SAPIENZA
UNIVERSITÀ DI ROMA

Robot Control and Plan Execution for Automated Experiments in Industrial Laboratory

Department of Computer, Control, and Management Engineering: "Antonio Ruberti"

Master Science in Artificial Intelligence and Robotics

Candidate

Franci Rrapi
ID number 1660405

Thesis Advisor

Prof. Luca Iocchi

Co-Advisor

Dr. Fabio Zonfrilli

Academic Year 2021/2022

Thesis defended on 20 July 2022
in front of a Board of Examiners composed by:

Prof. Marco Schaerf (chairman)
Prof. Daniele Cono D'Elia
Prof. Giuseppe De Giacomo
Prof. Giorgio Grisetti
Prof. Luca Iocchi
Prof. Massimo Mecella
Prof. Giuseppe Santucci

Robot Control and Plan Execution for Automated Experiments in Industrial Laboratory
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: September 7, 2022

Author's email: rrapi.franci@gmail.com

Abstract

The aim of this Thesis is to apply Artificial Intelligence to Robots for autonomous industrial laboratory experiments purposes, with human intervention only in specific technical situations, such as breakdowns or unexpected behaviours. In recent years, robots have been designed to collaborate with humans to perform industrial manufacturing tasks. While humans have the flexibility and the intelligence to consider different approaches for solving a problem, to choose the best option among choices, and to give commands to robots to perform a task, robots are able to be more precise and more consistent in both, performing repetitive and manual tasks and also for completing more dangerous operations, avoiding, this way, humans doing it. For this reasons, the interaction between Humans and Collaborative Robots (called also "Cobots", which are robots that share the workplace with humans without barriers and without exposing them to risks) in sharing or alternating some tasks, has become fundamental in this work. Various techniques were used to achieve this goal, such as Robot Motion Control and Computer Vision, where Robot Motion Control is used for asking the robot to reach an exact position with high precision orientation, e.g. to grasp an object or to close a drawer, and Computer Vision allows the robot to have a perception of the world through RGB camera, which transmits many important information to the robot, like the position of an object. The AI project development culminated in a complex logic application which enables the robot to take autonomous and intelligent decisions when completing laboratory testing activities.

Acknowledgments

Firstly, I would like to thank Professor Luca Iocchi, who has given me the opportunity to undertake this thesis work and always provided my colleague, Antonio Lomuscio, and I advice to achieve our project goals. I would also like to thank my supervisor, Dr. Fabio Zonfrilli, who from the first day welcomed us into the P&G family and always supported our work with critical suggestions and ideas. I would like to give my special thanks to my colleague and friend, Antonio Lomuscio, who has encouraged and collaborated with me since our earliest days of university. But most importantly I would like to thank my brother, Marius Rrapi, who has always been an example and offered invaluable advice not only in my studies but also, and above all, in life. I am grateful and humbled by his and my parents constant support as I have pursued my goals.

Contents

Chapter 1

Introduction

This thesis work was born from the collaboration between P&G Services Company in Belgium and University of Rome "La Sapienza" for the European project "AIPlan4EU"[1][2] that aims the development of new technological features for the companies through the AI Planning.

The aim of this work is the application and realization in the real world of the previous thesis developed by Mario Fiorino[3] and has been realized with the collaboration of my colleague Antonio Lomuscio.

Mario's work was focused on the creation of a nominal plan, from which the robot can recover each time there is a failure, in a simulated environment. So our purpose was to extend this plan and apply it in the real environment, in this case in the P&G laboratory with the coexistence of the human beings.

AIPlan4EU project was founded by the European Commission and launched in January 2021 with a duration of 3 years; it is composed by 16 European partners from different industries and with various expertise (research groups and companies).

AIPlan4EU aims to bring the most advanced planning technology Europe has to offer to companies through the AI4EU Platform[2], demonstrating how this technology can be adopted in different scenarios, boosting the use of planning technologies even in non-technical businesses and encouraging the research in AI planning in a modern, application-oriented fashion.

To do so, use-cases from diverse application areas has been considered to drive the design and the development of the framework, and include several available planning systems to solve practical problems.

For this work, the AI Planning use-cases were provided by Procter & Gamble, more exactly by Fabio Zonfrilli, who holds the role of R&D Principal Scientist Robotics and Automation, within the context of industrial automated experiments. More precisely that consists on the executive procedure for the analysis and test of the laundry pouch (soluble capsules that contains liquid and powder) performed through a robot composed by a robotic arm and a gripper, in a dedicated laboratory, using different measuring instruments. Furthermore, it was asked that the AI Planning framework included with it also the human component, in such a way to have a mixed system human-robot for the execution of the product test experiments.

My thesis work started by adding to the already existing simulator the Kinematics tools provided by MoveIt!, that allowed to reach positions or objects during the experiments with the Inverse Kinematics. Working at first on the simulator, to then pass on the real robot in real scenario, allows to reduce the high costs of design, experimentation and debug code, which would entail working directly on the hardware in real world situation.

For guaranteeing portability and self-sufficiency in most computing environments, was used a Docker image (a state-of -the-art containerization technology) that contains custom ROS (Robot Operating System: framework robot application), packages for the UR5e Robot with a 2F-85 Gripper, Gazebo (3D robotics simulator for ROS system) within is reconstructed the laboratory working environment in its fundamental aspects and MoveIt! (open-source robotic manipulation platform) that incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation.

The second phase of my thesis work consisted in the setup and configuration of the real robot in P&G laboratory. For doing this was created another Docker image (ROS Melodic and Ubuntu 18.04) from scratch with the UR and Robotiq drivers installed and all required libraries useful for the connection and interaction between robot and ROS pc.

The third phase of this thesis work consisted in the implementation of the actions and fluents, that will be used forward by the plan for interfacing with the real robot, measurement instruments and the surrounding environment. The interaction with the Robot was made with the Universal Robot ROS Driver and the Universal Robot package that provides all the robot URDFs. Have been also implemented actions and fluents concerning the interface with various instruments, such as the camera (used for the perception and detection task made through the YOLOv5 Neural Network that has been trained for the pouch detection), and also the Mettler Toledo scale and Mark10 tightness (for the interaction and useful test scientific data extraction).

The fourth phase of my thesis work consisted in the conception, implementation and verification test of AI plans, formulated through Petri Net Plans (PNP) and Plan Execution Interface (PLEXI) frameworks. Regarding formulation and implementation of a complex plan, these technologies guarantee modularity, high expressiveness (accessible also for an users are not experts of the robotic system) and flexibility in case of updates and structural changes. One of the most interesting parts of this phase was to work on recovery procedures: robotic systems are not yet "perfect" machines, in the real world failures or unforeseen situations can happen.

To deal with this type of problem, the strategy used was that of: at first, define a recovery plan thanks to which the robot autonomously try to resumes the plan from where it left off; then if this should not be sufficient, require for human intervention (Human-in-the-loop concept). An Application was used to optimize human-robot interaction. This document, in addition to presenting the thesis work, also aim to be a basis user guide for future projects and experiments.

Thanks to the work carried out during the thesis, these problems have been solved and the robot is now able to carry out all the pouch tests independently and when an anomalous event occurs (such as the loss of the pouch, the non-functioning or loss connection of some instrument), tries to solve the problem by itself or asks

for human intervention, and then resumes normal execution of the plan from where it stopped.

Keywords: AIPlan4EU, AI Planning, Universal Robot 5, Robotiq Gripper, UR5e, Docker, ROS, ROS Action, Gazebo, Petri Net Plans, Plan Execution Interface, MODIM, Recovery Procedure, Human–robot interaction, Human-in-the-loop.

1.1 Executive summary

In Chapter 2 an introduction of the State-of-the-Art will be introduced, such as AI planning, Previous Practice used, Robot Kinematics, Robot Control, Motion Planning, Cobots, Human-Robot Interaction, Robot Perception and Object Detection.

In Chapter 3 the software technology and tools used for sending commands and control the robot, perceiving the environment and tracking the system behaviour: Docker, ROS, Gazebo, MoveIt!, RViz, Transforms, Universal Robotic Description Format (URDF), Petri Net Plans (PNP) and Plan Execution Interface (PLEXI) frameworks with which the plans were implemented, OpenCV, ArUco Markers, Roboflow, YOLOv5, Tesseract OCR and PySimpleGUI is described. Their advantages and disadvantages, how and why they have been used to develop this thesis.

In Chapter 4 the robot and devices used for performing the experiment product tests: UR5e robotic arm, Robotiq 2F-85 gripper, Microsoft LifeCam Cinema, Mettler Toledo scale and Mark-10 Motorized Test Stand is described. Their principal features, how they works and why have been used in this thesis.

In Chapter 5 the aim and objectives that we wanted to reach with this thesis work are described. Also a little description of the main problems encountered during this work and their solution.

In Chapter 6 the Solution Design used for solving the problem. In particular is shown the Laboratory configuration, The Diagram of the *Plan* flow used to solve this tasks, the Action and Fluents implemented that composes the plan, the HRI Recovery procedures used to resume from a certain unexpected event and the advantages of this solution.

In Chapter 7 the implementation details of the modules realized by me and with my colleague collaboration will be shown and described, such as the Camera calibration, connection and interface between Gripper and PC-ROS, the position of the pouch with respect to the robot, interpolation over gripper grasping angles, the MoveIt! complete robot configuration for IK and the collision avoidance solution to prevent the robot collide with objects in the scene.

In Chapter 8 a set of pictures showing the results obtained. Is possible to observe the robot while opening the drawer, grasping some pouches, or going to Mark-10 position and waiting for test to be completed.

In Chapter 9 is done a discussion about this work and which could be the future works.

In Chapter A will be described some technical aspects used in this thesis, such as Drivers and packages used to make the robot work properly.

In Chapter B more details about the tools and software are described.

Chapter 2

Theoretical Starting Point

2.1 Previous practice

Here I will gonna explain how the experiments were done before, without the use of any AI planning or automated process, and also the limits of this approach.

Talking with our P&G supervisor Dr. Zonfrilli and other specialized laboratory employees, we have come to know that the testing process of the laundry pouches before our work was made in a "manual" way. This approach requires that a person, in a dedicated laboratory, execute all the necessary experiments for every pouch and all the results, obtained from different instrument measurements, has to be inserted in a dataset manually.

So we noticed that in this way, the process is very slow and takes too much time. It always and necessarily requires the presence of at least one human who performs all this, and this also entails a slowdown in the productivity of the employee who, if these experiments were carried out semi-autonomously by an intelligent agent such as a robot, could dedicate this time to relax, optimize their productivity or carry out other company jobs in parallel.

Also there is another factor that is not to be underestimated, that the employee is an human. This not only affects the speed of execution of the tests, but also the error probability due to fatigue, distraction or other. Probability that is very less or almost zero in a robot, which can repeat actions with very high precision for a long period of time without stopping.

2.2 AI Planning

AI Planning[7] consists in reasoning on a predictive model of a system being controlled (and possibly of its environment) and deciding what to do in order to achieve a desired objective. So, it can be used to identify a course of actions that achieves a certain objective.

Automated Planning and Scheduling is a central research area in AI that has been studied since the inception of the field and where European research has been making strong contributions over decades. Planning is a decision-making technology that consists in reasoning on a predictive model of a system being controlled and deciding how and when to act in order to achieve a desired objective.

The planning problem in Artificial Intelligence is about the decision making performed by intelligent creatures like robots, humans, or computer programs when trying to achieve some goal. It involves choosing a sequence of actions that will (with a high likelihood) transform the state of the world, step by step, so that it will satisfy the goal. The world is typically viewed to consist of atomic facts (state variables), and actions make some facts true and some facts false. Most works on planning use a state-transition system model (we often just write transition system), in which the world/system consists of a (finite or infinite) number of states, and actions/transitions change the current state to a next state.

The simplest language used for formalizing actions is the STRIPS (Stanford Research Institute Problem Solver) language. In STRIPS, the state variables have the domain 0,1 (equivalently FALSE, TRUE), and an action consists of three sets of state variables, the PRECONDITION, the $ADD = \{a_1, a_2, ..., a_n\}$ list, and the $DELETE = \{d_1, d_2, ..., d_m\}$ list (it is assumed that ADD and DELETE don't intersect). Petri nets are a model of state transition systems in which several transitions may take place simultaneously in the sense that they are independent. In state-transition nets, each place can hold 0 or more tokens. Essentially, a place is a state variable with the set of natural numbers as its domain.

In simply words, AI Planning is a branch of Artificial Intelligence that concerns the computational formulation of a plan made by a set of actions performed by an intelligent agent (in this case a robot) that interacts with environment (world) starting from a initial state until a desired goal is reached, that is a final state in which specific conditions are met. For interacting with the world the agent has to perform some actions that for being applied requires some verified preconditions in a certain state.

AI planning problem can be formulated as follows:

- the initial state s_0 of the world
- the goal g: the desired final state we want to achieve
- the state transition system: applies changes to the world with actions

The state transition system mentioned above is a 3-tuple composed of:

- Set of states: $S = \{s_1, ..., s_n\}$[1].
- Set of actions: $A = \{a_1, ..., a_n\}$. Each action is defined in the form of the unit of three lists:
  pre list - predicates that if verified allows the applicability of the action;
  add list - predicates that become true in the current state by the action application;
  del list - predicates become false in current state as a result of the applied action.
- The state transition function: $\delta : (S \times A^* \to S)$

All this elements contribute to the formulation of a plan: given in input an initial state of the world, goal, and an action model; a plan is a sequence of actions that drives the system from the initial state to the goal.

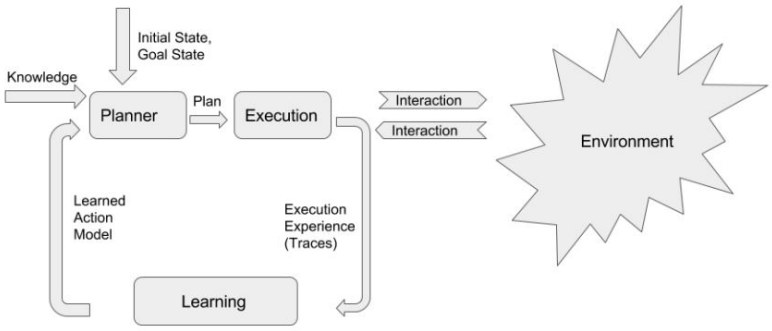


Figure 2.1. Learning planning action model diagram[13]

2.3 Robot Kinematics

Robot kinematics[9] applies geometry to the study of the movement of multi-degree of freedom kinematic chains that form the structure of robotic systems.The emphasis on geometry means that the links of the robot are modeled as rigid bodies and its joints are assumed to provide pure rotation or translation.

Robot kinematics studies the relationship between the dimensions and connectivity of kinematic chains and the position, velocity and acceleration of each of the links in the robotic system, in order to plan and control movement and to compute actuator forces and torques.

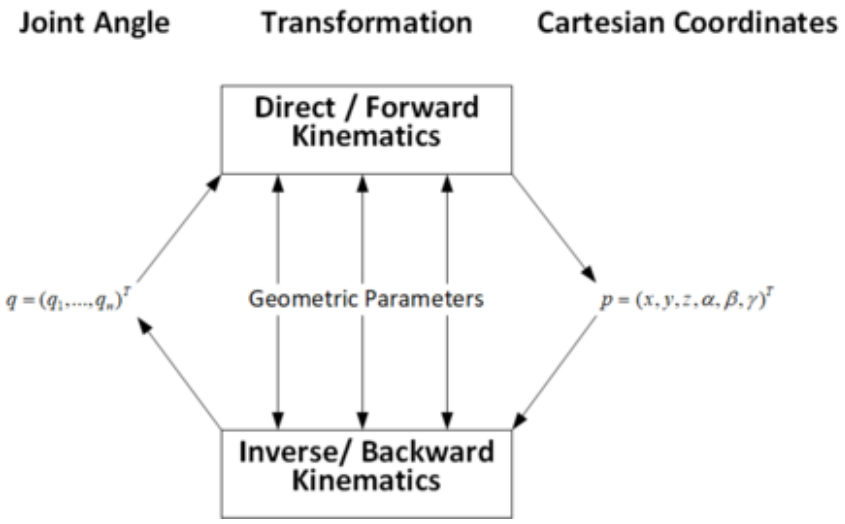


Figure 2.2. Transformation relations for a robot kinematic

The relationship between mass and inertia properties, motion, and the associated forces and torques is studied as part of robot dynamics.

A fundamental tool in robot kinematics is the kinematics equations of the kinematic chains that form the robot. These non-linear equations are used to map the joint parameters to the configuration of the robot system. Forward kinematics uses the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters.The reverse process that computes the joint parameters that achieve a specified position of the end-effector is known as inverse kinematics. Determining the movement of a robot so that its end-effectors move from an initial configuration to a desired configuration is known as motion planning. Inverse kinematics transforms the motion plan into joint actuator trajectories for the robot.

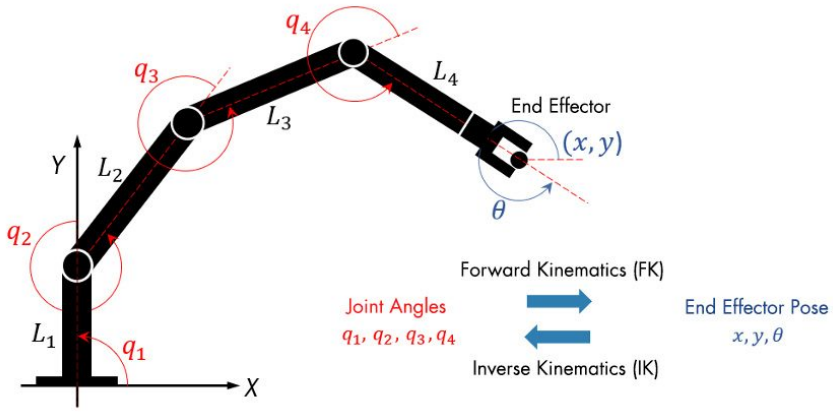


Figure 2.3. Kinematic chain of a robot manipulator arm.

The dimensions of the robot and its kinematics equations define the volume of space reachable by the robot, known as its workspace.

[1]Note: For the representation of a state is necessary to provide the planner with a model of world on which it operates

2.4 Motion Planning and Robot Control

The **Motion Planning**[10] (or **Path Planning**) concerns the problem of finding a sequence of valid configurations in such a way that a certain object moves from a start position and reaches a desired destination (or goal position). The term is used in computational geometry, computer animation, robotics and computer games.

Let's consider for example the navigation of a mobile robot inside a building to a distant waypoint. It should execute this task while avoiding walls and not falling down stairs. A motion planning algorithm would take a description of these tasks as input, and produce the speed and turning commands sent to the robot's wheels. Motion planning algorithms might address robots with a larger number of joints (e.g., industrial manipulators), more complex tasks (e.g. manipulation of objects), different constraints (e.g., a car that can only drive forward), and uncertainty (e.g. imperfect models of the environment or robot).

Motion planning has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game, architectural design, robotic surgery, and the study of biological molecules.

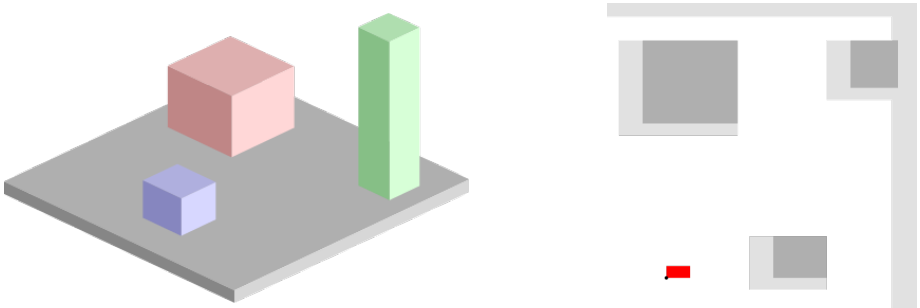
The robot does not know which path or trajectory to execute in such a way to reach a desired goal pose that we give as input to **MoveIt**. MoveIt is designed to work with many different types of planners and the one that we worked with is Open Motion Planning Library (OMPL). **OMPL** is an open-source motion planning library that primarily implements randomized motion planners. MoveIt integrates directly with OMPL and uses the motion planners from that library as its primary/default set of planners. The planners in OMPL are abstract; i.e. OMPL has no concept of a robot. Instead, MoveIt configures OMPL and provides the back-end for OMPL to work with problems in Robotics.

Among the set of paths/trajectories that robot can perform, the planner should select the best one, that depends on the optimization wanted, such as:

- *Time*: is selected the path that reaches the goal with the shortest time.
- *Distance*: is selected the path that reaches the goal with the shortest distance.
- *Other*: some other complex constraints or conditions.

A configuration describes the pose of the robot, and the configuration space C is the set of all possible configurations. For example:

- If the robot is a single point (zero-sized) translating in a 2-dimensional plane (the workspace), C is a plane, and a configuration can be represented using two parameters (x, y).
- If the robot is a $2D$ shape that can translate and rotate, the workspace is still 2-dimensional. However, C is the special Euclidean group $\mathbf{SE}(2) = \mathbf{R}^2 \times \mathbf{SO}(2)$ (where $\mathbf{SE}(2)$ is the special orthogonal group of $2D$ rotations), and a configuration can be represented using 3 parameters (x, y, θ).



(a) Workspace. **(b)** Configuration space.

Figure 2.4. Configuration space (b) in a workspace (a) for a rectangular translating robot (pictured red). White = C_{free}, gray = C_{obs}, where dark gray = the objects, light gray = configurations where the robot would touch an object or leave the workspace.

- If the robot is a solid $3D$ shape that can translate and rotate, the workspace is 3-dimensional, but C is the special Euclidean group $\mathbf{SE}(3) = \mathbf{R}^3 \times \mathbf{SO}(3)$, and a configuration requires 6 parameters: (x, y, z) for translation, and Euler angles (α, β, γ) for orientation.
- If the robot is a fixed-base manipulator with N revolute joints (and no closed-loops), C is N-dimensional.

The set of configurations that avoids collision with obstacles is called the free space C_{free}. The complement of C_{free} in C is called the obstacle or forbidden region. Often, it is prohibitively difficult to explicitly compute the shape of C_{free}. However, testing whether a given configuration is in C_{free} is efficient. First, forward kinematics determine the position of the robot's geometry, and collision detection tests if the robot's geometry collides with the environment's geometry.

Target space is a subspace of free space which denotes where we want the robot to move to. In global motion planning, target space is observable by the robot's sensors. However, in local motion planning, the robot cannot observe the target space in some states. To solve this problem, the robot goes through several virtual target spaces, each of which is located within the observable area (around the robot). A virtual target space is called a sub-goal.

Obstacle space is a space that the robot can not move to. Obstacle space is not opposite of free space.

A basic motion planning problem is to compute a continuous path that connects a start configuration S and a goal configuration G, while avoiding collision with known obstacles. The robot and obstacle geometry is described in a 2D or 3D workspace, while the motion is represented as a path in (possibly higher-dimensional) configuration space.

Robotic control[11] is the system that contributes to the movement of robots. This involves the mechanical aspects and programmable systems that makes it possible to control robots. Robotics could be controlled in various ways, which includes using manual control, wireless control, semi-autonomous (which is a mix of fully automatic and wireless control), and fully autonomous (which is when it uses artificial intelligence to move on its own, but there could be options to make it manually controlled). In the present day, as technological advancements progress, robots and their methods of control continue to develop and advance.

In recent years the use of robots has increased exponentially until it is also present in our daily lives. The areas where they are most used are those in the medical and surgical fields, in guiding people in crowded public places (such as train stations, airports) through human-robot interaction (HRI), in space exploration and in military field.

In the medical field, robots are used to make precise movements that are humanly difficult. Robotic surgery involves the use of less-invasive surgical methods, which are

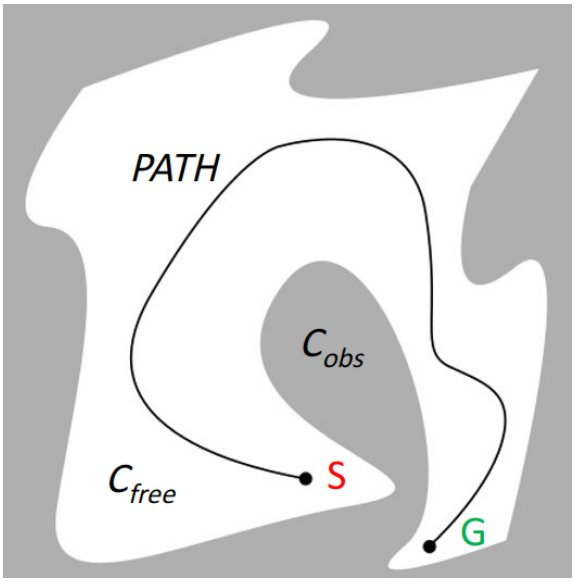


Figure 2.5. Example of a valid path.

"procedures performed through tiny incisions".

Space missions involve sending robots into space in the goal of discovering more of the unknown. The robots used in space exploration have been controlled semi-autonomously. The robots that are sent to space have the ability to maneuver itself, and are self-sustaining.

2.5 Cobots

Cobots[36] are Collaborative robots that are cost-effective, safe, and flexible to deploy. Collaborative robots - or cobots - are making automation easier than ever, even for small and mid-sized companies around the world. Cobots are designed to share a workspace with humans, making automation easier than ever before for businesses of all sizes. All of these benefits have made our cobots a game-changer for a wide variety of applications. Cobots are an ideal productivity tool for almost any manufacturer because they help everyone in the company achieve performance objectives.

A cobot, or collaborative robot, is a robot intended for direct human robot interaction within a shared space, or where humans and robots are in close proximity. Cobot applications contrast with traditional industrial robot applications in which robots are isolated from human contact. Cobot safety may rely on lightweight construction materials, rounded edges, and inherent limitation of speed and force, or on sensors and software that ensure safe behavior.

The International Federation of Robotics (IFR), a global industry association of robot manufacturers and national robot associations, recognizes two main groups of robots: industrial robots used in automation in an industrial environment and service robots for domestic and professional use. Service robots could be considered to be cobots as they are intended to work alongside humans. Industrial robots have traditionally worked separately from humans behind fences or other protective barriers, but cobots remove that separation.

Cobots[37] can have many uses, from information robots in public spaces (an example of service robots), logistics robots that transport materials within a building, to industrial robots that help automate unergonomic tasks such as helping people moving heavy parts, or machine feeding or assembly operations.

The IFR defines four levels of collaboration between industrial robots and human workers:

- *Coexistence*: Human and robot work alongside each other without a fence, but with no shared workspace.
- *Sequential Collaboration*: Human and robot are active in shared workspace but their motions are sequential; they do not work on a part at the same time.
- *Cooperation*: Robot and human work on the same part at the same time, with both in motion.
- *Responsive Collaboration*: The robot responds in real-time to movement of the human worker.

2.6 Human-Robot Interaction

Human-Robot Interaction[14] (HRI) is the study of interactions between humans and robots. Robots are artificial agents with capacities of perception and action in the physical world often referred by researchers as workspace. Their use has been generalized in factories but nowadays they tend to be found in the most technologically advanced societies in such critical domains as search and rescue, military battle, mine and bomb detection, scientific exploration, law enforcement, entertainment and hospital care.

These new domains of applications imply a closer interaction with the user. The concept of closeness is to be taken in its full meaning, robots and humans share the workspace but also share goals in terms of task achievement (**Figure 2.6**).

With the advance in AI, the research is focusing on one part towards the safest physical interaction but also on a socially correct interaction, dependent on cultural criteria. The goal is to build an intuitive, and easy communication with the robot through speech, gestures, and facial expressions.

Figure 2.6. Robotic arm interacting with human in the same workspace.

Industrial robots have been implemented to collaborate with humans to perform industrial manufacturing tasks. While humans have the flexibility and the intelligence to consider different approaches to solve the problem, choose the best option among all choices, and then command robots to perform assigned tasks, robots are able to be more precise and more consistent in performing repetitive and dangerous work.

Methods for perceiving humans in the environment are based on sensor information. Most methods intend to build a 3D model through vision of the environment. The proprioception sensors permit the robot to have information over its own state. This information is relative to a reference. A speech recognition system is used to interpret human desires or commands. By combining the information inferred by proprioception, sensor and speech the human position and state (standing, seated). In this matter, natural-language processing is concerned with

the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural-language data.

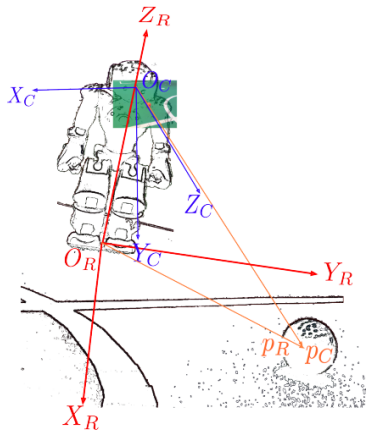
2.7 Robot Perception and Object Detection

Robots are increasingly being used in specific environments to perform concrete complex tasks. Such robot agents need to perform a set of computationally intensive functions, in order to be able to perceive, to reason about, and to act in their surroundings. The robot's actuation, whether it is to manipulate objects or just to navigate throughout its environment, is based on the information provided by its sensors.

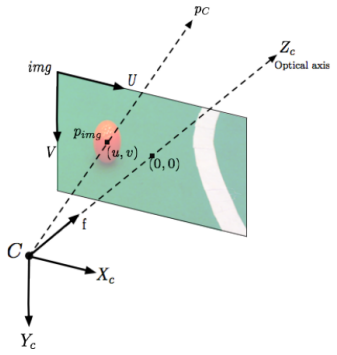
As mentioned in the previous section, robots can use information from various sensors for interacting with humans, and one of the richest and most complex sensory information sources is the visual information as captured by cameras. The perception of the world can be used not only for HRI, e.g. the interaction with a standing human in front of the robot and capturing his gestures, but also for object recognition in the world space.

Many of the modern robots are equipped with cameras to observe the world and detect the objects[15] relevant to accomplish the specific tasks. But the problem is that the image that the agent receives from the camera is not directly usable, so it has to be pre-processed and from that extract and transform the information needed.

In this case, an image stores the information collected from the camera and an image pixel $p_{img}(u, v)$ is the intensity of light or the color stored in the image coordinates (u, v) in the image frame. The reference frame of the camera C represents points in space with respect to the center of the camera. The image frame is a plane perpendicular to the axis Z_C , at a distance f far away from the center of the camera. Every point p_C in the camera frame has an unique projection p_{img} in the image plane, located at the intersection of the plane with the line linking p_C with the center of the camera. As we need to know the exact position of the object with respect to the robot, Instead of using the reference axes of the camera C, we use the reference axes of the robot R (**Figure 2.7a**). In this way we have the position (x, y, z) of the object wrt robot base frame.



(a) Robot and camera frames. Projection of the Hypothesis p_C (or p_R with respect the robot frame) in the image.



(b) *Pin-hole* camera model. The camera frame origin C is at the center of the camera. The line connecting the point P_C and the camera center cross the image plane in the point p_{img}, which is its projection in the image.

Figure 2.7. Object detection from robot and camera perspective.

Chapter 3

Tools and Software

3.1 Docker

Docker[16] is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

It provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Each Docker container starts with a Dockerfile. This text file provides a set of instructions to build a Docker image, including the operating system, languages, environmental variables, file locations, network ports, and any other components it needs to run.

A Docker image[18] is a portable, read-only, executable file containing the instructions for creating a container and the specifications for which software components the container will run and how. It has everything needed to run a containerized application, including application code, config files, tools, environment variables, libraries and runtimes. Docker's run utility is the command that launches a container. Each container is an instance of an image, and multiple instances of the same image can be run simultaneously.

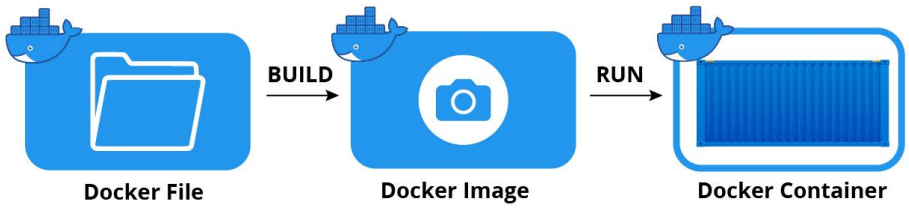


Figure 3.1. Docker container creation.

3.1.1 Docker container vs. Docker image

A Docker[17] image has everything needed to run a containerized application, including code, configuration files, environment variables, libraries and runtimes.

A **Docker container** is a virtualized runtime environment used in application development. It is used to create, run and deploy applications that are isolated from the underlying hardware. A Docker container can use one machine, share its kernel and virtualize the OS to run more isolated processes. As a result, Docker containers are lightweight.

A **Docker image** is a record of a Docker container at a specific point in time. Docker images are also immutable. While they can't be changed, they can be duplicated, shared or deleted. The feature is useful for testing new software or configurations because whatever happens, the image remains unchanged.

Containers need a runnable image to exist. Containers are dependent on images, because they are used to construct runtime environments and are needed to run an application.

3.2 ROS

ROS[19] is widely used in robotics companies, universities and robotics research institutes for designing, building, and simulating a robot model and interfacing it into real hardware.

Here is a definition of ROS from the original paper[21]: *ROS, an open-source robot operating system. ROS is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a heterogeneous compute cluster.*

The Robot Operating System (ROS) is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS is also a global open-source community of engineers, developers and hobbyists who contribute to making robots better, more accessible and available to everyone. The primary goal[16] of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed.

It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS[20] runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. ROS is not a realtime framework, though it is possible to integrate ROS with realtime code.

The Robot Operating System (ROS) [18] is not an actual operating system, but a framework and set of tools that provide functionality of an operating system on a heterogeneous computer cluster. Its usefulness is not limited to robots, but the majority of tools provided are focused on working with peripheral hardware.

ROS[22] provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines, tools for testing and visualization, and much more.

The key feature of ROS is the way the software is run and the way it communicates, allowing you to design complex software without knowing how certain hardware works. ROS provides a way to connect a network of processes (nodes) with a central hub. Nodes can be run on multiple devices, and they connect to that hub in various ways.

The main ways of creating the network are providing requestable services, or defining publisher/subscriber connections with other nodes. Both methods communicate via specified message types. Some types are provided by the core packages, but message types can be defined by individual packages.

3.2.1 Packages

ROS[23] is split up in more than 2000 packages, each package providing specialized functionality. The number of tools connected to the framework are probably its biggest power. Refering to [23], we have that:

Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages is to provide this useful functionality in an easy-to-consume manner so that software can be easily reused. In general, ROS packages follow a "Goldilocks" principle: enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.

Some of the important files/directories inside Packages are:

1. *Nodes*: A node is a process that performs computation.
2. *CMakeLists.txt*: It is the input to the CMake build system for building software packages.
3. *Package.xml*: It defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.
4. *yaml* files: To run a rosnode you may require a lot of parameters e.g, Kp,Ki,Kd parameters in PID control. We can configure these using *YAML* files.
5. *launch* files: To run multiple nodes at once in ROS we use launch files.

3.2.2 Catkin workspace

Any code that will be written should be in the form of packages. And the packages should be inside workspace.

A *catkin workspace*[24], is a folder where you modify, build, and install catkin packages. It can contain up to four different spaces which each serve a different role in the software development process.

1. The *source space* contains the source code of catkin packages. This is where you can extract/checkout/clone source code for the packages you want to build. Each folder within the source space contains one or more catkin packages.
2. The *build space* is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here.
3. The *development space* (or devel space) is where built targets are placed prior to being installed. The way targets are organized in the devel space is the same as their layout when they are installed. This provides a useful testing and development environment which does not require invoking the installation step.
4. Once targets are built, they can be installed into the *install space* by invoking the install target, usually with make install.

3.2.3 Master

When we run our rosnodes, they perform computations and obtain results. But they may require results from other nodes in order to perform some other functions. Hence we need a mechanism that can help us transfer data from one node to other. One of the first thing we need to do is to setup a ROS Master.

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

The transfer of data takes place via topics. If you want to send your data, you publish it to topics and whomever needs it, can subscribe to it using publishers. This is helpful in writing code and even more helpful when we are using ROS bags. ROS bags are helpful when we want to record some data, so we can play it later and replicate a behavior if we want to.

Services are another form of communication. They are used for remote procedural calls. i.e, one program can request a service from a program located in another computer. The data that is transferred is in the form of messages which are especially defined data types

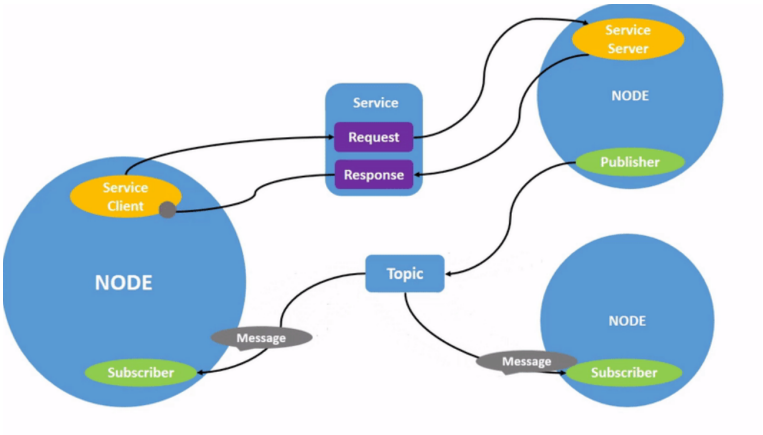


Figure 3.2. Simple scheme of node graph architecture in ROS.

used in ROS. The last type of communication we are going to look at are actions, They are similar to services but the long term goals can be preempted i.e, can be requested to change.

3.2.4 Tools

ROS has useful tools to visualize your robot. Once we have all the code ready and running, we need to test our code so that we can make changes if necessary. Doing this on a real robot will be costly and may lead to wastage of time in setting up robot every time. Hence we use robotic simulations for that. The most popular simulator to work with ROS is **Gazebo**. It has a good community support, it is open source and it is easier to deploy robots on it.

The robot will be setup with different sensors and actuators, and while running these sensors, we may need to visualize their data. We use **RViz**[1] for this purpose. Rviz is powerful 3D visualisation tool for ROS that allows you to view the robot's sensors and internal state. It is one of the most popular tools for visualization. It takes in a topic as input and visualizes that based on the message type being published. It lets us see the environment from the perspective of the robot. Rviz uses the URDF file to load our robot in the tool. It can also display live representations of sensor values coming over ROS Topics including camera data, point cloud data, infrared distance measurements, sonar data, and more. Using Rviz we can check our path planning or collision checking algorithms by inserting objects, trace the motion of any link of the robot, use interactive markers to move the robot and do lot things.

Another tool that can be used in ROS to control and manipulate a robot not only in simulation, but also in the real world, is **MoveIt!**[8] (MoveIt Motion Planning Framework), an easy-to-use open-source robotic manipulation platform that incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation.

3.2.5 Transforms

All the Transforms of the robot are tracked in ROS. We no more need to find the Forward kinematics equations to find the position of the end-effector. We just have to provide the robot description in URDF format. Thats it tf[2] or tf2 module keeps track of all the

[1] `http://wiki.ros.org/rviz`
[2] `http://wiki.ros.org/tf`

transforms of the robot from base to end-effector. Almost everything in robotics is concerned with where things are - either relative to the robot itself or relative to other things. Whenever a robot wants to interact with the real world, it will need to know where the things it wants to interact with are.

3.2.6 Universal Robotic Description Format (URDF)

URDF[3] is an XML file format used in ROS to describe all elements of a robot and their properties. URDF will have the information about the robot's physical construction, geometry, dynamics, etc. In short URDF will tell ROS how the links and joints are connected, type of joints, the dimensions of links, mass and inertia of the links, etc. So to start building your robots in ROS first we need to know how to represent the robot in URDF file.

3.3 Petri Net Plans (PNP)

Petri Net Plans (PNP)[29] is a formalism for high level description of complex plans (i.e., set of actions interacting in a complex way), based on the graphical system description Petri nets. For better understanding the importance of PNPs, we report below the words of its creators:

PNPs are more expressive than Finite State Machines (FSM) and allow for automatic plan analysis, which can provide formal guarantees on the performance of the plans. Execution of PNPs is extremely efficient and allows the design of real-time pro-active and reactive behaviors. It is useful to program cognitive agents (such as robots, videogame agents, multi-robot/multi-agent systems, etc.). PNPs are inspired to languages for reasoning about actions, yet they are more expressive than most of them, offering a full fledged set of operators for dealing with non-instantaneous actions, sensing actions, action failures, concurrent actions and cooperation in a multi agent context. PNPs include also choice operators used for non-deterministic execution and for learning in the plan space through a Reinforcement Learning algorithm.

Petri nets graphically represent the structure of a distributed system as a directed, weighted and bipartite graph. As such, a Petri net has two types of nodes connected by directed weighted arcs. The first type is called place and may contain zero or more tokens (where the number of tokens in each place denotes the state of the system). The other type of nodes, called transitions, represent the events modeled by the system. Transitions can consume or produce tokens from places according to the rules defining the dynamic behavior of the Petri net (i.e. the firing rule).

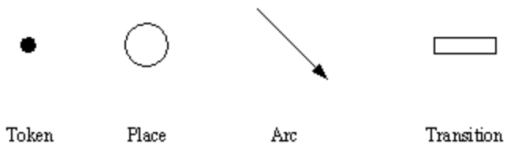


Figure 3.3. Petri nets basic elements.

A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition. An enabled transition may

[3] `http://wiki.ros.org/urdf`

fire at any time. When fired, the tokens in the input places are moved to output places, according to arc weights and place capacities. This results in a new marking of the net, a state description of all places.

A PNP[30] is a Petri Net structure augmented with a set of goal marking, so it is a 6-tuple $\langle P, T, F, W, M_0, G \rangle$, where:

- $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places. Places p_i represent the execution phases of actions; each action α is described by a place corresponding to its initiation, one corresponding to its execution, and one to its termination.
- $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions. Transitions t_i represent events and are grouped in different categories: action starting transitions, action terminating transitions, action interrupts and control transitions. Transitions may be labeled with conditions that control their firing.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.
- $W : F \to \{1\}$ is a weight function for edges. In PNPs $w(f_s, f_d) = 1$ for each pair $fs, fd \in F$.
- $M_0 : P \to \{0, 1\}$ is the initial marking, denoting the initial state of the structure.
- $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$
- $G : P \to \{0, 1\}$ is the goal marking.

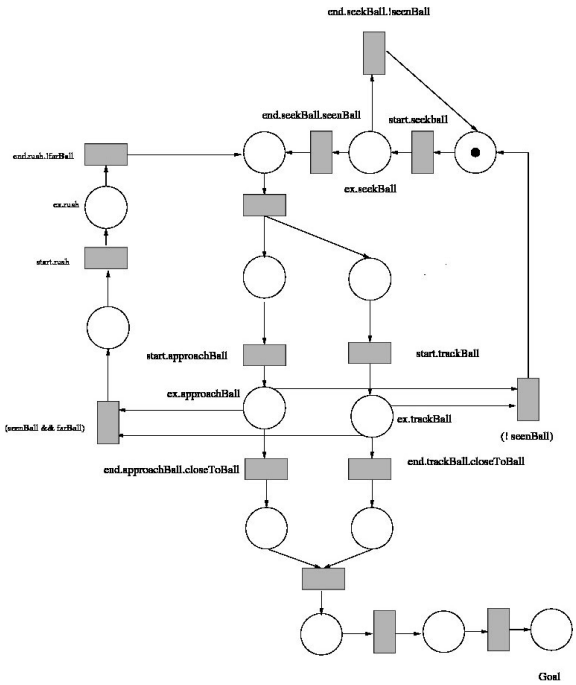


Figure 3.4. Example of PNP execution diagram.

PNP is formed by various components, such as PNPgen and PNPros that will be explained bellow.

3.3.1 PNPgen

Generation of PNPs is typically used in conjunction with a planner, with the objective of representing and executing the plan generated by a planner as a PNP.

There are two advantages in using this approach: 1) the plan (i.e., the PNP) can be executed and monitored with the PNP execution engine and thanks to the PNP-ROS bridge it can be easily integrated in a ROS environment; 2) the PNP can be extended and modified for dealing with unexpected situations (not modelled in the planning domain) during the execution of the actions, making the plan more robust for execution in real scenarios. The extension/modification of a PNP can be done either by manually modifying the PNP or by defining a set of execution rules and apply them with the software provided in the present library.

PNPgen is a library included the Petri Net Plans distribution that contains the implementation of several algorithms and functions for the generation of PNPs.

3.3.2 PNPros

ROSPlan is a framework that provides a method for general planning under ROS. It encapsulates both planning and plan dispatch.

PNPgen is the one responsible for the automatic generation of PNPs starting from a planner output.

PNPros serves as a bridge between ROS and PNP and allows to control the generated PNP execution in ROS. The aim of this interface is to map the output of ROSPlan, a computed plan, to the input of PNPgen module in order to produce robust conditional plans. Robust conditional plans are plans that comprehend branching and can represent loops, interrupts, skip-action, etc.

3.4 Plan Execution Interface (PLEXI)

A Plan Execution Monitor (PEM) aims at orchestrating the execution of actions and monitor the values of fluents (or predicates), following a plan generated to achieve a given goal. The implementation of the interface between a PEM and action/fluent implementations may be non-trivial and it is usually achieved through specific coupling PEM and action/fluent implementations.

Plan Execution Interface (PLEXI)[31] is a layer for increasing interoperability between a PEM and the implementation of actions and fluents in a complex system (e.g., a robotic application). PLEXI provides an interaction protocol between PEM and action/fluent implementations, allowing for decoupling all these components.

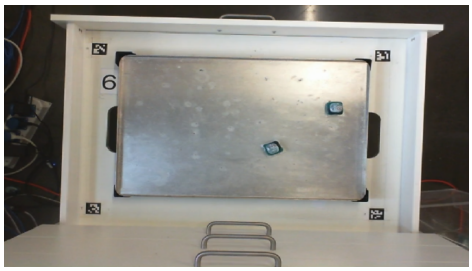
With PLEXI, action/fluent implementations will not depend on the specific PEM. There is no need to import libraries or include code from the PEM, there is no need to know which PEM will orchestrate the implemented actions/fluents.

Interaction between PEM and actions/fluents will use the FLEXI protocol over ROS and gRPC communication channels. ROS/gRPC Python/C++ implementations can be mixed in the same application. All implemented actions/fluents, each one using its own language (Python/C++) and communication layer (ROS/gRPC), will be managed by the same PEM.

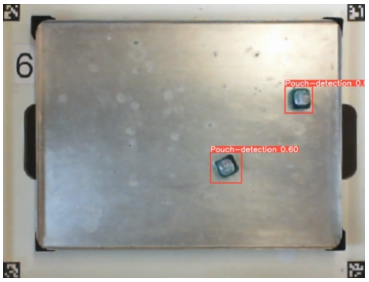
3.5 Other software and tools used

In this section will be covered other tools and software used for this work, that are:

- **OpenCV**[26]: used for capturing and pre-processing images from camera input.



(a) Original



(b) Processed

Figure 3.5. Image pre-processing for pouch object detection. The input image (Original) is transformed into output image (Processed).

- **ArUco Markers**[6]: for finding correspondences between points in the real environment and their 2d image projection that is useful for the pose estimation problem.
- **Roboflow**[27]: used to label and annotate objects (in this case the laundry pouches) from images for the training dataset creation.
- **YOLOv5**[4]: used for the training process about the object recognition, for pouch detection in this case (as shown in Figure 3.5b). For this purpose the training was made with only 25 images in low resolution.
- **Tesseract**[4]: used for text recognition and extraction from an image (for example the current drawer number open).

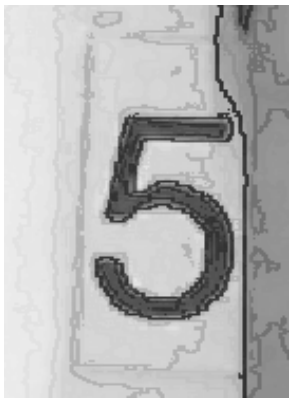


Figure 3.6. Drawer number recognition from preprocessed image.

In particular this tool[5][25] has been used, in case some drawer is already open, to recognize which of the available drawer is currently open. This was very important in case the human forget to close the drawer and so for processing the right pouches (to reach the exact $z - axis$ value)

- **PySimpleGUI**[5]: used for the creation of a GUI useful for the normal user (not necessary an expert on robotics or programming) to interact with the experiment process.

  This GUI is very important for the Human-Robot Interaction and, in case of remove starting experiment. control the advancement of each product test and state of the robot and plan, for example if any action failed, and take action if requested.

[4] `https://pypi.org/project/pytesseract/`
[5] `https://pypi.org/project/PySimpleGUI/`

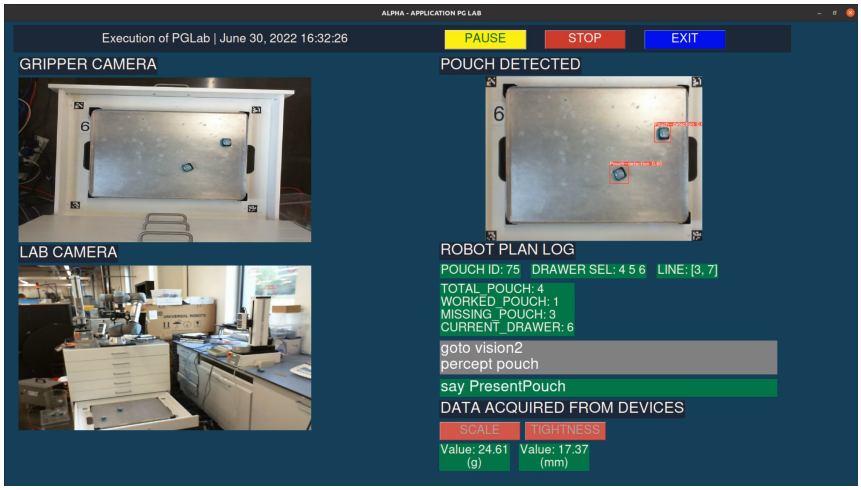


Figure 3.7. GUI application used for HRI.

Chapter 4

Robot and Devices

4.1 UR5e

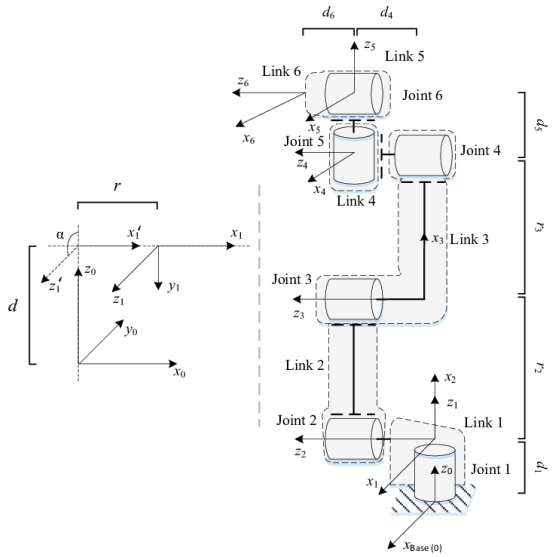
Universal Robots, founded in 2005 by three engineers, is a Danish manufacturer of smaller flexible industrial collaborative robot arms (cobots), robots designed to physically interact with humans in a workspace. In 2018 a new generation of Universal Robots' cobots called e-Series was launched, such as the UR5e used in our work that is the successor of UR5 (available on the Danish and German markets since 2008) model. The UR5e[32] is a lightweight, adaptable collaborative industrial robot that tackles medium-duty applications with ultimate flexibility. The UR5e is designed for seamless integration into a wide range of applications. This is how Geoff Escalette, CEO and RSS Manufacturing of Universal Robots, described the robot:

The UR5e does in 4 hours what it would take manual labor 2-3 days to accomplish. This progress has made it possible for us to compete with overseas manufacturers and bring back manufacturing jobs to the U.S.

Figure 4.1. UR5e robot in working position.

The lightweight, highly flexible, and collaborative UR5e industrial robot arm lets you automate repetitive and dangerous tasks with payloads of up to 5 kg. The UR5e flexible robot is ideal to optimize low-weight collaborative processes, such as picking, placing, and testing. With a working radius of up to 33.5 ins (850mm), the UR5e collaborative robot puts everything within reach, freeing up your employees' time to add value to other stages of production. The UR5e robot from Universal Robots is easy to program, offers fast set-up, is collaborative and safe, and like our other collaborative robots, offers one of the fastest

payback times in the industry.



UR5e							
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]
Joint 1	0	0	0.1625	π/2	Link 1	3.761	[0, -0.02561, 0.00193]
Joint 2	0	-0.425	0	0	Link 2	8.058	[0.2125, 0, 0.11336]
Joint 3	0	-0.3922	0	0	Link 3	2.846	[0.15, 0.0, 0.0265]
Joint 4	0	0	0.1333	π/2	Link 4	1.37	[0, -0.0018, 0.01634]
Joint 5	0	0	0.0997	-π/2	Link 5	1.3	[0, 0.0018,0.01634]
Joint 6	0	0	0.0996	0	Link 6	0.365	[0, 0, -0.001159]

Figure 4.2. UR5e robot parameters by Denavit-Hartenberg(DH) method.

For the robot connection, please refere to A.

4.2 Robotiq 2F-85

Robotiq is a company founded in 2008 that has the mission to make human hands free from repetitive tasks. Its tools and know-how make collaborative robot applications simple, so that factories can make productions at faster rate.

Figure 4.3. Robotiq 2F-85 gripper.

The Robotiq 2F-85[39][41] is one of the best-selling grippers in the world for collaborative robots. It is characterized by its user-friendliness and flexible possibilities. By installing a simple software plugin, the 2F-85 is easy to program with the software belonging to the collaborative robot.

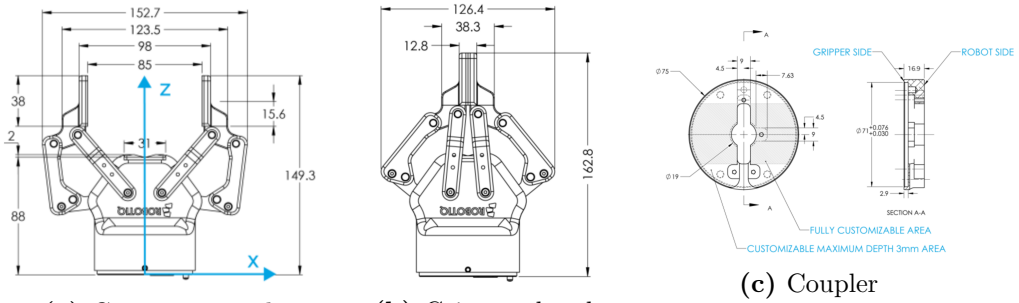
The 2-Finger Gripper[40] has two articulated fingers that each have two joints (two phalanxes per finger). The grasp-type gripper can engage up to five points of contact with an object (two on each of the phalanges plus the palm). The fingers are under-actuated, meaning they have fewer motors than the total number of joints. This configuration allows the fingers to automatically adapt to the shape of the object they grasp and it also simplifies the control of the grasp-type gripper. Below we will show the technical features of the most popular Robotiq grippers on market.

	Hand-E		2F-85		2F-140	
Stroke	50 mm	2.0 in	85 mm	3.3 in	140 mm	5.5 in
Grip Force	20 to 185 N	4.5 to 41 lbf	20 to 235 N	4.5 to 50 lbf	10 to 125 N	2 to 25 lbf
Form-fit Grip Payload	5 kg	11 lbs	5 kg	11 lbs	2.5 kg	5.5 lbs
Friction Grip Payload*	4.7 kg	8.8 lbs	5 kg	11 lbs	2.5 kg	5.5 lbs
Gripper Weight	1 kg	2 lbs	0.9 kg	2 lbs	1 kg	2 lbs
Closing speed	20 to 150 mm/s	0.8 to 5.9 in/s	20 to 150 mm/s	0.8 to 5.9 in/s	30 to 250 mm/s	1.2 to 9.8 in/s
Ingress protection (IP) rating	IP67		IP40		IP40	

*Calculated for the use of silicon covered fingertips to grip a steel object, at a low robot acceleration.

Figure 4.4. Robotiq grippers technical specifications.

The mounting of the gripper is easy and therefore a quick installation is guaranteed. In addition, the robot is equipped with sensors that ensure that the robot can transfer data to the software and perform tasks with high precision. The Robotiq 2-Finger Adaptive Robot Gripper requires a coupling provided by Robotiq to operate. The coupling is mandatory since it integrates electronics and electrical contacts.



(a) Gripper opened **(b)** Gripper closed **(c)** Coupler

Figure 4.5. Robotiq 2F-85 gripper + coupler technical dimensions.[42]

For the gripper connection please refer to 7.

4.3 Microsoft LifeCam Cinema

Another device very important for our work was the camera, that was indispensable for the robot's vision. Thanks to this the robot could view and recognize the objects that are present and surround him in the environment, in particular the laundry pouches, or for understanding which drawer is already open.

The camera used was the Microsoft LifeCam Cinema[43], a full-featured webcam with ClearFrame image-processing technology that delivers smooth, detailed video and crystal-clear audio. This camera has digital, noise-cancelling microphone that helps to improve speech quality. it is equipped with a 720p sensor that captures true high-definition (HD) quality video at up to 30 fps.

4.4 Mettler Toledo Scale

The first test that we carry out in this experiments was the scale or weight test. To perform this test we used the PG5002-S DeltaRange[44] scale, a very accurate scale from Mettler Toledo company, that is the leader of analytical balance.

METTLER TOLEDO offers a wide array of services, including a comprehensive weighing service portfolio, laboratory balance services, industrial scales service, and other measurement equipment services. METTLER TOLEDO instruments are detecting, measuring and weighing everything from nectar in an ant to water purity. From the infinitely small to the largest of vehicles or airplanes, they are leaders across their categories. Their solutions are used across many laboratory and industrial applications in research and development, quality control, process analytics, product inspection, and retail businesses worldwide.

Analytical balances[45] are highly accurate weighing instruments. Due to the sensitivity of the high-performance weighing cell, analytical balances are fitted with a draft shield to ensure a stable weighing environment. With capacity ranging from 52 to 520 g and readability from 0.002 to 1 mg, common analytical balance applications include sample and standard preparation, formulation, differential weighing, density determination, and pipette routine testing.

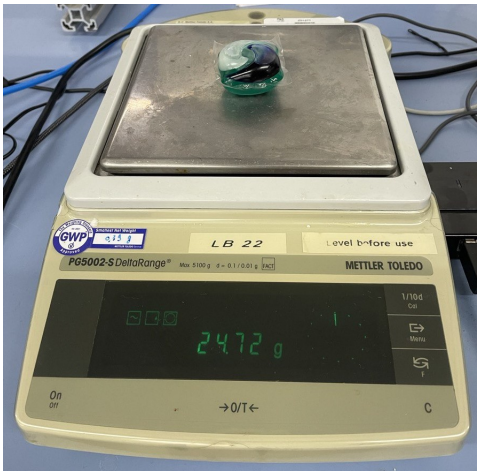


Figure 4.6. Scale used for the weight test.

The Mettler Toledo scale PG5002-S is a top-loading balance gives you basic weighing functions in a handy, easy to place footprint. This digital lab scale has a capacity of up to 5100g, with full capacity tare. This rugged and chemically resistant lab balance has an easy to use keypad for one-handed operation, with a large clear digital display. Built-in functions for this lab scale include piece counting, 13 selectable and switchable weigh units, motorized adjustment with a FACT system, and an RS232 interface. To ensure repeatable results, this lab scale also features a leveling bubble to determine exact horizontality, to make adjustments after a location change, the two back feet can be turned.

4.5 Mark-10

In this type of experiments is necessary to perform the "tightness" and "strength" tests. For doing this was used the Mark-10 Motorized Test Stand models ESM301 and ESM301L[46]. Mark-10 is a fully integrated US designer and manufacturer of force and torque measurement solutions founded in 1979.

The ESM301 and extended length ESM301L are highly configurable motorized test stands for tension and compression testing applications up to 300 lbF (1.5 kN) for laboratory and production environments. The stands can perform a wide range of tests, including break testing, cycling, limit testing to a load or distance, load holding, elongation testing, tensile testing, compression testing, and more. Satisfy various test methods through an easy-to-use menu, allowing the configuration of test speed, force and distance limits, cycling, preload, and many other functions. The stands can be controlled via its console or PC. Functions are offered individually through a unique “build-your-own” configuration platform. Order any combination of functions to customize the stand to specific requirements. Functions not ordered originally can be activated in the field.

The controller features an integrated angle display and output, programmable angle travel limits, PC control capability, programmable cycling, auto return, overload protection, password protection, and more.

Chapter 5

Problem overview

5.1 The aim of our work

Our aim in this work was to make automated tests on laundry pouches. This tests has to be done through the robot that interacts with the surrounding environment such as instruments, drawers and also humans.

A single product experiment consists in:

- Detecting the pouch position
- Grasping the pouch
- Executing the weight test
- Executing the tightness test
- Executing the strength test[1].
- Trashing the pouch on the bin

The robot has to iterate for every pouch present in the tray of the drawer and this procedure has to be repeated for all the available drawers.

At each test during an experiment, a new value in the dataset[2] is added. The position where this value is wrote depends on the pouch ID and in which test was performed. This is very fundamental for this work because in this way was possible to track, if there is any, the defects of each pouch. The dataset reports the following information:

- Date and Time
- Pouch ID
- Weight (g)
- Tightness (mm)
- Strength
- Line

During each experiment, some *failures or unexpected events* can occurs and the robot has to be ready to deal with them by itself or with human help. This events or failures can regard the missed pouch during grasping, the loosing of the pouch during its transportation to an instrument, instruments or robot loss connection, robot collision with some scene object and so on.

The solution that we implemented deals with this type of events/failures and it is shown in the diagram of the following section.

5.2 Problems encountered

- **Camera's visibility**: dust, steam and light reflection on the tray. This occurs mainly as we were working in a laboratory where there is produced a lot of dust, steam from the machines. And the other problem occurs as near the robot was a big window where in a certain time of the day the sun lights were very strong and they reflects on the metallic tray. This could cause an error in the pouch detection (as shown in 5.1)

[1] present in this work but it can be easily added in the future works with the presence of another robot with a vacuum gripper.

[2] The creation of the dataset is made with *pandas*[3] python library.

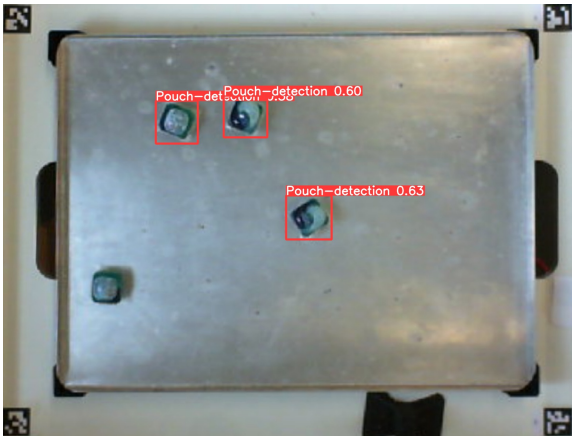


Figure 5.1. Light reflection on bottom left of the tray.

- **Camera's release and re-activation**: this problem occurs when the camera is released and then reactivated. In this way the photos will be of very bed quality because the camera needs a certain time for light adaptation. To solve this problem we discarded a certain number of photos (for us worked very well 50 photos) before the good shoot.
- **Camera's wire**: this problem occurs when the robot for reaching a goal position performs some further rotations and the camera's wire could get stuck with robot body due to the friction. To solve this we put some elastic plastic tubes where the wire can flow easily.
- **Gripper connection**: this problem took for us a lot of time and it is very common. It's solution is better explained in chapter 7.
- **Robot *bringup* disconnection**: this problem occurs where then the robot and PC-ROS are connected to the same network and there is Internet access. As there are a lot of data requests, especially if any browser is running, such as advertising or background applications (like Gmail, Telegram, ...) there can be an interference between the two devices and this entails to the robot disconnection. To avoid as much as possible this problem the workaround should be to close all the applications that works with Internet (such as browsers) or turn Off the internet connection.

Chapter 6

Solution design

6.1 Laboratory setup

The complete laboratory setup is made as shown bellow:

Figure 6.1. Laboratory complete setup.

The elements shown in 6.1 are the following:

- 1) the complete robot: a robotic arm with a gripper and camera attached.
- 2) the Mettler Toledo scale used for getting the weight of the pouch.
- 3) the Mark-10 used for measuring the tightness of the pouch.
- 4) example of a laundry pouch used for testing.
- 5) the bin in which the pouches have to be trashed after each experiment.
- 6) the Mark-10 used for measuring the strength of the pouch (this experiment was not included in this work but is ready to work, the "vacuum" gripper is needed).
- $\{d_1, \ldots, d6\}$ are the available drawers in which is possible to put a tray of pouches.

6.2 Diagram

In this diagram6.2 is shown the flow of the final plan implemented for the experiment tests execution on laundry pouches. The plan generated is composed by a set of actions, fluents, conditions and also Execution Rules (ER)[1]. This diagram is composed by different nodes with different colors where each color represents its implementation origin and its domain as shown below:

- Green: the modules implemented by me.

[1]ER are statements that allows to define additional conditions to a conditional plan.

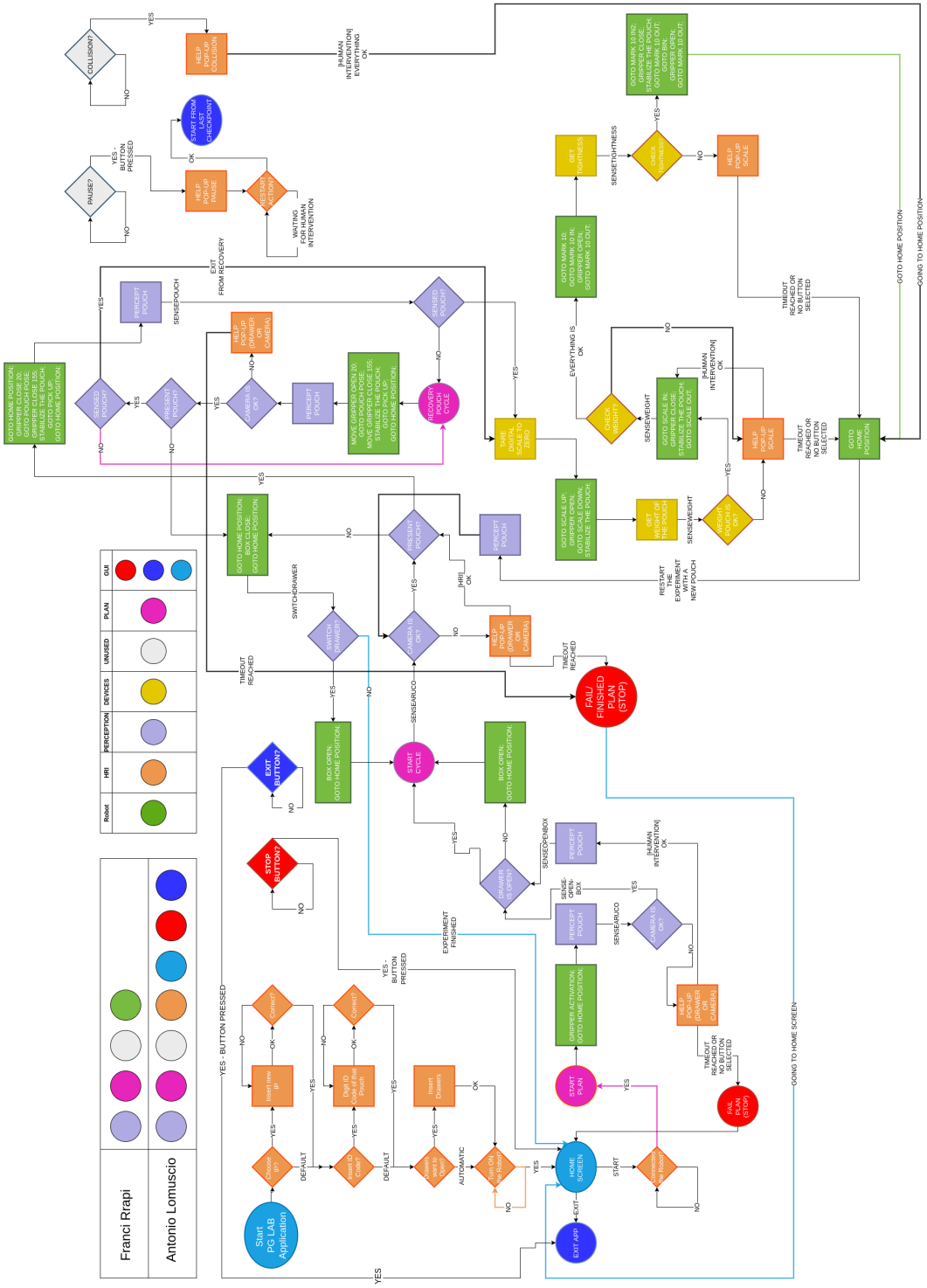


Figure 6.2. Diagram of PG LAB plan.

- Violet, Red Violet, Gray: the modules implemented with the collaboration of my colleague.
- Orange, Light Blue, Dark Blue, Red: the modules implemented by my colleague.

For the execution of the plan was useful the creation of a new MoveIt! robot configuration (that includes also the gripper attached), new actions and fluents for controlling the robot, interfacing with the laboratory instruments and also recovering from unexpected events.

6.3 Actions

The actions implemented for the plan execution are:

- *goto*: this action performs complex movements of the robot for reaching various joint states or pose goals that are reachable in the environment.
- *box*: this action performs lots of movements of the robot for opening or closing a certain drawer.
- *movegripper*: this action is useful for controlling the gripper, such as the activation, reset, full gripper open/close, open/close with a certain middle value or change the open/close speed/force.
- *percept*: this action performs the perception of the pouches with the extraction of their exact position with respect to the robot and also the number recognition of the current drawer.
- *scale*: this action performs the interaction with the scale (through *USB* connection) during weight test.
- *tightness*: this action performs the interaction with the Mark-10 (through *socat* connection) during tightness test.
- *strength*: this action performs the interaction with the Mark-10 (through *socat* connection) during strength test.
- ***say***: this action performs a simple label print on screen.
- ***wait***: this action performs a time delay during some action.

The implementation details of the actions in green, that have been implemented by me, and the ones in violet implemented with the collaboration of my colleague, will be explained in Chapter 7. The actions in yellow have been implemented by my colleague and are better explained in his thesis and the ones in **black** were already implemented from the previous work[3].

6.4 Fluents

The fluents implemented for the plan execution are:

- *presentpouch*: returns *true* if at least one pouch has been detected, *false* otherwise.
- *switchdrawer*: returns *true* if at least one drawer is available to be open, *false* otherwise.
- *sensepouch*: returns *true* if the pouch has been grasped, *false* otherwise. To know if pouch has been grasped another pouch perception is performed; if the number of pouches detected is different from the previous perception, it means that the pouch with high probably has been grasped.
- *failure*: returns *true*[2] if the robot failed to finalize a *goto* action, *false* otherwise.

[2]The reasons of the failure could be the robot *bringup* disconnection or collision with scene object or human.

- *sensearruco*: returns *true* if the number of arucos detected is equal to 0 or 4, *false* otherwise.
- *pause*: returns *true* if a *pause* request has been received from the GUI, *false* otherwise.
- *senseopenbox*: returns *true* if all the ArUco markers has been detected and the drawer number has been perceived, *false* otherwise.
- *sensestrength*: returns *true* if the value perceived during the strength test exceeds a threshold, *false* otherwise.
- *sensetightness*: returns *true* if the value perceived during the tightness test exceeds a threshold, *false* otherwise.
- *senseweight*: returns *true* if the value perceived during the weight test exceeds a threshold, *false* otherwise.

6.5 HRI: Recovery procedures

The failure or unexpected events that we treated and that the robot is able to deal with are:

- ***Missed pouch***: when this event occurs (*sensepouch* = *False*), the robot enters in the RECOVERY POUCH mode. During this "subplan", the robot tries to grasp the pouch until it succeeds if at least a pouch is available in the tray. Then the execution resumes from the point where the failure occurred.
- ***Scale loss pouch***: when this event occurs (*senseweight* = *False*) a Pop-up appears from GUI that asks for human intervention. This HRI occurs when the pouch is loss between the tray and the scale, and the human is asked to manually find and place the pouch above the scale and then insert the weight value. Then the execution resumes from the point where the failure occurred. If the human refuses to accomplish this task, the experiment of that pouch will be discarded and a new experiment begins.
- ***Scale check weight***: when this event occurs (*senseweight* = *True*) a Pop-up appears from GUI that asks for human intervention. This HRI occurs when the weight value of the pouch was get correctly but the the robot did not grasp it correctly to remove the pouch from scale. In this case the human is asked to manually transfer the pouch from scale to the Mark-10 tightness base. Then the execution resumes from the point where the failure occurred. If the human refuses to accomplish this task, the experiment of that pouch will be discarded and a new experiment begins.
- ***Tightness/Strength HELP Pop-up***: this HRI events are the same as the last two but with the difference of fluents they work with, that are *sensetightness* and *sensestrength*. In this HRI also the human is asked to place the pouch in the next step or manually perform a new test and insert the result in the dataset.
- ***Camera problem***: when this event occurs (*sensearruco* = *False*, number of ArUco markers perceived $\in [1, 3]$) a Pop-up appears from GUI that asks for human intervention. This HRI occurs when the camera is not working properly or it is crooked. In this case the human is asked to control the camera and fix the problem.
- ***Failure or Pause***: this events occurs when the human press the *PAUSE* button on GUI or when the there is a loss robot connection. In the first case the plan resumes when the *START* button is pressed and in the last case the human is asked to reconnect the robot. The implementation of this Recovery procedures has been implemented through the *Execution Rules (ER)*.

6.6 Solution advantages

This solutions offers better work environment conditions, where robot and human can collaborate together to solve a common task or help each other in some specific tasks that the other can not accomplish. For example there could be some tasks that, even if the robot is completely autonomous and never fails, cannot execute because of its limits. So in this case is fundamental the human intervention. For this reasons it is very important the presence of HRI for the communication between humans and robot.

Chapter 7

Implementation details

7.1 Camera calibration

The first thing to do was to have a correspondence between camera pixel and real coordinates. To achieve this transformation between the two different coordinates (camera vs reality) we had to use the ArUco Markers. In particular we used the marker class from OpenCV $DICT\_6x6\_250$. We used only the first 4 markers of this dictionary (so markers with ids $\in [0, 3]$) and attach them in exact position on the drawer.

In this way, when we capture a photo, the steps done are:

- The image is first calibrated with *Homogeneous Transformation*
- The ArUco marker are detected with known position
- The image than is cropped and resized with $size = (Width, Height)$ with the following values:

$$Width = W \cdot s \tag{7.1}$$
$$Height = H \cdot s \tag{7.2}$$

Where W and H correspond to Width and Height dimensions of the ArUco plane in the drawer. The s instead is the scale factor used for the conversion, in this case we used $s = 2.0 \cdot 10$

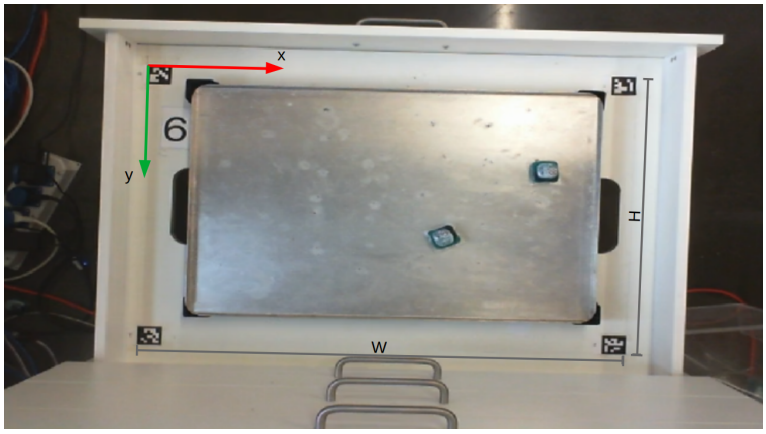


Figure 7.1. ArUco plane dimensions and reference frame.

In this way, we can easily know the exact real position of each object in the picture.

7.2 Gripper-PC Ros connection

One of the most insidious problems encountered during this work was the Gripper conncection and comunication with PC Ros.

We spent a lot of time on trying to solve this problem, making many and various attempts, but without success. We also searched various tutorials and forums but without an answer that could solve the problem.

In the end, after a large number of tests and many combinations, we were able to make the right one which made the gripper attached to the robot work perfectly.

The steps to be done in such a way that the gripper works are:

1. Be sure that the *Gripper URCap* is uninstalled from UR Teach Pendant

2. Install on robot **rs485-1.0.urcap** that you can find in [33]. Same installation made also for the **external-control** URCap. After installing the URCap the robot will expose its tool communication device to the network.

3. In order to use the tool communication in ROS, simply pass the correct parameters to the bringup launch files, that are:

   - use_tool_communication:=true
   - tool_device_name:=/tmp/ttyUR

   So, the complete robot bringup launch command from terminal will be:

```
$ roslaunch ur_robot_driver <robot_type>_bringup.
    launch \
  robot_ip:=<robot_ip> \
  kinematics_config:=${HOME}/pg_lab/configs/<
     robot_type>_calibration.yaml \
  use_tool_comaunication:=true \
  tool_device_name:=/tmp/ttyUR
```

   In our work the <robot_type> was ur5e_2f85. Once the robot connection is done and the communication tool started, we can continue to connect the gripper.

4. In another terminal we run the gripper communication node (that is possibile to find in package [34]) as shown below:

```
$ rosrun robotiq_2f_gripper_control
    Robotiq2FGripperRtuNode.py /tmp/ttyUR
```

   The gripper results to be connected when the light changes from red to blue and it is a fixed light.

5. Once the gripper has been connected, we can start a *Controller* node that sends commands to the gripper. An axample of controller could be:

```
$ rosrun robotiq_2f_gripper_control
    Robotiq2FGripperSimpleController.py
```

7.3 Pouch pose

Once the robot was fully configured and working, one of the first problems we occurred and needed for a solution was the exact position of the pouch with respect to the robot.

When we capture a picture from the camera, that picture refers to the reference frame of the camera, so we had to apply a transformation in such a way it refers to the robot reference frame (in this case we used the "base_link" frame).

But before this we had to detect the ArUco markers on the drawer; we had then to preprocess, cut and resize the image for the that will be the input image for the NN object detetction (in this case we used YOLOv5).

So after YOLOv5 detected the object in the image, we have the pose of the pouch wrt to the image (that is wrt to aruco plane) as $(\bar{x_I}, \bar{y_I})$.

For doing the transformation we had to find before the relation between the Image and Robot reference frames.

$$x_I^R = -W/2 \tag{7.3}$$
$$y_I^R = d_I^R + H \tag{7.4}$$
$$\tag{7.5}$$

To make a good transformation, we resized the image with a scalar of the aruco drawer dimensions such that we don't loose any precision in the conversion.

$$x_R = \bar{x_I} + x_I^R \tag{7.6}$$
$$y_R = -\bar{y_I} + y_I^R \tag{7.7}$$
$$z_R = -(d_1 + d_2 + 11.5 * (n-1)) \tag{7.8}$$

As we managed with various drawers, and so with different height, the z value depend on the number of the current drawer (n).

where W is the weight/length of the Aruco Drawer, H is the height of the Aruco Drawer, R = robot, I = image of drawer.

So, at the end, the position of the pouch with respect to the Robot is given by the tuple (x_R, y_R, z_R).

7.4 Interpolate over grasping orientation

Another problem we met, was the gripper orientation. Not only for an estethic reason, but also for a better ans safe pouch grasping.

This because if the gripper reach the pouch position in perpendicular way, so with an orientation (180, 0, 0) degrees, when it goes down can collide with the (maniglie) of the drawer.

For not having this problem, we took the orientations in 4 different points, that are the 4 points on each extremity of the drawer. So we extracted in this way the bounded positions (in this case a rectangle) and also the orientation interval for each frame axis.

Then we calculated the interpolated angles of the gripper through a combination of linear interpolations (also called quadratic interpolation, used to interpolate in a plane) and the procedure is shown below:

- First we interpolated linearly between P0 and P1 → named Ph1
- Then we interpolated linearly between P2 and P3 → named Ph2
- Finally, we interpolated between Ph1 and Ph2

Below is shown as an example the linear interpolation for the α, β and γ angle between P0 and P1:

$$\alpha = \alpha_0 + ((x - x_0)/(x_1 - x_0)) * (\alpha_1 - \alpha_0) \tag{7.9}$$
$$\beta = \beta_0 + ((x - x_0)/(x_1 - x_0)) * (\beta_1 - \beta_0) \tag{7.10}$$
$$\gamma = \gamma_0 + ((x - x_0)/(x_1 - x_0)) * (\gamma_1 - \gamma_0) \tag{7.11}$$

In case of a linear interpolation between P2 and P3, below is an example of α angle intderpolation:

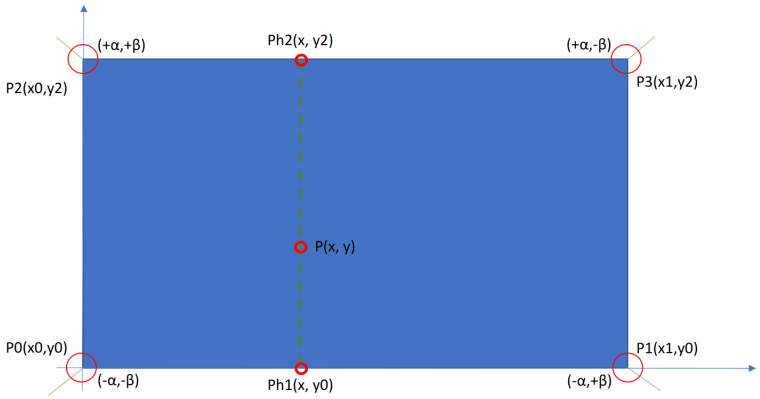


Figure 7.2. Interpolation used for gripper orientation.

$$\alpha = \alpha_2 + ((x - x_2)/(x_3 - x_2)) * (\alpha_3 - \alpha_2) \tag{7.12}$$

In this way we got the final alpha, beta and gamma angles for the grasping pouch orientation.

In our work, we interpolated only for the α and γ robot gripper angles since β was not in our interest. In particular, the values were in a range as showed below:

- $\alpha \in (-160, -200)$
- $\gamma \in (-45, 45)$

The reason of the bounded α between that values, and not for example $(-135, 135)$, is that the default alpha grade of the gripper orientation before grasping pouch is set to 180° with respect to robot "base_link" frame. So in this case we want 180° to the middle value (not 0° as γ) and put its range in ± 20.

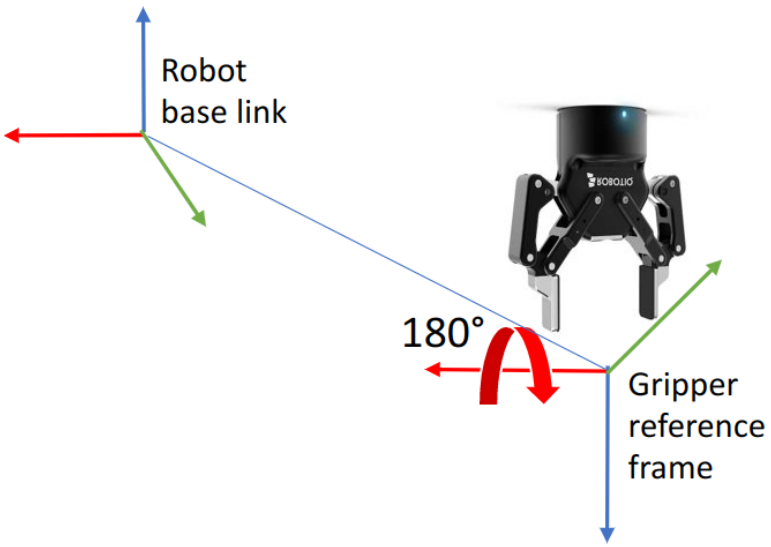


Figure 7.3. Gripper default orientation.

Notice that we are talking about a default orientation of gripper wrt robot of 180° and $\alpha \in (-160, -200)$. This because the robot has limited joints, in this case ±180°, and only in this case the angle can be in the same time even positive and even negative.

7.5 MoveIt! Robot Configuration

As we were dealing with a robot made by a robotic arm and a gripper attached, we needed a new MoveIt! configuration for using properly the Inverse Kinematics and founding the right trajectory for reaching a certain goal position.

For doing this we created a new *xacro* file (another way to define a URDF model) through the *MoveIt Setup Assistant* that includes both robot UR5e and gripper 2F-85, attach this one in the "flange" link of the robot.

Also, we defined another link (named *gripper_tip_link*) that will be the **end effector** planning reference frame.

7.6 Collision avoidance

One of the most problematic issue of this work was the avoidance of collision with objects that were present in the scene. This occurs because the MoveIt! trajectory is not predictable and not known a priory as it depends in many aspects and current joint states, that are always different.

Sometimes the points that MoveIt! finds as a trajectory to reach a desired pose goal, can not be simple paths but paths with weird joint rotations and this can lead to strange robot movements and also collision.

A way to avoid this was to add to the MoveIt! planning scene the most important objects that are present in the scene, such as the table where the robot is located, the table where the instruments are located, the instruments itself and also the camera attached to the gripper.

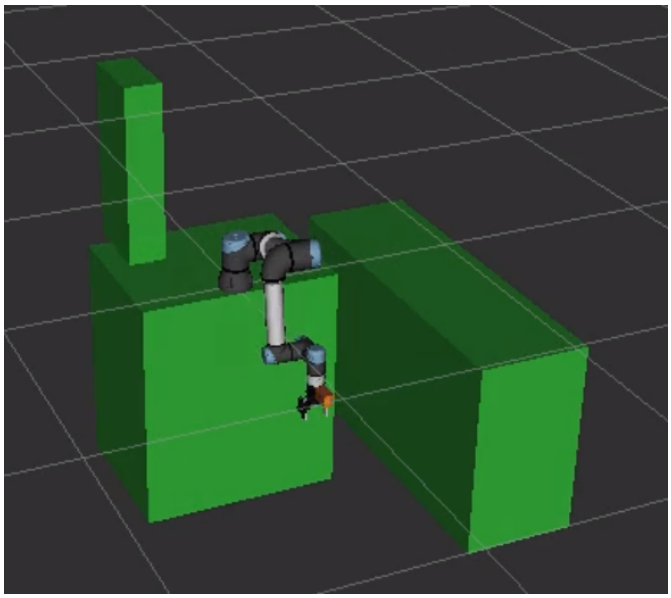


Figure 7.4. MoveIt! Planning Scene objects.

All this objects scene added to MoveIt! avoided collision of the robot during its movements. In this way MoveIt! calculates a trajectory for reaching a target goal knowing

that in the environment around him there are objects that he must take into account in his movements.

Chapter 8

Experimental results

In this chapter I will show some of the experiment execution results obtained with the solutions described in the above chapter.

The next image shows the robot while performing the ***box*** action, for opening the drawer.

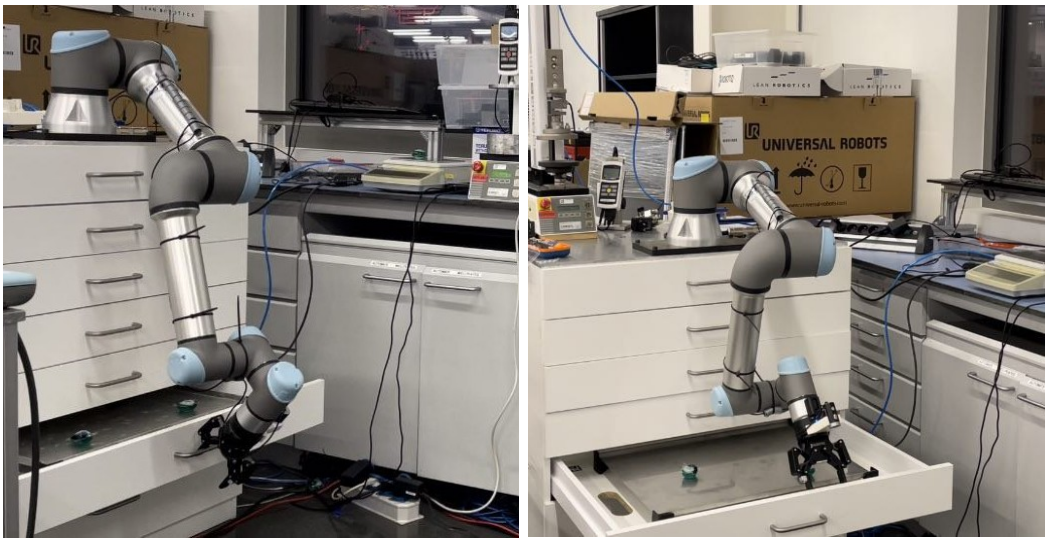


Figure 8.1. Robot while opening the drawer.

Figure 8.2. Robot while grasping pouch (with angles interpolation). From the left image we can see that thanks to the interpolation the robot reaches the pouch avoiding collision between its internal body and other drawers.

Figure 8.3. Robot reaching the Mark-10 for "tightness" test from Scale device on the left. Robot waiting for test termination.

Chapter 9

Discussion and Future Works

The results achieved in this work have been very satisfying for us but, despite this, it is clear that there are many things that can be done in order to improve the models and the application, to refine or to add new features. These points could be the objective for future thesis works.

Having worked up to this point in this project, therefore having a full picture of the path to arrive till here and also being able to recognize what future developments could be, below I will illustrate main points to work on for future developments:

- Pause and Failure recovery integration: the functioning of these two recovery procedures in the plan would be a step forward in order to further increase the handling of unexpected events.
- More Human-Robot Interaction: with a more communicative relationship between robots and humans, would be possible to reduce dangerous and risky situations by having the situation under control and managing all unexpected events, thanks to a very efficient information exchange between robots and humans.
- Re-trained dataset: this would mean an even more accurate detection of the pouches and would greatly improve the accuracy of the robot, avoiding failures.
- Multi robot system: by having two or more robots for our experiments, would make possible to speed up the execution of each experiment thanks to the collaboration between robots in completing tasks, a sort of team working, and would also make the strength test possible, thanks to the mounting of the vacuum gripper in one of them.

Appendix A

Driver and Packages

A.1 Universal Robot Driver

The Driver used for the connection between UR5e and PC-ROS is Universal_Robots_ROS_Driver[33] from UniversalRobots GitHub repository[1] supporting both CB3 and e-Series.

The complete guide for setup and configuration for robot and pc side is available in their GitHub repository.

A.2 UR MoveIt! configurations

As indicated in the **building** step from Universal_Robots_ROS_Driver repository, for our system configuration, we used the calibration-devel branch of universal_robot package[35] (from fmauch GitHub repository) for the MoveIt! configuaration packages for our UR robot.

The package provides the robots configurations for working with the external tools, as 3D simulators or Kinematics solver, with all the kinematic chain and robot links description.

A.3 Robotiq MoveIt! configuration

For the Robotiq gripper configuration files, descriptions, connection and controller, we used the robotiq package[34] from ros-industrial GitHub repository.

The package provides the gripper configuration for working with the external tools, as 3D simulators or Kinematics solver, with all the kinematic chain and gripper links description. It also provides the Rtu communication useful for connecting gripper to the PC-ROS and a simple controller to send commands to it.

A.4 Robot connection

For the robot connection, the steps to follow are:

- Calibrate the robot (only in the first connection):

```
$ roslaunch ur_calibration calibration_correction.
    launch \
     robot_ip:=<robot_ip> \
     target_filename:="${HOME}/pg_lab/configs/<
        robot_type>_calibration.yaml"
```

[1] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

- Bringup the robot:

```
$ roslaunch ur_robot_driver <robot_type>_bringup.
    launch \
     robot_ip:=<robot_ip> \
     kinematics_config:=${HOME}/pg_lab/configs/<
        robot_type>_calibration.yaml \
     use_tool_communication:=true \
     tool_device_name:=/tmp/ttyUR
```

  Note: after this command start the external control program from robot side and a message should appear in case of successful connection.

- Launch MoveIt planning execution:

```
$ roslaunch <robot_type>_moveit_config \
     <robot_type>_moveit_planning_execution.launch
```

- Launch MoveIt RViz visualization:

```
$ roslaunch <robot_type>_moveit_config moveit_rviz
    .launch
```

- After the robot has been connected, you can continue with the gripper connection as showed in 7.

Appendix B

Software and tools

B.1 Docker image used

As in this work we had to deal with real experiments, and so with real robot that required some drivers[33] installation for the connection between ROS Pc and Robot, the docker image used in the previous thesis work[3] was not compatible in this case as this driver requires new System and ROS versions. So what we did was to create a new Docker image[12] from scratch, with new versions of the system and ROS and with all the required packages, libraries and configuration needed for this work. In particular, the final Docker image was build with:

- OS: Ubuntu 18.04 LTS
- ROS: Melodic

In this image, in addition to the general libraries and packages (such as OpenCV, NumPy, ...), are also been installed the Universal Robot ROS Driver[33], the ROS Robotiq[34] and the Universal Robot Moveit! configuration[35] packages useful for the complete control of the robot.

The docker image used for this work is pg_lab_ur5:0.5 in francirrapi/ docker repository.

B.2 OpenCV

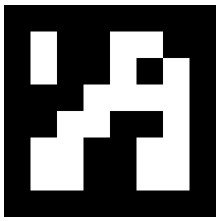
OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

B.3 ArUco Markers

Pose estimation is of great importance in many computer vision applications: robot navigation, augmented reality, and many more. This process is based on finding correspondences

between points in the real environment and their 2d image projection. This is usually a difficult step, and thus it is common to use synthetic or fiducial markers to make it easier.

One of the most popular approaches is the use of binary square fiducial markers. The main benefit of these markers is that a single marker provides enough correspondences (its four corners) to obtain the camera pose. Also, the inner binary codification makes them specially robust, allowing the possibility of applying error detection and correction techniques.



An ArUco marker is a synthetic square marker composed by a wide black border and an inner binary matrix which determines its identifier (id). The black border facilitates its fast detection in the image and the binary codification allows its identification and the application of error detection and correction techniques. The marker size determines the size of the internal matrix. For instance a marker size of 4x4 is composed by 16 bits.

Sometimes can happen that a marker can be found rotated in the environment, however, the detection process needs to be able to determine its original rotation, so that each corner is identified unequivocally. This is also done based on the binary codification.

A dictionary of markers is the set of markers that are considered in a specific application. It is simply the list of binary codifications of each of its markers.

The main properties of a dictionary are the dictionary size and the marker size. The *dictionary size* is the number of markers that compose the dictionary. The *marker size* is the size of those markers (the number of bits).

The marker id simply denotes the marker index within the dictionary it belongs to. For instance, the first 5 markers in a dictionary have the ids: 0, 1, 2, 3 and 4.

For more information about ArUco Marker creation and detection visit the OpenCV dedicated page[6].

B.4 Roboflow

Roboflow empowers developers to build their own computer vision applications, no matter their skillset or experience. It provides all of the tools needed to convert raw images into a custom trained computer vision model and deploy it for use in applications. Today, Roboflow supports object detection, classification models and the most popular annotation formats including JSON, XML, CSV, and TXT. This means you can edit existing annotations and even add new ones to your training images. Once your images are uploaded to Roboflow, you can draw bounding boxes around the objects you want to detect and assign classes as you go.

B.5 YOLOv5

Identification of objects in an image considered a common assignment for the human brain, though not so trivial for a machine. Identification and localization of objects in photos is a computer vision task called ‘object detection’, and several algorithms has emerged in the past few years to tackle the problem. One of the most famous object detection algorithms due to its speed and accuracy is YOLO (’You only look once’). It divides images into a grid system where each cell in the grid is responsible for detecting objects within itself. In particular, YOLOv5 has been implemented using the PyTorch framework.

Figure B.1. Training process for object detection with YOLO.

B.6 Tesseract

Tesseract is an Optical Character Recognition (OCR) tool for python. In other words, OCR systems extract printed text from images. They transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. OCR as a process generally consists of several sub-processes to perform as accurately as possible. The subprocesses are:

- Preprocessing of the Image.
- Text Localization
- Character Segmentation
- Character Recognition
- Post Processing

Tesseract developed from OCRopus (an open-source OCR system) model in Python which was a fork of a LSMT in C++, called CLSTM. CLSTM is an implementation of the LSTM recurrent neural network model in C++, using the Eigen library for numerical computations.

B.7 PySimpleGUI

PySimpleGUI is a Python package that enables Python programmers of all levels to create GUIs. You specify your GUI window using a "layout" which contains widgets (they're called "Elements" in PySimpleGUI). Your layout is used to create a window using one of the 4 supported frameworks to display and interact with your window. Supported frameworks include tkinter, Qt, WxPython, or Remi (browser-based). It transforms this GUI frameworks into a simpler interface. Adding a GUI to a program opens that program up to a wider audience. It becomes more approachable. GUIs can also make interacting with some programs easier, even for those that are comfortable with a command-line interface.

Bibliography

[1] AIPlan4EU: https://www.ai4europe.eu/

[2] AIPlan4EU Project: https://www.aiplan4eu-project.euhttps://www.ai4europe.eu/, https://www.ai4europe.eu/ai-community/projects/aiplan4eu

[3] Mario Fiorino: https://github.com/MarioFiorino/Master-Thesis

[4] YOLOv5: https://github.com/ultralytics/yolov5

[5] Tesseract: https://github.com/tesseract-ocr/tesseract

[6] ArUco markers: https://docs.opencv.org/4.x/d5/dae/tutorial-aruco-detection.html

[7] A brief overview of AI planning: https://users.aalto.fi/~rintanj1/planning.html

[8] MoveIt!: https://moveit.ros.org

[9] Robot kinematics: https://en.wikipedia.org/wiki/Robot_kinematics

[10] Motion Planning: https://en.wikipedia.org/wiki/Motion_planning

[11] Robot Control: https://en.wikipedia.org/wiki/Robot_control

[12] Docker image: https://hub.docker.com/repository/docker/francirrapi/pg_lab_ur5

[13] A Review of Learning Planning Action Models: https://hal.archives-ouvertes.fr/hal-02010536/file/A_Review_of_Learning_Planning_Action_Models.pdf

[14] Human-Robot Interaction: https://en.wikipedia.org/wiki/Human-robot_interaction

[15] Effective Real-Time Visual Object Detection: https://www.cs.cmu.edu/~mmv/papers/12pai-MartinVeloso.pdf

[16] Docker overview: https://docs.docker.com/get-started/overview/

[17] What is Docker? The spark for the container revolution: https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html

[18] What is a Docker Image?: https://www.techtarget.com/searchitoperations/definition/Docker-image

[19] What is ROS?: https://ubuntu.com/robotics/what-is-ros

[20] ROS Wiki: http://wiki.ros.org/ROS/Introduction

[21] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. №3.2. 2009.

[22] ROS Intro: https://www.toptal.com/robotics/introduction-to-robot-operating-system

[23] ROS packages: http://wiki.ros.org/Packages

[24] ROS catkin: http://wiki.ros.org/catkin/workspaces

[25] OCR with Tesseract: https://nanonets.com/blog/ocr-with-tesseract/

[26] OpenCV: https://opencv.org/about/

[27] Roboflow: https://docs.roboflow.com/

[28] YOLOv5 Doc: https://docs.ultralytics.com/#yolov5

[29] Petri Net Plans: https://sites.google.com/a/dis.uniroma1.it/petri-net-plans/

[30] Petri Net Plans A Formal Model for Representation and Execution of Multi-Robot Plans: https://www.researchgate.net/publication/221456969_Petri_Net_Plans_A_Formal_Model_for_Representation_and_Execution_of_Multi-Robot_Plans

[31] PLEXI: https://github.com/iocchi/PLEXI

[32] UR5e robot: https://www.universal-robots.com/products/ur5-robot/

[33] UR ROS Driver: https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[34] Robotiq ROS Driver: https://github.com/ros-industrial/robotiq

[35] UR MoveIt! configuration package: https://github.com/fmauch/universal_robot/tree/calibration_devel

[36] Collaborative robots: https://en.wikipedia.org/wiki/Cobot

[37] Why collaborative robots? https://www.universal-robots.com/products/collaborative-robots-cobots-benefits/

[38] Universal Robots: https://it.wikipedia.org/wiki/Universal_Robots

[39] Robotiq: https://robotiq.com/products/2f85-140-adaptive-robot-gripper

[40] UR robotiq ripperhttps://www.universal-robots.com/plus/products/robotiq/robotiq-2f-85/

[41] Gripper 2F-85: https://wiredworkers.io/product/robitiq-2f-85/

[42] Robotiq 2F-85 manual: https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf

[43] Microsoft LifeCam Cinema: https://www.microsoft.com/en-ww/accessories/products/webcams/lifecam-cinema?activetab=overview%3aprimaryr2

[44] Mettler Toledo: https://github.com/janelia-pypi/mettler_toledo_device_python

[45] Analytical balances: https://www.mt.com/us/en/home/products/Laboratory_Weighing_Solutions/analytical-balances.html

[46] Mark-10: https://mark-10.com/