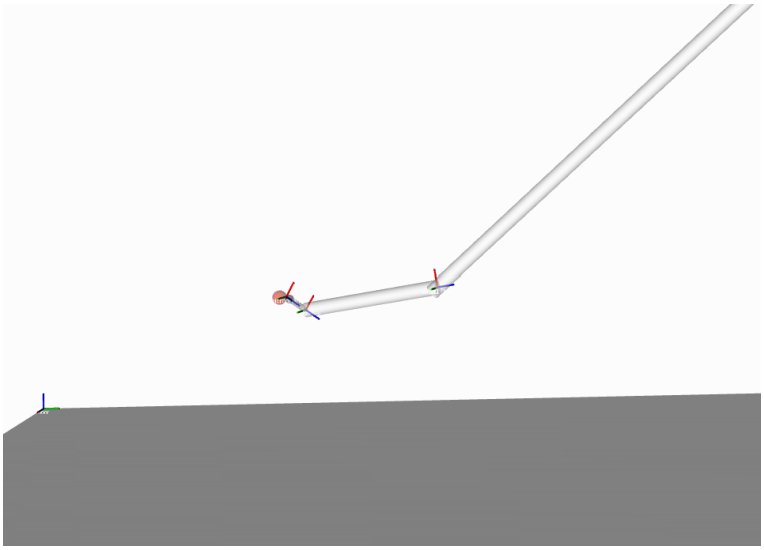

Figure 2: **Giraffe robot reaching the desired position**

Since the robot has more degrees of freedom (5) than required by the task (position+pitch=4), the redundancy had to be handled. It was exploited to maintain a posture similar to q_0 during the task, to not only reach the goal but also doing it with a desired joint configuration.

The PD gains were chosen following the formula: $K_d = 2\sqrt{K_p}$, that usually ensures good control avoiding overshoot. For position the gains were set higher due to larger inertia, especially with the design of the considered robot, for pitch are set lower for the

same reason. The postural gains are set to small values to ensure the priority is always given to the reaching of the goal position.

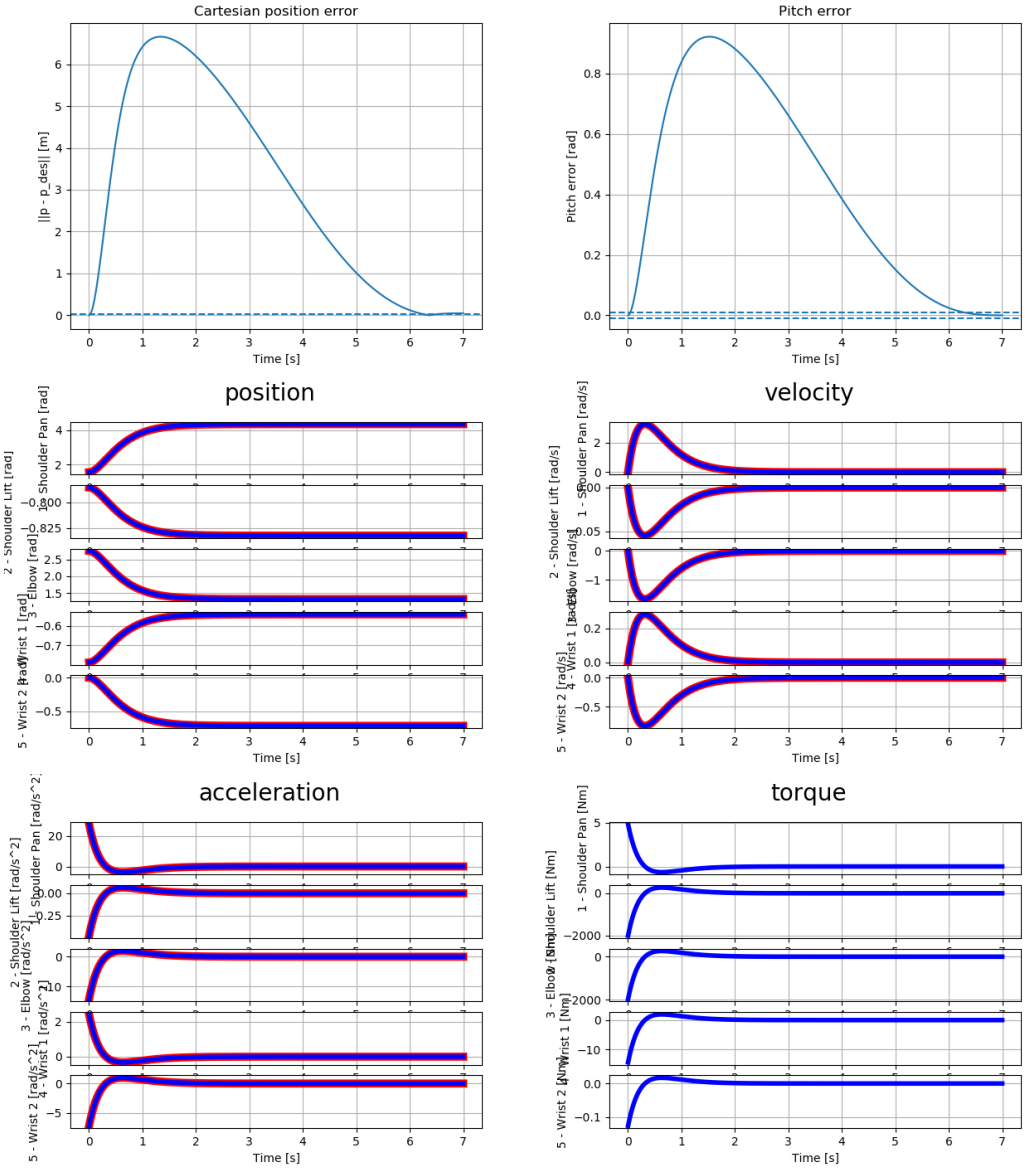

Figure 3: **Task space plots**

References

[1] URL: `https://github.com/Allelallecialle/Giraffe-robot.git`.

[2] Laboratories's code from the course.