Example of PhD Thesis with RoboticsLaTeX template

Simone Lombardi

DIBRIS - Department of Computer Science,
Bioengineering, Robotics and System Engineering

University of Genova

Supervisors:
Prof. Giorgio Cannata
PhD. Francesco Grella
PhD. Francesco Giovinazzo

In partial fulfillment of the requirements for the degree of

Laurea Magistrale in Robotics Engineering

December 17, 2025

Declaration of Originality

I, Simone Lombardi, hereby declare that this thesis is my own work and all sources of information and ideas have been acknowledged appropriately. This work has not been submitted for any other degree or academic qualification. I understand that any act of plagiarism, reproduction, or use of the whole or any part of this thesis without proper acknowledgment may result in severe academic penalties.

Acknowledgements

I want to thanks all the people that helped me during my time at University of Genova, starting with professor Cannata. His assistance was essential in the development of this thesis. I than extend my depest gratitude to Francesco Grella and Francesco Giovinazzo, them with all the other people of the MACLAB laboratory made me feel welcomed and have given me invaluable advice throughout my journey with them.

On a personal note, I want to thanks all my colleagues of the Robotics Engineering course. The friendship I found are extremely meaningful to shape me in the person I am today. Last but not least, in the slightest I want to tell my family and friends that their unwavering support and belief in me did not go unnoticed, I would not be here today if it wasn't for them.

This is a short, optional, dedication. To all the Master and PhD students of Robotics Engineering at the University of Genova.

Abstract

Since the 1960s, the use of robotic systems in industrial applications has continuously increased. However, even with this incredible force driving innovation, some tasks have proven to be too complex or not cost-effective to be performed by a robot. With the advent of Industry 4.0, the proposed solution to these problems was **Human-Robot Collaboration** — building work-cells capable of integrating a human agent performing a set of tasks that can be coordinated with a robotic agent to achieve a common objective. This approach opened up a completely new set of challenges, the first of which are safety and perception. The robotic agent needs a way to perceive the human in the workcell and must be able to react to unpredictable movements to avoid collisions. During my thesis, I worked within the **SESTOSENSO project**, specifically in Use Case 1. Their robotic system, composed of two 6-DoF industrial articulated robots mounted in series, is equipped with a set of proximity and tactile sensors. My work focused on creating a unified architecture for the two robots, exploring the capabilities of a 12-DoF robot, and proposing possible directions to improve the system's functionalities. Moreover, this work also aimed to identify potential problems and weaknesses. I achieved these objectives through a series of simulated experiments, using a task-priority approach for system control, as I was interested in exploiting the high redundancy of the robot to perform multiple tasks simultaneously. I than analyzed the result to evaluate the effect of each task on the behavior of the robot.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research problem

1.2 Thesis objective and structure

Chapter 2

State of the art

2.1 Industrial robotics

2.1.1 Early days

2.1.2 Modern approach

2.2 Collaborative robotics

2.2.1 Definitions

2.2.2 Objectives and challenges

2.3 High DOF system

2.3.1 System types

2.3.2 Macro/Micro configuration

Chapter 3

Architecture implementation

3.1 System Description

The robotic system I worked with was composed of two articulated industrial robot, namely a **Kuka KR150** from *Kuka* and a **UR10e** form *Universal Robot*.

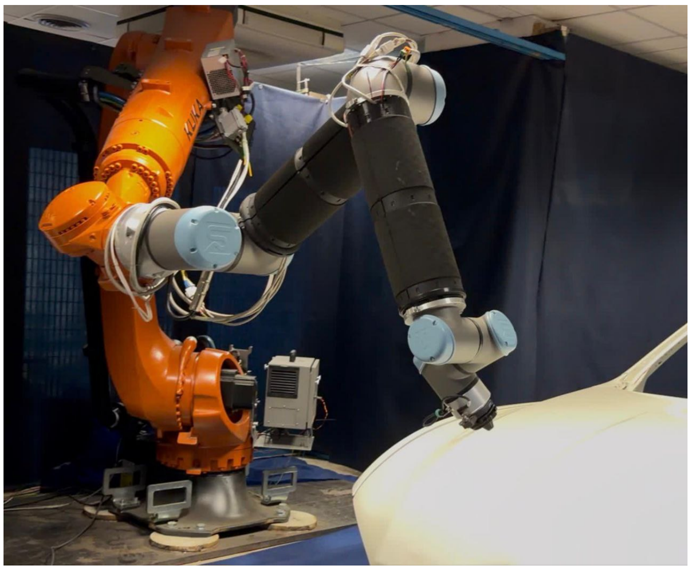

Figure 3.1: photo of the real system, inside the workcell

I started the from the work done by the team at MACLAB, and since their code already included the simulation part, I opted to incorporate their work in my architecture.

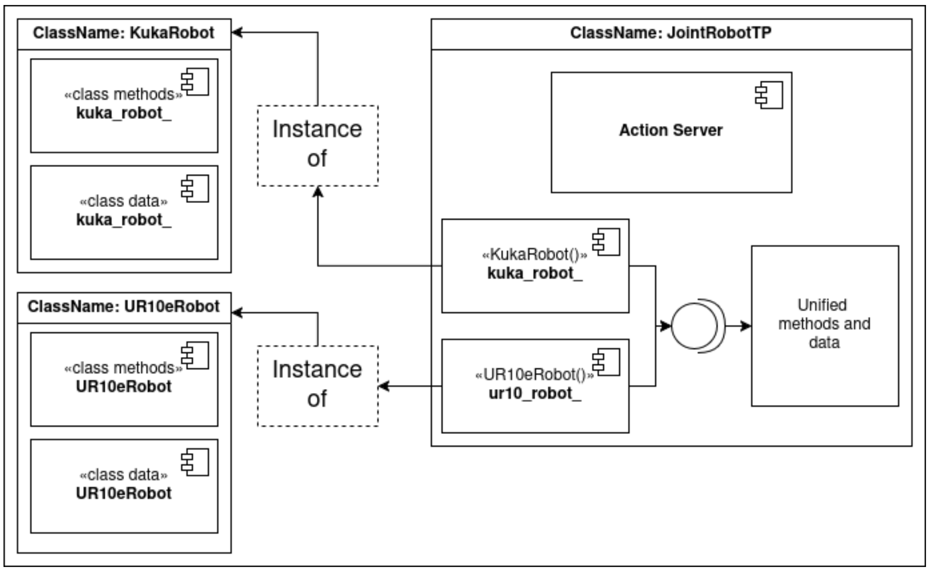

Figure 3.2: Simplified structure of the class JointRobotTP

The main objective was to use the existing classes developed for the single robots, and incorporate them in the unified architecture. The goals I wanted to pursue were the following:

1. Have an efficient way to send commands to the unified robot.
2. Use data structure that allowed me to add and remove tasks and configuration easily.
3. Keep a degree of separation between the robot representation and the control algorithm.

In the following section I will describe more in detail the structure of the *JointRobotTP* class.

3.2 JointRobotTP Class Implementation

The main part of the implementation of the class, are:

1. custom Action Server **RobotMoveTP**, implemented in the `uc1_robot_controllers_interfaces` package.
2. The two data structure used for the initial configuration reaching `initial_configurations_map_` and `TP_task_map_` to compute and store the matrix relative to each tasks.

3. The class used to compute each ”*step*” of the task priority algorithm.

3.2.1 Action server

The custom message definition for the Action server I used to send goals to the robot is as follows:
(<pkg> : uc1_robot_controllers_interfaces)

```
<pkg>/MoveRobotGoal         goal
        string         init_config_name
        ------               ------
        string               result
        ------               ------
        float64            linvel_norm
        float64            angvel_norm
```

and the custom message defined for the action goal is:

```
<pkg>/MoveRobotPoint   translation
                       float64    x
                       float64    y
                       float64    z
<pkg>/MoveRobotOrient  orientation
                       float64   roll
                       float64  pitch
                       float64    yaw
```

The goal is sent from the user as a *translation* and *rotation* relative to the initial position of the end-effector(in this case I am referring to the end-effector of the *UR10e* which is the end-effector of the unified robot). The action server than uses the information from the two robots to broadcast the *Goal* with respect to the *Kuka base link*. For each *Goal* recived by the robot, I can set a different *Initial configuration*. The field called `init\_config\_name` uses a map defined inside the class to set the robot in a specific configuration before starting the *Reaching loop*.

```
std::map<std::string, Eigen::VectorXd>   init_config_name
```

By using this data structure I can use the field of the `goal` message to directly select the desired initial configuration. The set of initial configuration is defined to have interesting starting position of the robot, to analyze different behavior of the robot in the experiment part.

3.2.1.1 Action Server process flow

Here is a flow chart to better explain the functionalities of the action server implemented in the *JointRobotTP* class:

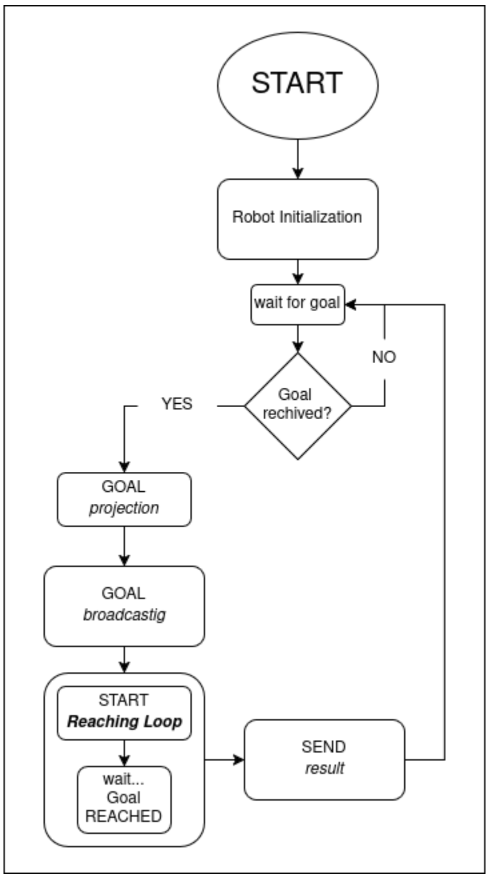

Figure 3.3: Action server flow chart

The method handling this loop is called:
`void execute(const std::shared_ptr<MoveRobotTP> goal_handle);`,
after reciveing the goal and accepting it, the requested goal is projected in the *Kuka robot base* and than broadcasted in the simulation. Lastly the method call for the *Reaching loop*, which I will explain in the next section.

3.2.2 Task Priority implementation

The control part is composed of two separate part: the data part, which is a member of the *JointRobotTP* class as for 3.2. This part is composed of a map `TP_task_map_` that contains all the matrixes relative to each task. A set of three function that are used to update the information inside the *Reaching loop*.

The definition of the data structure:

```
std::map<std::string, tp_task> TP_task_map_;
```

and `tp_task` is a `struct` with the following fields:

```
Eigen::MatrixXd    RefRate;
Eigen::MatrixXd   ActMatrix;
Eigen::MatrixXd  TskJacobian;
```

Secondly the set of functions for updating the information in `TP_task_map_` for each task have the structure:

Type	**Name**	**Args**
`void`	`Update_TRR_<task_name>`	`void`
`void`	`Update_AFunc_<task_name>`	`void`
`void`	`Update_TskJac_<task_name>`	`void`

The *Task Priority* control part is implemented trough a separate class. This class, called: `TPComputation`, has as private members two matrixes,

```
Eigen::MatrixXd    Q;
Eigen::MatrixXd  ydot;
```

these matrixes are the **projector** and the $\dot{y}$ of the last computed "*step*". Also in the initializaiton step i can define the values for the constants used in the computation of the pseudo-inverse matrix, these values will be described in 4.
As public members this class has methods to call for computing the *Task Priority algorithm*, task by task. The priority is imposed by the calling order in the *Reaching loop* code. These methods are structured as follows:

Type	**Name**	**Args**
`void`	`init_TPComputation`	`int Ndof,` `float lambda,` `float weigth,` `float treshold`
`void`	`computeTP_step`	`Eigen::MatrixXd ActFunct,` `Eigen::MatrixXd TskJacobian,` `Eigen::MatrixXd RefRate`
`Eigen::MatrixXd`	`getTP_ydot`	`void`

3.2.2.1 Reaching loop process flow

Finally I include a flow chart to inform about the process behind the *Reaching loop* implementation, for reference to the entire architecture 3.3.

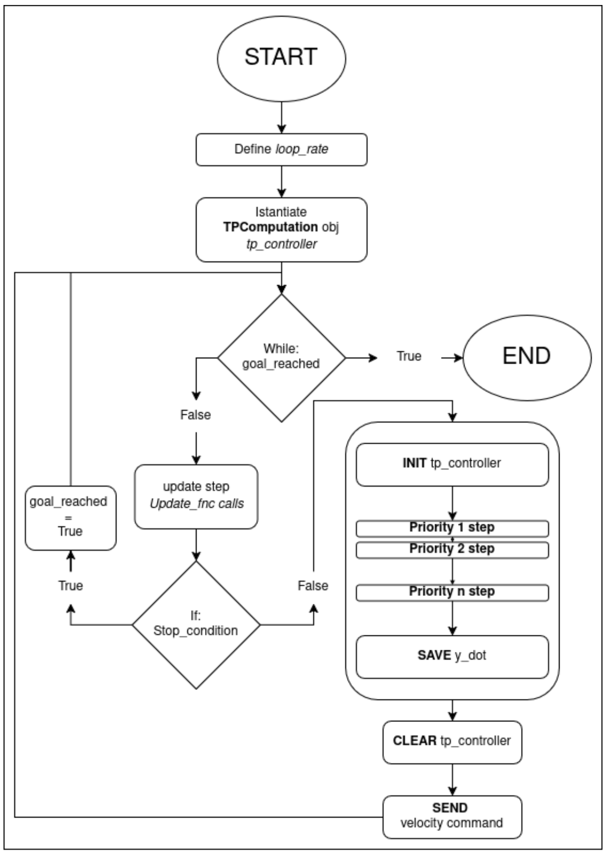

Figure 3.4: Reaching loop flow chart

The method used to implement this loop is one of the public member of the *JointRobotTP* class, namely:
`void RunCartesianReachingLoop(std::string goal_frame, bool reached_goal)`
In the *Update step*, all the function created that are relative to a *task* are called, starting with the *Reference Rate*, *Activation function* and lastly *Task Jacobian*. The second part, the *Stop condition* is checked. If the control is positive the loop is immediately stopped, and the result is

sent to the action server client. Next the *tp_controller* is initialized, the variables for the pseudo-inverse are initialized in this step. Than each "*step*" of the algorithm is computed, finishing with a "*null*" task, composed of two Identity matrix for *Activation function* and *Task Jacobian*, and with a zero vector for *Reference Rate*. The loop is repeated until the condition is met.

Chapter 4

Methodology

4.1 Reaching Loop Description

4.2 Goal broadcasting

4.3 Task Priority

4.3.1 Task Description

4.3.1.1 Joint Limits

4.3.1.2 Obstacle Avoidance

4.3.1.3 End Effector Target

Chapter 5

Experiments

5.1

Chapter 6

Conclusions

Write the conclusions here...

Appendix A

Extra

Write here...

References

ÅRZÉN, K.E. (1999). A simple event-based PID controller. *IFAC Proceedings Volumes*, **32**, 8687 – 8692, 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.