Implementation of Human-Robot Collaboration in a Manufacturing Environment

Prajval Kumar Murali

EMARO - European Masters on Advanced Robotics

University of Genova, Italy
Ecole Centrale de Nantes, France

Supervisors:
Fulvio Mastrogiovanni, Dušan Tomašec

Co-supervisors:
Kourosh Darvish, Stéphane Caro

In partial fulfillment of the requirements for the degree of

Master of Science (M.Sc) in Robotics Engineering

September 14, 2018

Acknowledgements

I would like to thank my supervisors Prof. Fulvio Mastrogiovanni (UNIGE) and Mr. Dušan Tomašec (Schaeffler) for supervision and support.

I would also like to thank Mr. Kourosh Darvish (PhD student, UNIGE) for his valuable help and mentorship throughout the thesis.

I would like to acknowledge Schaeffler Kysuce, Slovakia for providing the infrastructure and equipment necessary for the thesis. I would also like to thank Mr. Marian Starek for providing me the opportunity to work at Schaeffler and my colleagues at the Industrial Engineering Department at Schaeffler for constant support and availability throughout the internship.

Furthermore, I would like to thank my co-supervisor Prof. Stéphane Caro and other faculty from Ecole Centrale de Nantes for enriching classes and discussions.

Finally, I would like to thank Prof. Renato Zaccaria for allowing me to do an industrial Masters Thesis.

This thesis is dedicated to my parents.

Abstract

The Industry 4.0 paradigm promises shorter development times, increased ergonomic customization, higher flexibility, and resource efficiency. An important tangible technology for implementing this paradigm is collaborative robots or *cobots*. A major concern to effectively deploying cobots to manufacturing industries is developing *task planning* algorithms that enable cobots to recognize and naturally adapt to human actions while simultaneously ensuring overall efficiency in terms of production cycle time. In this context, an architecture encompassing vision-based object pose recognition, task representation, task planning and robot manipulation and motion control has been developed and implemented in a real industrial environment. The architecture uses ROS-Industrial based robot drivers for the UR10 Universal Robots, machine vision systems and AND/OR graphs for task representation and planning. The humans and robots interact in the human-semantic level through an intuitive User Interface. Furthermore, objective measures of overall computational performance and subjective measures of naturalness of human-robot collaboration have been evaluated by performing experiments with production-line operators.

Keywords: Human-Robot Collaboration; AND/OR graphs; Industry 4.0; ROS-Industrial

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

"We're not going to see an exclusively robotic factory, but we will see the optimum use of robots and people." - Dennis Muilenburg

Robots have been used in manufacturing industries from the late 1960s. These industrial robots are very efficient to perform the same mundane, repetitive tasks with high accuracy and efficiency. It was estimated that by the end of 2008, over one million robots have been in operation worldwide [Goodrich *et al.* (2008)] and this number is steadily increasing ever since. However, most of the industrial robots are physically separated from human operators by fences and tasks are divided without work or space overlap. This paved way for the introduction of collaborative robots.

A *collaborative robot* or *cobot* is a robot intended to physically interact with humans in a shared workspace [1]. These cobots can act as co-workers alongside humans in many applications such as manufacturing assembly lines, logistics, personal healthcare and so on. The collaborative robots in literature dates back to pioneering works of Akella *et al.* (1999) as a result of General Motors initiative to find a way to make robots or robot-like equipment safe enough to team with people. In manufacturing industries, cobots can reduce physical and cognitive stress of the worker in the assembly line, and simultaneously improve quality, productivity and safety. This is a key issue, since according to statistics of the Occupational Safety and Health Department of the US Department of Labour, more than 30% of European manufacturing workers are affected by lower back pain, leading to enormous social and economic costs [Cherubini *et al.* (2016)]. This is a major motivating factor for studying human-robot collaboration from an industrial point-of-view. Furthermore, with collaborative robots we combine

[1] https://en.wikipedia.org/wiki/Cobot

the best of both worlds: precision, speed and efficiency of robots with cognitive capabilities and dexterity in dynamic environments of humans.

There are multiple ways in which Human-Robot Collaboration (HRC) can be implemented in an industrial environment [Figure 1.1].

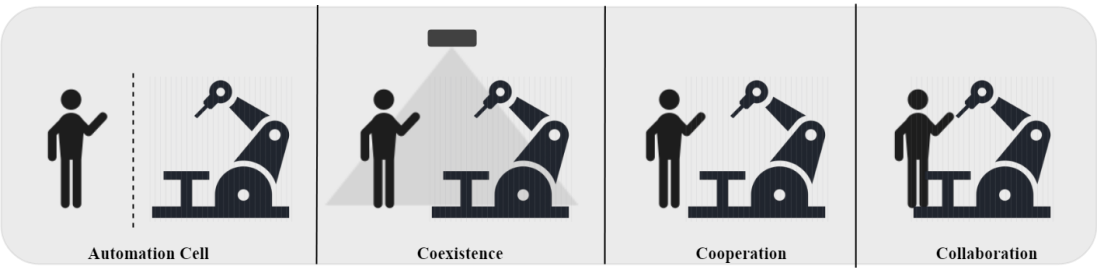


Figure 1.1: Various ways of Human-Robot Collaboration

- **Automation cell** is where the robots and humans are physically separated by fences: mechanical or optical. The robots and humans perform discrete tasks, although there may be task sharing there is no sharing of workspace. The contact is strictly prevented during robot motion. Conventional industrial robots are used in this case.

- **Coexistence** entails a shared workspace between robot and humans. However, humans are monitored by safety sensors and contact between human and robot is strictly forbidden. The robot motion is stopped when humans enter the workspace. Similar to automation cells, conventional industrial robots may be used for coexistence.

- **Cooperation** is where robots and humans share tasks and workspace, and contact between humans and robots is possible during robot motion. This calls for the use of collaborative robots that are inherently safe to work alongside humans. Finally, tasks are **sequentially** processed i.e, tasks are carried out by humans and robots one after the other sequentially. This is also called **co-action**.

- **Collaboration** is the final case where humans and robots share tasks and workspace. Similar to cooperation, it requires an inherently safe collaborative robot. But unlike cooperation, during collaboration there can **joint processing** of tasks i.e humans and robots maybe in contact with the work-piece at the same time and perform their respective tasks.

Most of the collaborative robots deployed in manufacturing industries work in automation cells or coexistence principle [Robla-Gómez *et al.* (2017)]. Although inherently safe, the collaborative robots are not used to their complete potential mainly due to the apprehension of operators and decision makers in industry to allow contact between humans and robots. This thesis will explore the principle of cooperation and collaboration in HRC in a **real industrial environment** and attempt to change the mindset and attitude of operators towards robots.

Secondly, it was studied that people preferred semi-autonomous robots over completely autonomous robots in a HRC scenario [Munzer *et al.* (2017)]. Ideally while sharing tasks between humans and robots, the human satisfaction levels are higher when the people have freedom to choose the tasks they perform. Autonomy of humans can improve the team efficiency, but the trade-off of providing too little or too much control to the workers can be alienating or overwhelming, respectively. Hence this thesis will study the implementation of a novel *Task Planner* and *Task Representation* framework developed at the University of Genova which allows for flexible task allocation amongst agents (namely, humans and robots) [Darvish *et al.* (2017)].

1.1 Description of the scenario

This thesis was performed at **Schaeffler** [1], a global automotive and industrial manufacturing conglomerate, in collaboration with the University of Genova, Italy. The present workstation is shown in figure 1.2, where a group of two human workers check the integrity of a certain automobile part which has to stacked in a pallet container. Due to repetitive bending-pick-place task, a robot is proposed to perform the palleting after the part has been inspected by the human worker.

This end-of-production inspection and packaging is done for a variety of parts in Schaeffler, ranging from inner rings to the complete wheel bearing, Figure 1.3[2]. We will consider the packaging of the inner bearing ring component, henceforth referred to as **part**. Note that the architecture developed in this thesis does not depend on the work-piece and can be configured for any work-piece depending on availability of hardware.

[1] https://www.schaeffler.com/

[2] https://www.carid.com/articles/wheel-bearings.html

Figure 1.2: Human workstation at present

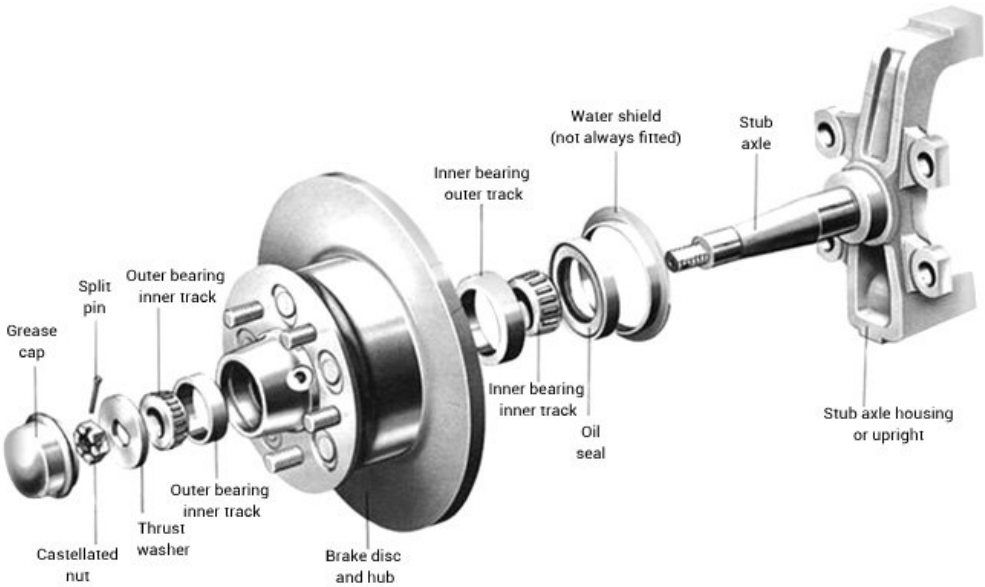


Figure 1.3: Parts of wheel bearing

In the scenario, the operator receives the part on a conveyor from the previous stage of the production line. Assembly must be carried out according to the *One*

Piece Flow methodology [1]. The operator must perform certain actions before palleting the part as follows:

- Receive part from conveyor from previous stage.
- Perform tactile inspection on both edges to detect sharp edges or burrs.
- Perform visual inspection for scratches and defects on the polished surface.
- In case of presence of oil/dirt, clean the part using the provided material.
- Place the part inside the pallet in correct sequence.
- Once a layer of pallet is finished, place the second pallet.
- Log information about number of parts, date, time, details of parts etc and place in the pallet.

During the process of pick-and-place actions by operators, there are possibilities for error:

- Wrong placement of parts in pallet
- Forget to add details of parts inside the pallet
- Forget to log information about the pallet
- Misses defects in the part during visual/ tactile inspection.

These errors may occur due to the continuous physical and mental stress on the operators. Hence the proposed solution is :

- Perform all inspection tasks as above.
- Place part on conveyor which leads to a designated pick-up area.
- A vision system detects the position of the part.
- The robot grasps the part and places it in correct sequence in the pallet box.
- The position of pallet box is known *apriori*.
- For any reason during the robot motion, the operator must be able to stop the robot motion by contact and perform the palleting task himself.

[1] http://www.emsstrategies.com/dd040107article.html

This would reduce the physical stress on the operator and he can concentrate on cognitive tasks of inspection and information logging. Based on the proposed solution, we would require a collaborative robot, a collaborative gripper, and a 2D vision system. Moreover, we would require a task allocation framework which decides the tasks performed by the agents in temporal sequence. These requirements leads to the formal definition of the overall objectives of the thesis.

1.2 Overall Objectives

We formally describe the overall objectives of the thesis which will serve as guiding principles for the forthcoming chapters.

O1. Design of safe, modular and adaptive architecture for collaborative robots which optimizes for the following performance metrics:

   (a) Process-centered : The architecture must attempt to minimize overall cycle time.

   (b) Human-centered : The architecture must be flexible i.e, the operators must not be constrained to perform a strict pre-defined sequence of actions but allow freedom to choose an action to perform *online* and the robot must be able to adapt *reactively*. It should also ensure ergonomy so that the operators avoid picking and placing of heavy parts.

O2. Incorporate a linguistic level of communication between operators and robot, such that the operators should be capable of intuitively understanding robot actions and intentions.

O3. Implement a vision-based perception using RGB information for task planning, motion planning and manipulation for collaborative robots.

O4. Adopt component based software engineering methodology that is testable, scalable and robot independent for an industrial setting.

O5. Study the trade-off between expressivity of employed formalisms and associated computational performance, as the system is expected to support human-robot collaboration in a real production line at runtime.

The rest of the thesis[1] is organized as follows: Chapter 2 describes the relevant state-of-the-art literature review; Chapter 3 explains the overall architecture

[1] **Disclaimer:** Throughout the thesis, while referring to the human operator we may use male pronouns for the ease of reading, however the arguments are gender-neutral and apply equally to all genders.

describing the necessary concepts to solve the problem; Subsequently, Chapter 4 explains the methodology for experimental implementation followed by Chapter 5 discussing the results and conclusions and future work in Chapter 6.

Chapter 2

Literature Review

Human-robot interaction (HRI) is a research field with wide ranging applications in households, elderly care, automobile industry, entertainment outlets and so on. In this thesis, we use human-robot interaction (HRI) and human-robot collaboration (HRC) interchangably. However, we must note the subtle difference in meaning between them. Interaction is more generic, which includes collaboration. Interaction determines action on someone else. Collaboration is working with someone on a common goal (Bauer *et al.* (2008)). Humans and robots collaborate on a common task form a team. All agents in the team must know the intentions of each other. Based on this, a robot can plan its own actions that will eventually lead to reaching the common goal. Therefore, the robot needs the abilities of perceiving and comprehending their environment, planning, decision making, and learning. In this chapter we present the literature review of major subtopics of HRI relevant to the field of research involved in this thesis: HRI in manufacturing industry; learning; architecture, planning and task allocation; physical human-robot interaction (pHRI); and human action recognition.

2.1 Collaborative Robots in Manufacturing

In this section we will discuss the general topics related to incorporating collaborative robots in manufacturing industries. A comprehensive article on HRI in manufacturing industries is provided in Krüger *et al.* (2009). The survey was conducted in 2009 when the state-of-art cooperation system was weight compensators/ balancers. However, the authors predicted the need for more advanced ergonomic tools for human robot cooperation in the shop-floor setting.

A more recent article by Faber *et al.* (2015) discusses the requirements for an ergonomic workplace where humans and robots perform tasks without a separa-

tion between their workplaces. Michalos *et al.* (2010) show that mass customization in automotive industries requires high technological flexibility. They proposed designs comprising of robot and human-based assemblies. In a recent EU project ROBO-PARTNER Michalos *et al.* (2014), the authors discuss the various levels of cooperation such as concurrent execution of tasks by the human and the robot, cognitive level cooperation and direct physical interaction. Direct physical interaction is defined as a study in which a robot, commonly in touch with a human collaborator, responds to the human's intentions and actively contributes to achieve the shared goal. This will be expanded in the following sections. They also present a case study for human-based assembly vs human-robot-based assembly and use comparison metrics such as number of tasks allocated for humans, number of tasks for robot, maximum weight handled by human, human working time, robot working time and so on.

	Current state	ROBO-PARTNER
No of Tasks allocated to human	20	15
No of Tasks allocated to Robot	0	7
Max weight handled by human (kg)	12.15	1.5
Human working time in cycle (sec)	96.3	65.2
Robot working tine in cycle (sec)	0	75
Total Cycle time (sec)	96.3	79

Figure 2.1: Metrics used for HRC [Michalos *et al.* (2014)]

It is interesting to note key points from figure 2.1. The current state is without usage of any robots and is completely done by humans. The number of tasks allocated to robot increases in the case of ROBO-PARTNER. Also, the weight handled by human significantly reduces. Finally, overall cycle time is also reduced thus improving the overall process efficiency.

Other aspects such as safety of a shared workspace is discussed in detail in Pedrocchi *et al.* (2013). Since the safety is provided by infrared sensors, the authors propose a two fold solutions: collision avoidance strategy allowing online replanning of robot motion and an obstacle-free subspace of the working environment. This paper extensively describes all the techniques that can be used for safety of robots and humans in HRC scenario exclusively using infra-red (IR) sensors. Methodologies for collision avoidance in literature include: Artificial potential fields, probabilistic methods and use of Danger Indices. Artificial potential fields (APF) although are very suitable for IRs, it does not guarantee stability in reaching the final position. Probabilistic methods are suitable incase of complex scenarios. Danger Indices (DI) which is the product of distance, velocity

and inertial factor of the robot and environment can also be used to generate sort of repulsive forces as in APF. In this paper, the authors develop a finite state machine as shown in figure 2.2. Finite State Machine (FSM) with three superstates: humans can be inside Safe, Warning, and Danger Areas. The system is initialized in Danger state; Warning and Safe states have nested FSMs for processing motion planning according to the conditions elaborated from sensors. The collision avoidance strategy takes place only in Warning state. The planning

Figure 2.2: Finite state machine [Pedrocchi *et al.* (2013)]

phase is divided into offline and online phases. The offline phase consists of modelling the static obstacles and safely reachable points in the workspace. During the online phase, humans are considered as dynamic obstacles. The algorithm calculates the distance to the dynamic obstacles and if its less than the minimum distance (d), the robot stops, more than maximum safe distance (D), the robot continues as usual and in between the collision avoidance techniques are used. The framework is designed to include redundancy in each phase. There are two independent pipelines of processing and the collision avoidance algorithm, kinematics and motor commands are checked with each other at every step.

Also interesting to note that even during replanning, the robot motion must be predictable for the humans as sudden and unpredictable behavior must result in scaring the human co-workers. This is an important psychological factor that needs to be taken care in the context of HRC. Further strategies on safety of robot co-workers is discussed in section 2.4.

2.2 Architecture, Planning and Task Allocation

In complex environments, a robot needs to collaborate with other agents (humans or other robots) to complete the given task. In collaboration scenario all agents have a shared goal. They can either have a shared plan or not, in the latter case

all the agents need to recognize the intentions of other agents. The task allocation problem is deciding which agent performs which task. As briefly stated in Section 1, collaboration entails a joint processing of human and robot actions. For seamless integration of the human and robot actions performed in temporal sequence, a reliable and efficient task planner is crucial. This is a motivation for detailed study on architecture, planning and task allocation.

Regarding task planning, we can distinguish between three major abstraction levels: *System level*, *Team level* and *Agent level*. This is depicted in Figure 2.3. At system level, teams of human and robot agents are formed and the overall sequence of tasks and resources are distributed amongst them. The system level is the highest level of abstraction. At team level, the human and robot must collaborate with each other to complete the given task. The outcomes of the task execution is communicated back to the system level planner. Finally, the agent level planner maps the team level planning to a level of abstraction above the hardware (robot sensors and actuators). It is responsible to decompose the semantic action commands from the team level planner to robot motion plans and manipulation commands.

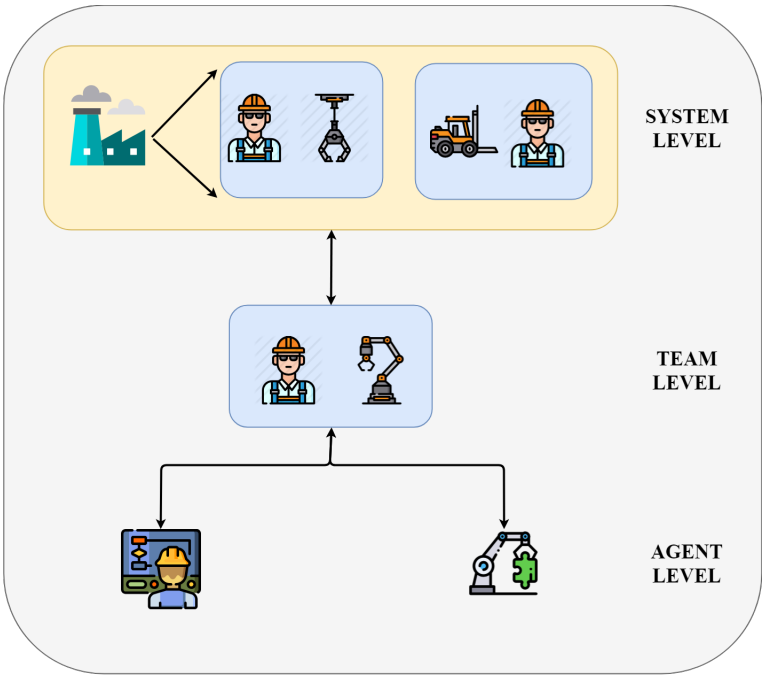


Figure 2.3: Various levels of abstraction of task planning

2.2.1 System Level Planning

In general there has been a lot of research in various fields such as vehicle routing, job-shop scheduling and multi-robot task allocation for optimizing the distribution of work and resources. We are mainly interested in the human-robot and multi-robot task allocation problems (MRTA). The authors Korsah *et al.* (2013) define the taxonomy of MRTA stating the dependence of the related tasks as follows:

- No dependence between tasks (each agents perform the tasks independently)
- In-schedule dependence (intra-task dependence)
- Cross-schedule dependence (inter-task dependence)
- Complex dependence (inter-task dependence)

Most of the literature handles this problem with approaches such as meta-heuristic based and market based approaches [Khamis *et al.* (2015)], Markov Decision Process (MDP), decentralised scheduling algorithms or distributed constraint optimization [Nunes *et al.* (2017)].

In a manufacturing scenario, we are mainly interested in optimizing the following performance criteria: minimal production changes, reduction in cycle time and/or improvement in ergonomy. The authors [Chen *et al.* (2011)] approach the problem by splitting the complex problem into simpler ones and allocating them to agents depending on their respective skills. The authors use a generalised stochastic Petri Nets to model the various situation during the assembly process and Monte Carlo methods to generate the finishing times and costs. These are optimized using a multi-objective optimizer to find the ideal task allocation for humans and robots.

However as noted in Khamis *et al.* (2015), robust solutions that handle complex tasks, highly dynamic task allocation are still underdeveloped.

2.2.2 Team Level Planning

Once the tasks are allocated and scheduled, the work must be distributed amongst the members of the teams. The authors Johannsmeier & Haddadin (2017) propose a multi-agent hierarchical human-robot team approach. They propose a

framework of three different architectural levels: *team-level assembly task planner*, *agent-level skill planning*, and the *skill execution level*. The tasks are modelled using an AND/OR graph. The team-level planner produces task sequences for every agent via A^* graph search. The agents in turn implement their modular skills via hierarchical and concurrent state machines in order to map abstract task descriptions to the subsequent real-time level. However, a noticeable feature in this paper is that only the agent level (real time action) is online while rest of the process is offline i.e., static architecture based on AND/OR graph. The authors also propose further improvements such as introducing other synchronization schemes such as time-scaling, e.g., the robot slows down in vicinity of humans.

Other techniques such as Semi-Markov Models are used in Rozo *et al.* (2016) where the robot learns to react to user actions and lead the task when necessary. Also Hawkins *et al.* (2014) propose a fixed-task duration AND/OR graph. Furthermore, the interaction can be improved if the preferences of the humans are known or can be inferred. These preferences can be known *a priori*, inferred through communication or learned through experience. The following subsections will explore each of these approaches.

2.2.2.1 Preferences known a priori

The authors Wilcox *et al.* (2013) devised an algorithm that takes as input a Simple Temporal Problem with Preferences (STPP), that encloses the variables, constraints, preferences and the optimization function. The outcome is a dispatchably optimal form of the STPP, where each executable event has assigned a time within the specified timebounds and the preference function is maximized.

2.2.2.2 Preferences learned through experience

Instead of coding the human preferences beforehand, they can also be learnt from experience. The authors [Agostini *et al.* (2011)] approached the problem by using a STRIPS-like planner that is constantly refined by a learner. It evaluates multiple cause-effect possibilities and chooses the one with highest possibility to occur. Similarly, in Koppula & Saxena (2016), the humans and robots are modelled using a Markov Decision Process to anticipate human behavior. The authors models the overall HRC task as a two-agent Markov decision game with the goal of completing the task.

2.2.2.3 Preferences acquired through communication

The task allocation can be modified online by acquiring the human preferences via speech, gestures or user interfaces. In Caccavale & Finzi (2017), the authors propose a hierarchical task planner which can be modified online through speech and gestures. These signals are fused to recognize the human intentions online and is used to replan actions.

Very recently, Darvish *et al.* (2017) proposed a novel improvement over the static architecture based AND/OR graphs. They aimed to enable novel and natural interaction between humans and robots in industries. Their main contributions are two fold: i) a framework for the representation of the cooperation task, which allows for run-time adaptation; ii) a dynamic procedure to monitor task execution based on AND/OR graphs. This allows the human to choose freely among a number of alternatives to perform a task. The preference of the humans to perform a certain action is acquired through gesture recognition via a smart-watch. This framework was validated using a human and a Baxter-dual arm robot performing turn taking actions in any allowed sequence.

The planning through acquiring preferences requires the recognition of human actions by the robot. Hence, this acts as a motivation for a deeper study in human action recognition in section 2.5.

2.2.3 Agent Level Planning

Agent level planning translates the incoming semantic action commands to robot trajectories and manipulation commands. The authors of Haigh & Veloso (1998) developed a planner that generates appropriate plans, delivers them to the robot, monitors their execution and learns from feedback regarding task performance. In Johannsmeier & Haddadin (2017), the authors designed the agent level planners as a set of *skills* such as "pickup part", "assemble part" and "hand-over" which can be further decomposed into *atomic actions*. These atomic actions are presented to the robot drivers to execute the action.

As noted by Pinto *et al.* (2017), one of the major open issues of task planning in HRC is that most tests are done on graduate students and scientists. There is a need to evaluate acceptance of planning software in real world scenario during more extended period to see if people would effectively use system in a daily basis. This issue is addressed in our thesis where we evaluate our architecture involving task planning in a real industrial scenario with shop-floor operators.

2.3 Learning

There are many facets of learning in HRI: learning of the planning problem, human model, new tasks and actions, teaching by demonstration and so on. Typically, a human expert is required to explicitly teach a particular skill or task to a robot and thus the robot learns by demonstration Atkeson & Schaal (1997). In this section we will concentrate on the learning of human model by the robot.

In a more recent and seminal article, Nikolaidis *et al.* (2014) proposed a novel method to automatically learn *human types* in an unsupervised fashion from demonstration of human teams. The author defines human types as the preference the human has for a subset of the task-related actions taken by the robot during a collaborative task. Clustering of human operators into "safe" or "efficient" is done using *Expectation Maximization* followed by using *Mixed Observability Markov Decision Process (MOMDP)* to compute the policy based on the human cluster 2.4. When a new human subject is introduced to execute the collaborative task with the robot, his type is inferred either offline from prior demonstrations or online during task execution. This strategy can be exploited in our scenario to tune the working of the robot according to the type of human co-worker.

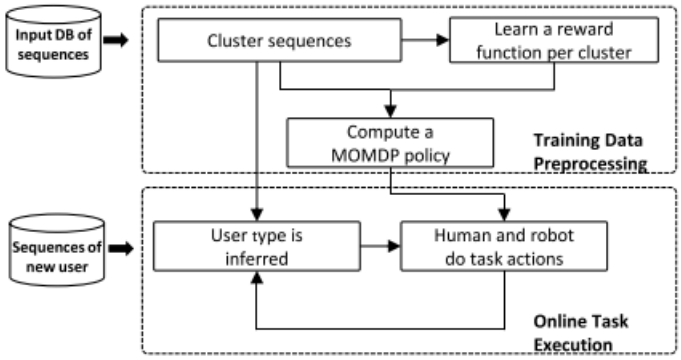


Figure 2.4: Human "type" recognition [Nikolaidis *et al.* (2014)]

Furthermore, anticipating human gestures enables the robot to plan ahead. In Koppula & Saxena (2016) the authors represent every possible future using an anticipatory temporal conditional random field (ATCRF) that models the rich spatial-temporal relations via object affordances. The authors show that anticipation by the robot can improve detection of past activities. For new human subjects (not in training set), they have a prediction accuracy of 84% for 1 sec-

ond anticipation time.

However in the above related works, it is assumed that the actions of humans is clearly and reliably observed. It is not the case in a real-world industrial environment with lot of obstacles and noisy disturbance sources leading to task and sensor uncertainty. Hawkins *et al.* (2014) attempt to solve this issue of task and sensor uncertainty by representing the task by an AND-OR tree structure from which a probabilistic graphical model is constructed. Inference methods for that model are derived that support a planning and execution system for the robot. This attempts to minimize a cost function based upon expected human idle time. To model the sensor inaccuracies, the authors use parameters that represent the confidence in the detector's accuracy and recall. They perform experiments in simulation and real-world scenario using a UR-10 Universal Robot, camera and Kinect RGB-D sensors. The authors attempt to find a correlation between reliable/ unreliable detectors with high/low confidence measurements in terms of human idle time. They find that having low confidence in the measurements in case of perturbations, the performance is better. This is due to the fact that as uncertainty is introduced in the system, the robot does not commit to any particular task until a later time. However, we must note certain open issues from this paper: target population of this experiment were expert users. The aim of HRC in shop-floor scenarios must be targeted towards non-expert users. Also very few trials were done.

2.4 Physical Interaction and Safety

Physical Human-Robot-Interaction (pHRI) is defined as a study in which a robot, commonly in touch with a human collaborator, responds to the human's intentions and actively contributes to achieve the shared goal. As there is physical contact between humans and robots, it is imperative to define safety and dependability metrics [De Santis *et al.* (2008)]. Regarding safety, there are a number of industrial standards established as shown in Figure 2.5.

- **ISO 12100 General principles for design:** specifies basic terminology, principles and a methodology for achieving safety in the design of machinery. It specifies principles of risk assessment and risk reduction to help designers in achieving this objective [ISO (2010)].
- **ISO 13849 Safety-related parts of control system:** provides safety requirements and guidance on the principles for the design and integration of safety-related parts of control systems (SRP/CS), including the design of software [Iso (2006)].

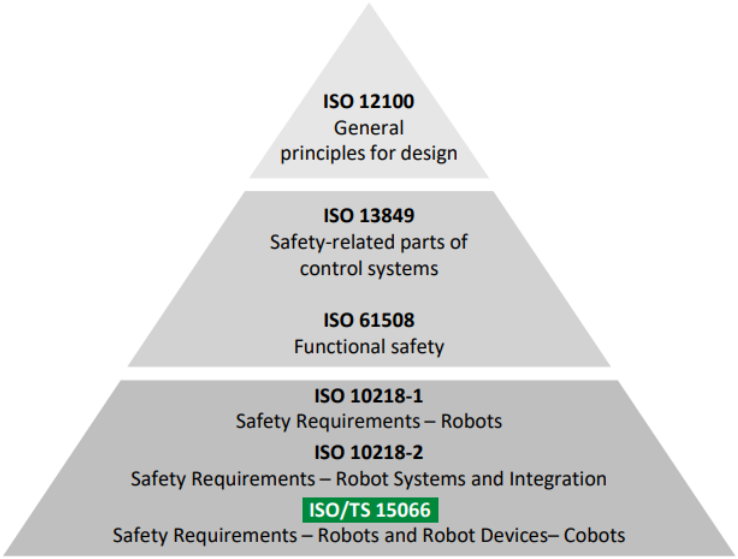


Figure 2.5: Industrial safety standards

- **ISO/IEC 61508 Functional Safety:** is a basic functional safety standard applicable to all kinds of industry. It defines functional safety as: "part of the overall safety relating to the EUC (Equipment Under Control) and the EUC control system which depends on the correct functioning of the Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related systems, other technology safety-related systems and external risk reduction facilities." [IEC (1998)]

- **ISO 10218-1 Safety Requirements- Robots:** introduces the concepts of collaborative operation, collaborative workspace, and collaborative robot. It includes other details such as start-up controls, functioning of the safety control system, motion braking, speed control [Iso (2011)].

- **ISO 10218-2 Safety Requirements- Robots Systems and Integration:** focusses on the above definitions and provides further details such as *collaborative operation requirements* and *cooperation task typologies*. It provides details on manual guidance, interface window and cooperative workspace [Iso (2011)].

- **ISO/TS 15066 Safety Requirements- Robots and Robot Devices-Cobots:** specifies safety requirements for collaborative industrial robot systems and the work environment, and supplements the requirements and

guidance on collaborative industrial robot operation given in ISO 102181 and ISO 102182 [15066 (2010)].

These standards serve as guiding principles for physical HRI research. The pHRI research is primarily motivated by the following considerations Argall & Billard (2010):

- **Safe operation around humans**. The robot possibly interacts with a human during behavior execution, perhaps unexpectedly (e.g. unintended collisions).
- **A necessary element of behavior execution.** The robot definitely, and necessarily, interacts with a human during behavior execution. The human might guide (e.g. indicate behavior selection) or be a partner in (e.g. humanrobot team tasks) the execution, or the humanrobot contact might be the entire point of the behavior (e.g. robot-assisted touch therapy).
- **A necessary element for behavior development.** The robot depends on tactile contact from a human while building, refining or adapting a behavior.

Broadly the literature on safety for pHRI can be classified into "Separating human and robot workspaces" and "Shared human-robot workspaces".

2.4.1 Separating Human and Robot Workspace

The robot behavior is modified directly when a human is detected in the robot workspace. When an intrusion into the robot workspace is detected the robot speed is reduced in proportion to the detected hazard level, with the robot stopping its movement at the highest one. This methodology is studied in Kittiampon & Sneckenberger (1985).

2.4.2 Shared Human and Robot Workspace

Using shared workspace, there are two approaches followed in literature: Minimizing injury by collision in HRC and Collision avoidance.

Minimizing injury during collision requires the use of specialized external hardware. Usually a combination of several mechanical compliance systems such as viscoelastic coverings and absorption elastic coverings are used. This is done in Lim & Tanie (1999) and Ulmen & Cutkosky (2010). Furthermore, the cobot manufacturers employ the strategy of using light-weight structures such as carbon-fibre or aluminum bodies to reduce the impact force during collision. A few

examples of such robots are the Kuka LBR IIWA (kuk (2017)), UR10 Universal Robots (url (2017)), Sawyer and Baxter from Rethink Robotics (ret (2017)). Moreover, force sensors can be used to detect collision and perform safety measures such as stop the robot motion as studied in detail by authors Magrini *et al.* (2015).

Finally, the collision avoidance strategy employs active replanning of robot motion to avoid any collision with the humans. This requires defining pre-collision strategies in software such as *Artificial Potential Fields* [Khatib (1986)], *Virtual Forces* [Tsuji & Kaneko (1999)], *Kinetostatic Danger Fields* [Lacevic & Rocco (2010)] and so on. The philosophy of the *Artificial Potential Field* is that the manipulator moves in a field of forces, where there are attractive poles for the end-effector (for example: the position to be achieved), and repulsive forces (obstacles to be avoided). Similarly, *Virtual Force* aims to push the robot away from the danger area. This virtual force is a function of the effective impedance at the closest point, and a function of the relative distance between the robot and the object. Whereas, in *Kinetostatic Danger Fields* the position and velocity of the robot arm are measured through proprio-ceptive sensors and are used to compute the danger field at any point of the workspace.

Additionally, for collision avoidance approaches, it is necessary to use specialized external hardware sensors and incorporate sensor fusion techniques to compute new robot trajectories. For instance, the sensors used include: motion-capture systems, advanced vision systems, range sensors and RBG cameras. The authors Robla-Gómez *et al.* (2017) detail the list of algorithms to incorporate external sensors for robot collision avoidance.

2.5 Human Action Recognition

For effective HRI, the human and robot must communicate in the *human semantic level*. In order to achieve the shared goal, the robot must be able to recognize the human's intentions and corresponding actions. Broadly, humans can communicate through explicit and implicit methods. In the Figure 2.6, the various ways of implicit and explicit methods of communication is shown. The implicit communication is shown in grey [Bauer *et al.* (2008)].

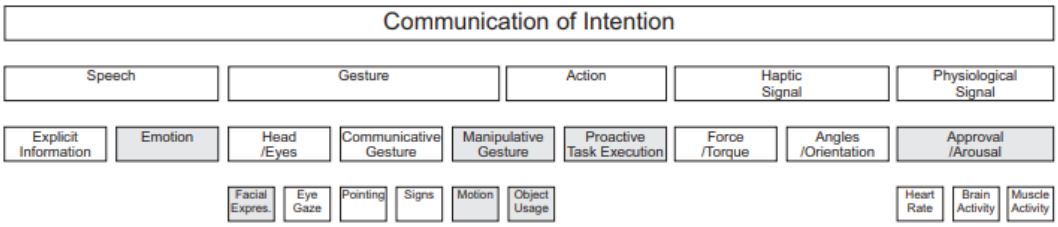


Figure 2.6: Various methods of communication
Bauer *et al.* (2008)

2.5.1 Explicit Communication

Using explicit communication an agent makes sure the peer receives certain information about his or her intention. Explicit communication can happen through speech, gesture, actions or haptics.

The authors Gleeson *et al.* (2013) present a comprehensive study of the communicative terms and **Gestures** used in industry. The main actions necessary for industrial assembly task can be subdivided into: part acquisition, part manipulation, and part operations. The authors implemented these gestures on a robot arm in an HRI context, and found that the human observers could interpret the gestures with high accuracy. Similarly, human's intentions can also be inferred through head gestures such as eye gaze directions and facial recognition. This is explored by Sakita *et al.* (2004) and Kanade *et al.* (2000). The human's eye gaze can convey what the human is focusing on and allows the robot to understand the intention of the humans. Similarly, the facial expression exposes the moods of the human and allows the robot to operate more socially.

Speech Recognition accounts for a more natural and human-like interaction between humans and robots. The important approaches to speech recognition has been done using semantic networks [Woods (1991)] and conceptual dependency [Schank (1972)]. A semantic network, or frame network is a knowledge base that represents semantic relations between concepts in a network. Conceptual dependency theory is a model of natural language understanding used in artificial intelligence systems. Additional details regarding the semantic networks and conceptual dependency theory is out of the scope of the thesis.

Finally, we consider **Haptic Communication** wherein the robot infers the

human intentions and actions by measuring the direction of force and torque applied by the human agent. Besides force and torque, angles and orientation can be used as information in haptic communication. These are researching in the works of Kosuge & Kazamura (1997).

2.5.2 Implicit Communication

In general, **manipulative gestures** are communicated implicitly by the human while performing a given task. For this, these gestures include handing over a part, moving towards an object, and so on. "Movements are defined as motions whose execution is consistent and easily characterized by a definite space-time trajectory. An activity is defined as a statistical sequence of movements, its recognition requires knowledge about both the appearance of each constituent movement and the statistical properties of the temporal sequence. For a system to recognize actions it must include a rich knowledge base about the domain and be able to hypothesize and evaluate possible semantic descriptions of the observed motion [Bauer *et al.* (2008)]". In most cases, the action recognition is done using vision systems by finding patterns in the changes of pixels [Bobick (1997)].

Similarly, human action recognition in an industrial environment was studied by Roitberg *et al.* (2014). Their study is based on spatial and temporal features derived from skeletal data from human workers performing assembly tasks. They considered the following three groups of activities: Movement, Gestures, and Object handling. The features were used to train a machine learning network, classifying discrete time frames with Random Forests and models the temporal dependencies using a Hidden Markov Model. The authors claim to achieve 73% overall accuracy in classifying the activities. Two synchronized RGB-D Kinect sensors were used to obtain the skeletal joint data of the humans. However, since its a classification problem involving temporal data, Recurrent Neural Networks (RNN) could also be used, which has not been considered by the authors. Also, their analysis is based on a fairly small dataset (24 recordings) using leave-one-out cross validation. The authors also suggest that gaze tracking could be a viable future improvement, which directly gives information of the human's intentions.

Based on the discussion regarding the state-of-the-art in human-robot collaboration, the next chapter will discuss in detail the overall architecture that will be developed in this thesis.

Chapter 3

Overall Architecture

This chapter will define the terms and explain the concepts involved in designing the overall architecture. The architecture is designed in a way to achieve our predefined objectives O1.-O5. The overall cooperation task is based on a modular, reactive architecture which is conceptually described in Figure 3.1. The important modules of the architecture are the *Vision System* (includes the *Human-Action Recognition*), *Task Representation*, *Task Planner* and *Robot Interface*. The architecture reliably handles messages between modules and the interaction between them is necessary for the overall operation.

The layered framework serves as a way of abstracting the complexity at the lowest level of execution so that the user can focus on what to do, namely the process, by creating a *Task Representation*. The *Task Representation* and *Planner* are chiefly derived from the FlexHRC system developed by Darvish *et al.* (2017) in the University of Genova, Italy. *Task Representation* maintains a set of models for the cooperation tasks to be carried out, and decides which action an operator or robot must perform next by interacting with the *Planner*. Cooperation models are represented using AND/OR graphs, as described subsequently. The *Planner* operates on the graph via an ad hoc online graph traversal procedure to determine the most appropriate sequence of actions to ground the cooperation on, based on the graph structure. As discussed later, these action sequences are encoded within graph edges, referred to as hyper-arcs.

To generate the next most optimal action, the *Planner* must be informed when the previous action is completed successfully or unsuccessfully by the *Human Action Recognition* module (for human actions) or *Robot Interface* (for robot actions). However, there is an important difference between the two loops involving the *Planner* and *Human Action Recognition* and *Robot Interface*, respectively. In the first case, the *Planner* just suggests the next action to operators, leaving them

free to execute it or not, therefore taking into account Objective O1., whereas in the second case it imposes the next action for the robot to perform. In order to achieve Objective O2., the *Planner* presents the actions to human agents via a simple and effective User Interface. The *Vision System* is responsible for autonomously checking if the human actions are performed successfully or not and additionally, providing feedback of object pose to the *Robot Interface* thereby achieving Objective O3.. Furthermore, the *Planner* does not have access to low level robot trajectory planning and internal parameters in the *Robot Interface*, as we employ a modular component based software architecture using the *Robot Operating System (ROS)* as our middle-ware (Objective O4.).

Nonetheless, we must re-iterate that the architecture assumes the human agents are not trying to *cheat* the system. Although the operators are given the flexibility to perform an action or not, the system cannot recognize if the operator performs an action completely unrelated to the overall cooperation goal. Admittedly the system is not completely fool-proof, principally due to lack of advanced sensors and actuators.

The chapter is designed in logical order based on the architecture as follows: Section 3.1 explains the AND/OR graphs structures used for *Task representation*; Section 3.2 develops on the concepts involved for the *Planner*; Section 3.3 defines the approaches for the *Vision System* i.e., object pose recognition and robot-vision guidance techniques; finally Section 3.4 explores the concepts of *Robot Interface* namely: robotic manipulation and robot motion planning.

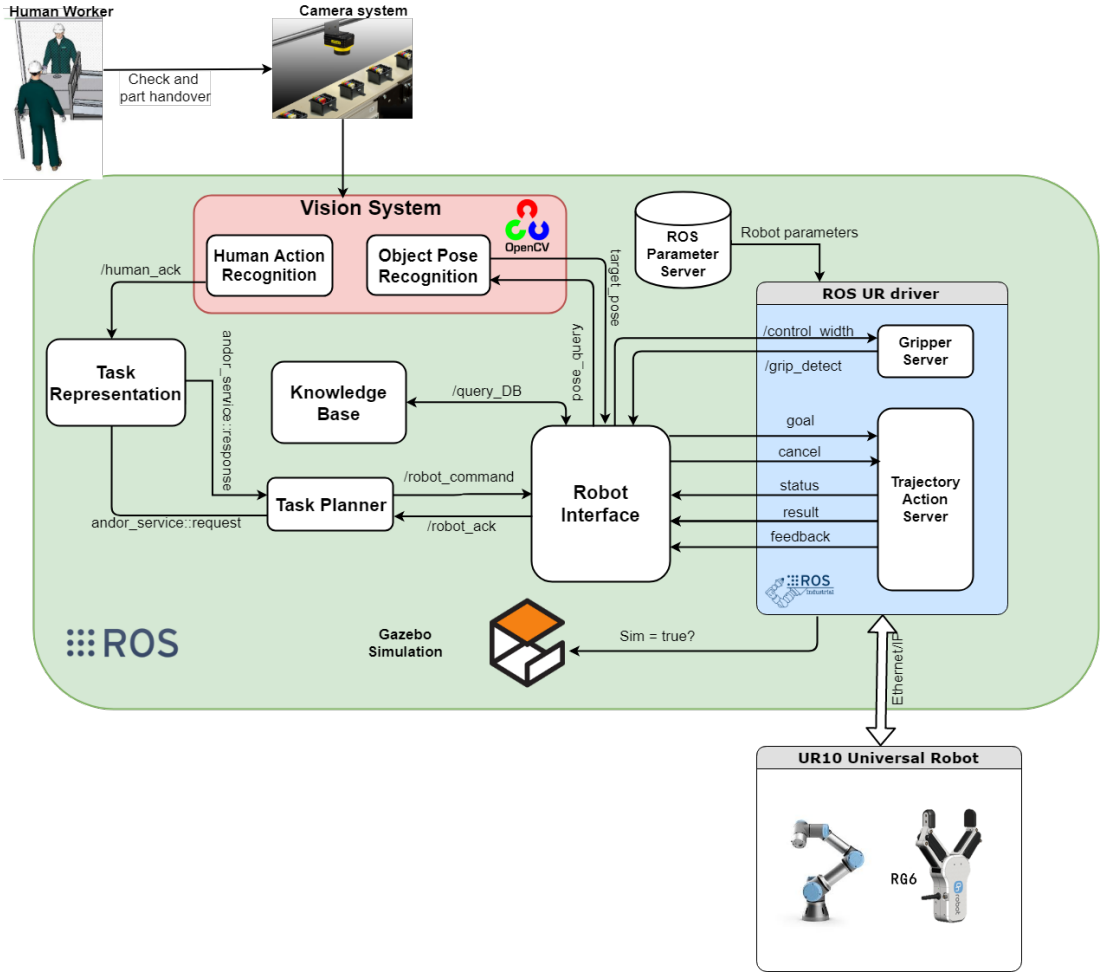


Figure 3.1: Graphical description of the overall architecture

3.1 Task Representation

The *Task Representation* module encodes the co-operation task as an AND/OR graph. The AND/OR graph was first defined by De Mello & Sanderson (1990). An AND/OR graph is a compact representation of all possible assembly plans of a given assembly product. It is equivalent to a state transition graph but requires fewer nodes and simplifies the search for feasible plans. Unlike a generic directed graph[1], the AND/OR graph can perform parallel execution of assembly operations and show the time dependence of operations that can be executed in parallel. It is ideally suited for parallelized multi-agent assembly.

[1] https://en.wikipedia.org/wiki/Directed_graph

Deriving from the work of Darvish *et al.* (2017), we define the AND/OR graph $G(N, H)$ as a data structure where N is a set of $n_1, \ldots, n_{|N|}$ nodes and H is a set of $h_1, \ldots, h_{|H|}$ hyper-arcs. Nodes in N define reachable states, whereas hyper-arcs in H define transition relationships among states. Nodes follow the classical definitions of tree structures: *Root node* is the node at the top of the graph, *child nodes* are nodes directly connected to another node when moving away from the root, whereas *parent nodes* is the converse notion of the child. The relations between transitions of a hyper-arc are in **logical *and***, while the relationships between different hyper-arcs are in **logical *or***. For instance in Figure 3.2, h_1 hyper-arc defines an AND relationship between nodes n_1 and n_2 such that *both* n_1 and n_2 needs to be satisfied to reach n_r. In contrast, hyper-arcs h_2 and h_3 define an OR relationship such that to reach node n_1, either the pair n_3 and n_4 (via h_2) or alternatively node n_5 must be reached first. Both the nodes and hyper-arcs are associated with costs. Additionally, hyper-arcs define *many-to-one* transition relationships between *many* child nodes and *one* parent node.

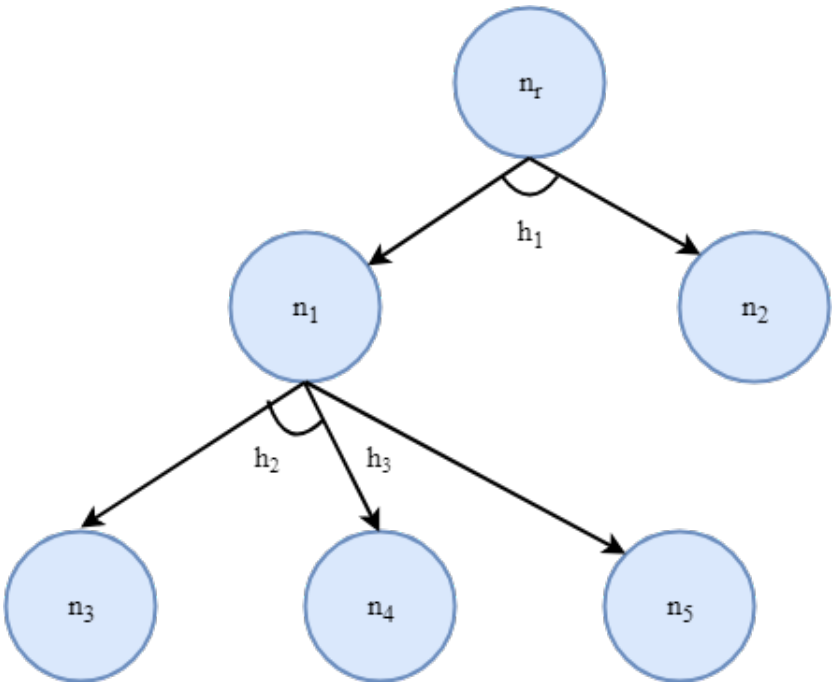


Figure 3.2: AND/OR graph 6 nodes and 3 hyper-arcs: h_1 and h_2 are *and* hyper-arcs and h_3 is *or* hyper-arc

Each hyper-arc h_i models a set of actions A_i, where an action $a_j \in A_i$ can be performed either by a human or a robot during the cooperation process. An

important **assumption** is that the responsible agent for a particular action is pre-defined based on the capabilities of the agents. For instance, say an action *Hand-over* which is handing over a part is the responsibility of the human agent while *Grasp* action is assigned to the robot agent. If the order in which to execute actions in A_i is important, A_i is defined as an ordered set such that $A_i = (a_1, \ldots, a_{|Ai|}$; $\preceq)$, i.e., a *temporal sequence* is assumed in the form $a_1 \preceq a_2 \preceq \cdots \preceq a_{|Ai|}$. Initially, all the actions in A_i are labelled as unfinished; when they are executed successfully we label them as finished. If all the actions in A_i are finished, then h_i is done. In addition, if an ordering is induced, the hyper-arc h_i holds if and only if the temporal execution sequence is satisfied.

Nodes can be either solved or unsolved. A node $n_k \in N$ is solved if there is at least one hyper-arc h_i to this node, h_i is done and all its child nodes are solved. The *leaf nodes*, which are nodes without any child nodes, in the AND/OR graph G are initialized as solved or unsolved at the beginning, depending on the initial state of the cooperation. This procedure iterates going upward to the root node of G. When the root is solved, then G is labelled as solved. During graph traversal, n_k is feasible if there is at least one h_i to it such that all its child nodes are solved. Otherwise, n_k is *unfeasible* and h_i is labelled as active. At all times, we have a set of active hyper-arcs $H_a \subset H$ in G.

The temporal task representation state S is the set of all the feasible nodes and active hyper-arcs in G. S defines the possible action alternatives for the human or the robot in cooperation. We define as *cooperation context* a sequence of actions performed by humans or robots during the cooperation, corresponding to an allowed traversal path in G. Each cooperation path is associated with a traversal cost which defines how effortful following the path is, on the basis of the involved node and hyper-arc weights. Depending on the optimal cooperation path, i.e., the one minimizing the overall cost depending on node and hyper-arc weights, the robot may start moving or waiting for operator actions.

When a new human action a_n is detected by the *Human Action Recognition*, the *Planner* interacts with *Task representation* to determine whether a_n belongs to A_i such that the latter corresponds to an hyper-arc in H_a. If this does not happen, i.e., a_n does not belong to the current cooperation context, the robot enters a null mode and waits for further knowledge to determine which traversal path in G is involved. If there are multiple active hyper-arcs possibly involving a_n, the robot enters an ambiguous mode. Otherwise, the next action to suggest or perform is defined in G so that the overall cost of the traversal path is minimized. In the case of robot action however, it is simpler as they are pre-defined with only two possible outcomes *success* or *failure*. In case the assigned robot

action is returned successful by the *Planner*, the AND/OR graph updates the graph and provides the next action in the current active hyper-arc h_i. If the robot action outcome is a failure, the current hyper-arc is set as *unfeasible* and the next optimal hyper-arc is chosen. Therefore, the most optimal route to the root node is chosen at each stage *online* accounting for any changes in actions by the humans and success/failure of robots.

In the next section, we will discuss how the *nodes* and *hyper-arcs* are updated by the *Planner* based on feedback from *Human Action Recognition* and *Robot Interface*.

3.2 Task Planning

The *Task Planner* is a *Sequential planner*, i.e. the plan is a totally ordered sequence of actions and the optimal plan is the shortest one (one with least cost). Once the graph S is updated in the *Task Representation module*, the *Planner* decides the next node to be solved in the current cooperation path. Graphically the working on the *Planner* is shown in Figure 3.2.

Given the last solved node n, the *Planner* checks if it is the root node. If yes, the graph G is labelled *solved* and the cooperation task is complete. If not, the planner proceeds into a loop until the root node is reached. The set of all paths to update is determined as those containing the last solved node n. For each path P, its associated cost is updated as:

$$cost(P) = cost(P) - (w_n + h_n^m - w_h) \tag{3.1}$$

where w_n is the weight associated with n, h_n^m is the maximum weight of the hyper-arcs connecting any parent node to n, and w_h is the weight of the hyperarc connecting any parent node to n in P.

The optimal cooperation path P^* is determined, such that it is characterized by the minimum cost from Equation 3.1. Then, for all nodes in P^*, the first node n is found such that it is *feasible* and *not solved*, which is labeled as n^*. On the basis of n^*, the actions to expect from the operator or to be executed by the robot are determined, and the procedure continues.

When the *Planner* module receives the hyper-arcs from the *Task Representation* module, it uses the associated information (in terms of feasible states, their weights and the associated actions) to create an *Action-State table*, which is used to determine the next action for a robot or an operator to perform. For

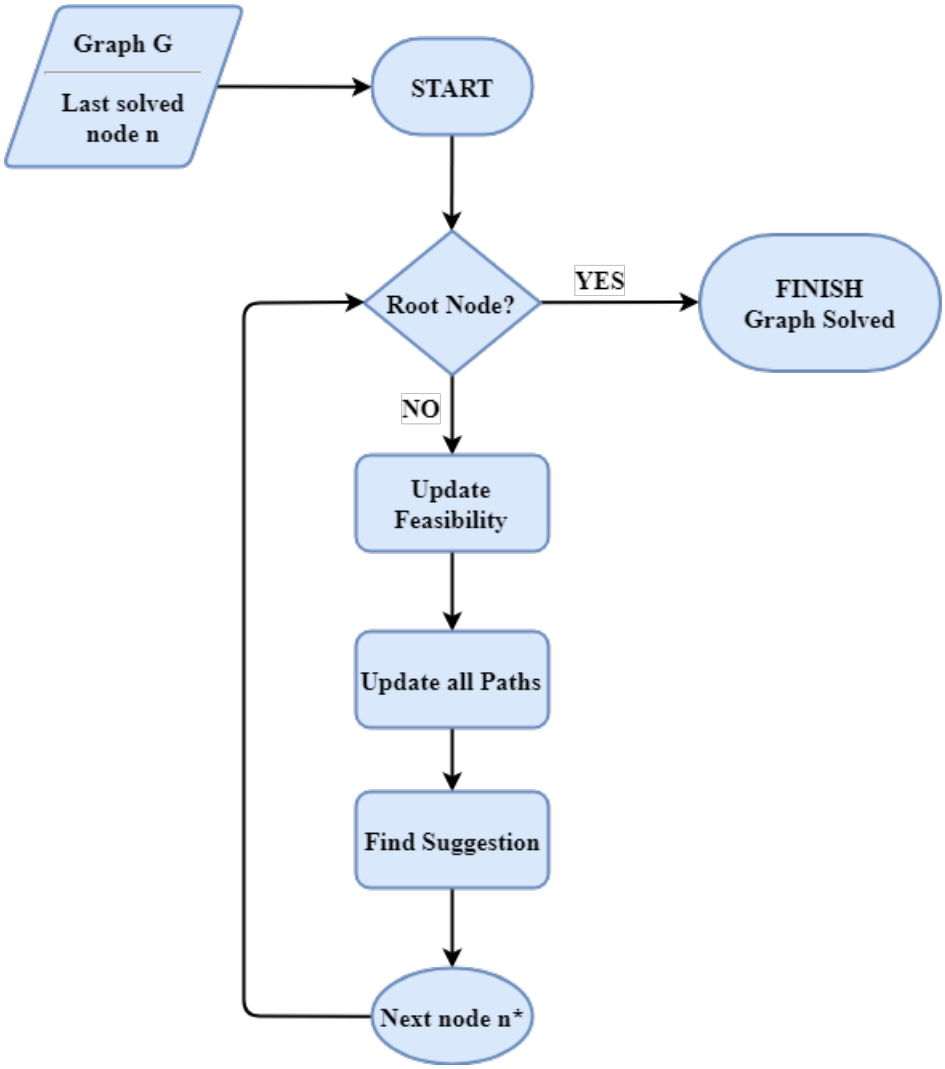


Figure 3.3: Flowchart of Planner module

each feasible state, the table contains the action sequences which, when executed successfully, can lead to that state. Each feasible state $n_i \in N$ in the table is associated with its weight w_{ni}, and a (possibly sequentially ordered) set of actions A_i. Each action $a_j \in A_i$ is associated with a responsible agent, namely the human operator or the robot. For each action the responsible agent is predefined. Feasible states are ranked on the basis of their weight, such that the state with lowest weight has highest priority.

During the cooperation process when an *acknowledgement* is received from the *Robot Interface* module, the next action in the sequence is evaluated and

submitted to the *Controller*. If the action needs to be performed by the human operator, the *Planner* sends a message to the *User Interface* asking the operator to perform the task. When an acknowledgement is received from the *Human Action Recognition* module, a check is done first to determine whether the operator is still following the same cooperation path.

We will look at the various states in the *Action-State table* in more detail with Figure 3.4 from Darvish *et al.* (2017). Each row shows a sequence of actions that the human or the robot must perform to execute the related feasible state. If the perceived sequence of actions is a subset of a sequence of actions of a feasible state the corresponding row is active, or otherwise it is inactive. In the beginning, as the perceived set of actions are empty, all rows are active. Depending on the number of active rows (more than one, equal to one or zero) the search is in *ambiguous*, *clear* or *null mode*. In the *ambiguous mode*, the following state is the active row with minimum weight. In the *null mode*, no row is active (the perceived sequence of actions is none of the admissible ones) and the cooperation is labelled failed. In the *clear mode*, there is only one row active and the human or robot follow the expected sequence. In the example of Figure 3.4, we assume that after the robot has performed action a_1, the human has performed action a_2. At this point the search is in ambiguous mode, since more than one feasible node (i.e., n_1, n_2, n_3, n_4) are reachable, the robot will therefore follow the row with minimum weight (the one related to n_1, which has weight $w_{n1} = 3$) and perform actions a_3 and a_4. Once action a_4 is finished the search is in clear mode and node n_1 is reached. Let us instead assume than after action a_2, the human performs action a_8. In this case the search enters null mode and the cooperation is failed. Lastly, let us assume that while the robot is performing action a_3, the human interrupts it and performs action a_6. Upon the recognition of a_6, only the third and forth rows of the table remain active and since the weight of state n_3 is the lowest, the robot switches to performing action a_4 and thus reach state n_3.

Concluding the *Planner* section, the following section on *Vision System* performs the *Human Action Recognition* to provide the feedback to the *Planner* when the designated human action has been completed successfully.

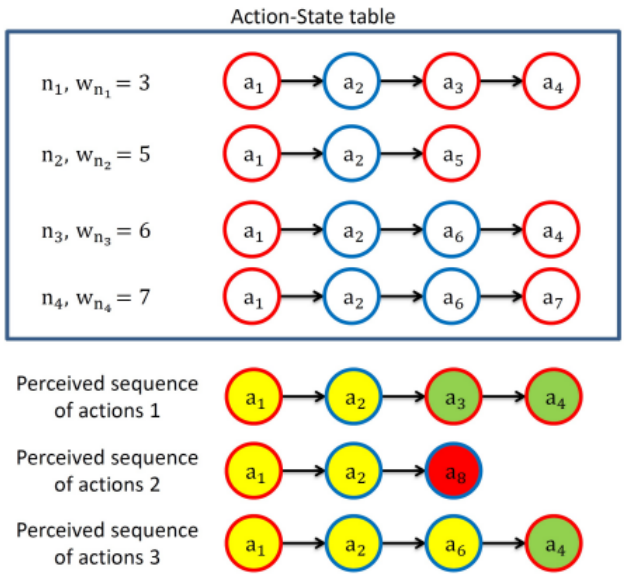


Figure 3.4: State-Action table search and update example: red circles denote robot actions; blue circles denote human actions. Yellow, red and green filling colors denote ambiguous, null and clear mode of the table search respectively. $a_1 - a_8$ are actions labels; $n_1 - n_4$ are feasible nodes labels;and $w_{n1} - w_{n4}$ denote the weight of each state [Darvish *et al.* (2017)]

3.3 Vision System

In this section we will discuss the techniques for recognizing and estimating the pose of rigid objects in an industrial setting. In essence, *object recognition* is the task of recognizing whether a particular object is present in an image and *pose estimation* is the task to precisely locate the object with a position and a rotation with respect to a pre-defined coordinate system.

3.3.1 Machine Vision

According to the Automated Imaging Association (AIA), machine vision encompasses all industrial and non-industrial applications in which a combination of hardware and software provide operational guidance to devices in the execution of their functions based on the capture and processing of images [1]. Industrial machine vision requires high robustness, high reliability, acceptable accuracy, moderate cost and high mechanical and temperature stability.

[1] https://www.cognex.com/what-is/machine-vision/what-is-machine-vision

Typically in industrial robotics applications, there are two broad categories of computer vision applications: 2D and 3D imaging. 2D imaging uses a single camera to create an image in grey-scale or colour. The major applications of 2D vision include: online Character Recognition (OCR), 1D/2D code reading, checking of labels and package, quality inspection, meteorology, Robot Guidance and so on. This can be seen from Figure 3.5.

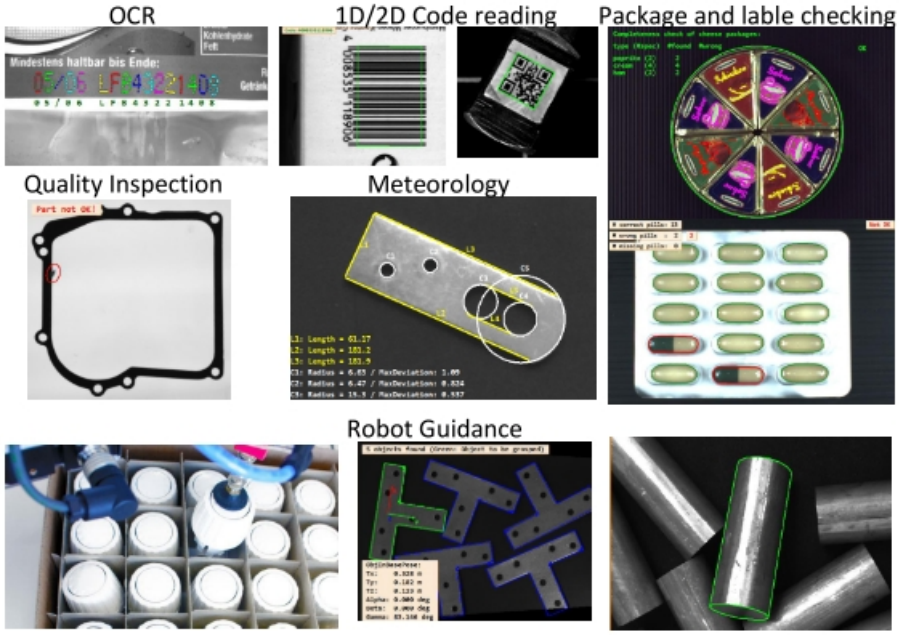


Figure 3.5: 2D machine vision applications [Sølund & Aanæs (2017)]

In comparison, a 3D vision system uses a variety of hardware (such as multiple cameras or camera with depth sensor) to create a 3D image of the target object which comprises of the pixel (x,y, intensity) and depth information. In this thesis, the task dictates object pose detection and manipulation placed on a plane. Hence we will concentrate on 2D vision technologies henceforth.

Commercial machine vision systems can be broadly divided into: vision sensors, smart cameras and vision systems. Vision sensors are imaging devices with pre-defined dedicated purposes such as online character recognition (OCR) or code reading. These sensors are not expensive, easily configurable and deployable by operators. Integration of the device in a production line is more or less Plug-n-Play. A few examples of these kind of sensors are SICK Lector, Omron and Banner P4.

For more functionality, smart cameras can be used. Smart cameras are scriptable cameras that allow an application engineer to customize a vision algorithm for a special purpose by combining predefined functions such as finding shape models (2D pattern recognition), blobs and different methodology tool etc. They are often marketed as general purpose devices with some limited functionality. However, these smart cameras still requires technical skills and know-how for deployment. Often accessories like Human Machine Interfaces are available that allows to show results using simple displays. It requires some knowledge from the technician to integrate smart cameras in a robotic application. In both vision sensors and smart cameras data processing is running on-board, without the need for external computing devices. This enables fast integration with a Programmable Logic Controllers (PLC) or Robotic system. Many of the manufactures of smart cameras delivers functional blocks for different PLCs, for example Siemens or Beckhoff which make the integration of the camera even easier. Issues like reliable detection under different lightning condition, calibration and coordinate transformations are challenges which require technically skilled operators.

When vision sensors and smart cameras are not providing the required flexibility and functionality for a given application, vision systems are the best solutions. Vision systems include industrial cameras, a computational unit such as an industrial PC, machine vision external lighting and one of the comprehensive professional image processing libraries like Halcon [1], Cognex [2] , Matrox[3], Scorpion [4] etc. With a vision system, a computer vision engineer can customize a vision solution for a given application. The development of a vision system for a factory automation application requires in-depth knowledge of programming, computer vision, cameras and lighting technology but provides the full flexibility.

As we will see in Section 4.1.3, we will use the **Cognex Vision System** for our experiments. These self-contained, industrial-grade vision systems combine a library of advanced vision tools, in particular the **Cognex Vision Library (CVL)** with high-speed image acquisition and processing.

3.3.2 Camera Calibration

Camera calibration is the process of finding the intrinsic parameters and/or the extrinsic parameters of the camera. This is a crucial process for: (1) finding the location of the object or scene; (2) finding the orientation and location of the

[1] http://www.halcon.com/
[2] http://www.cognex.com/
[3] http://www.matrox.com/imaging/en/
[4] http://www.scorpionvision.com/

camera; and (3) reconstructing a 3D image of an object or scene [Tsai (1987)].

A camera has two sets of parameters, *intrinsic parameters* which describe the internal properties of the camera, and *extrinsic parameters* which describe the location and orientation of the camera with respect to some coordinate system. We will utilise a pinhole camera model and it relates the 3D world points to 2D image projections. While this is not a perfect model of cameras used in machine vision systems, it gives a very good approximation and when lens distortion is taken into account, the model is sufficient for the most common machine vision applications. We can model a pinhole camera as in Equation 3.2:

$$x = K[R|t]X \tag{3.2}$$

Where X is the 3D point coordinates and x the image projection of X. (R, t) are the *extrinsic parameters* where R is the 3×3 rotation matrix and t the 1×3 translation vector. K is the *camera intrinsic matrix* and it describes the intrinsic parameters of cameras as in Equation 3.3:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where (f_x, f_y) are the focal lengths and (c_x, c_y) the coordinates of the principal point along the x-axis and y-axis respectively. The principal point is the point at which the light that passes through the image is perpendicular to the image, and this is often, but not always, at the centre of an image. The Figure 3.6 shows the model in detail, where P is the Principal point, f is focal length, C is the camera centre, X_{cam} is the camera point in the 3D space while x_{im} is its image point i.e, 2D projection on the image plane.

As mentioned previously, lens distortion needs to be taken into account for pinhole camera models. Two types of distortion exist, which are *radial* and *tangential*. An infinite series is required to model the two types of distortions, however, it has been shown that tangential distortions can often be ignored, in particular for machine vision application. It is often best to limit the number of terms for the distortion coefficient for radial distortion for stability reasons Tsai (1987). Below is an example of modelling lens distortion using two distortion coefficients for both tangential and radial distortions [Xie *et al.* (2008)]:

$$\tilde{x} = x + x[k_1 r^2 + k_2 r^4] + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.4}$$

$$\tilde{y} = y + y[k_1 r^2 + k_2 r^4] + [2p_1 xy + p_2(r^2 + 2y^2)] \tag{3.5}$$

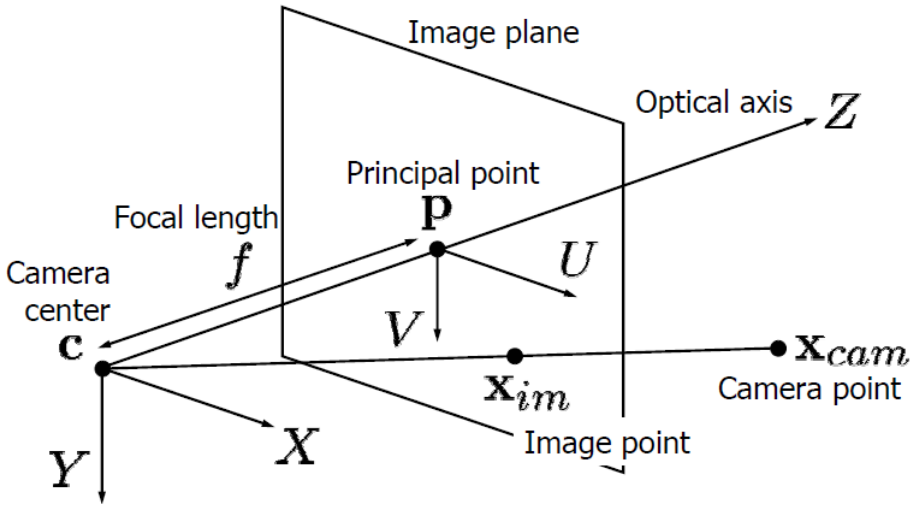


Figure 3.6: Pin-hole camera model [Martinet (2016)]

$$r = \sqrt{x^2 + y^2} \tag{3.6}$$

where (x, y) are the ideal image coordinates and $(\tilde{x}, \tilde{y})$ are the real (distorted) image coordinates.

In the Cognex system as we will see later, the lens distortion is corrected with the following equation:

$$\tilde{r} = (t[0] \times r + t[1] \times c + t[2])/(1 + t[6] \times r + t[7] \times c) \tag{3.7}$$

$$\tilde{c} = (t[3] \times r + t[4] \times c + t[5])/(1 + t[6] \times r + t[7] \times c) \tag{3.8}$$

Where r,c are the pixel coordinate point and $\tilde{r}$, $\tilde{c}$ is the world coordinate point.

For each point pair, we get the 2 equations above. 4 point pairs give 8 equations with 8 unknown coefficients. The equations are solved using QR factorization which gives a linear least squares solution in the over-determined case (more than 4 point pairs) and a least norm solution for the under-determined case (less than 4 point pairs). The terms $t[0], t[1], \ldots t[7]$ are the transformation coefficients that are solved for when performing the QR factorization[1].

[1] http://help.cognex.com

3.3.3 Object Pose Recognition

Object recognition is used in many computer vision applications. It is particularly useful for industrial inspection tasks, where an image of an object must be aligned with a model of the object. The transformation or pose obtained by the object recognition process can be used for various tasks, for example pick and place operations or quality control. In most cases, the model of the object is generated from an image of the object. This 2D approach is taken as it usually is too costly or time consuming to create a more complicated model, such as a 3D CAD model. Therefore, in industrial inspection tasks it is usually aligned towards matching a 2D model of an object to the image. The object may be transformed by a certain class of transformations, such as rigid transformations, similarity transformations, or general 2D affine transformations. The latter are usually taken as an approximation to the true perspective transformations an object may undergo [Ulrich & Steger (2002)].

3.3.3.1 Convention

The pixel coordinate system used is as per the Cognex Vision Library (CVL) conventions as shown in figure 3.7. The origion (x_0, y_0) is located on the top left corner of the frame, with horizontal x-axis and vertical y-axis. The scale is relative to pixel resolution, but as mentioned in the CVL documentation [1], Cognex alogrithms works at sub-pixel accuracy, meaning the pixel coordinates are specified as floating-point doubles instead of integers. This is done so that round-off errors does not propagate throughout the computation.

3.3.3.2 Pattern recognition: PatMax® tool

PatMax® is a geometry-based pattern recognition tool developed by Cognex. It was originally patented in 1997 as the first high-accuracy, high-speed, high-yield, object location technology for machine vision. The geometry-based vision algorithms looks for relations between edges in an image. An edge is defined as the boundary between regions of dissimilar grey-scale vales.

PatMax® is pattern-based search technology that differs from other pattern-location technologies in that it is not based on pixel grid representations that cannot be efficiently and accurately rotated or scaled. Rather, it uses tables of feature-based representations that can be transformed quickly and accurately for pattern matching. It closely resembles the way humans distinguish between similar things and recognize the familiar ones. The features of a model together form

[1] https://support.cognex.com/en/downloads/in-sight

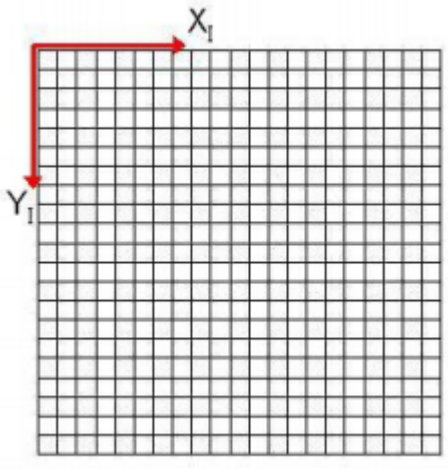


Figure 3.7: Origin, x- and y-axis of the pixel coordinate frame

a pattern. The pattern is described not only by the properties of each feature but also by the positions of the features in relation to each other. PatMax® identifies likely candidates within the specified degrees of freedom, then finds the transformation that best describes the change between model and search instance. To speed up the search, a coarse-to-fine approach is implemented. It initially uses large features to identify all candidates in the search image, followed by smaller features to adjust the pattern perfectly.

We will describe the possible steps used in PatMax®, however it must be noted that this is not meant to be a complete description of the patented technology, but instead an exploration of the underlying concepts based on what is known about PatMax®.

Filtering low- and high-pass

As defined earlier, edges are regions with dissimilar grey-scale values. But edges can be subjective i.e., other elements in an image can be taken as edges such as clutter and noise. To get rid of such ambiguous features, the image is subjected to a low-pass filter thus smoothing the images. The concept of image filtering can be explained through a 5×5 filter kernel as in Equation 3.9. The kernel is placed above a pixel point. Add the 25 points below the kernel, take the average and replace the central pixel with the new value. This is done for every pixel in

the image.

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.9}$$

High-pass filters help in finding edges. But in general, other complex and effective algorithms such as the Canny edge detectors are used for finding edges.

Canny Edge Detection

The Canny Edge Detection algorithm was developed by John F. Canny in 1986. It comprises of 4 major steps.

- **Noise Reduction:**

  As explained earlier, the first step is removing noise by using a 5×5 Gaussian filter.

- **Finding Intensity Gradient of the Image:**

  The smoothed image is then filtered using a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). The x-direction kernel detects horizontal edges, and y-direction kernels detects vertical edges. From these two images, edge gradient and direction for each pixel can be found as follows.

$$\text{Edge\_Gradient(G)} = \sqrt{\text{G}_\text{x}^2 + \text{G}_\text{y}^2} \tag{3.10}$$

$$\text{Angle}(\theta) = \tan^{-1}\frac{\text{G}_\text{y}}{\text{G}_\text{x}} \tag{3.11}$$

  where $G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$ and $G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

  The direction of gradient is always perpendicular to the edge. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

- **Non-maximum Suppression:**

  The next stage is removal of unwanted pixels which may not constitute edges. At every pixel, pixel is checked if it is a local maximum in its

neighborhood in the direction of gradient. This can be explained in Figure 3.8. Consider the point A which is on the edge direction. Points B and C are in the gradient direction. Point A is checked for local maximum amongst A,B and C. If yes, it is considered for next stage or else its suppressed.

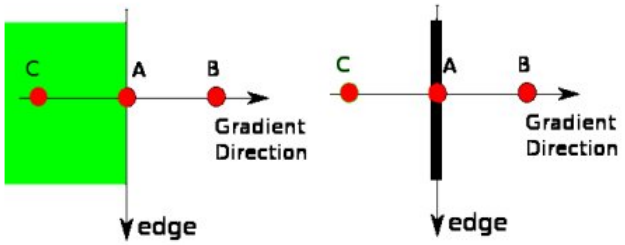


Figure 3.8: Non-maximum suppression [1]

- **Hysteresis Thresholding:**

  This stage decides whether an edge is really an edge or not. Two threshold values `minVal` and `maxVal` are chosen. If an edge lies below `minVal`, it is surely a non-edge. If it lies above the `maxVal`, its is a sure-edge. If it lies between the thresholds, the connectivity is checked. If the edge is connected to a sure-edge, such as C in the figure 3.9, it is retained. If its not connected to any sure-edges, such as B, it is suppressed.

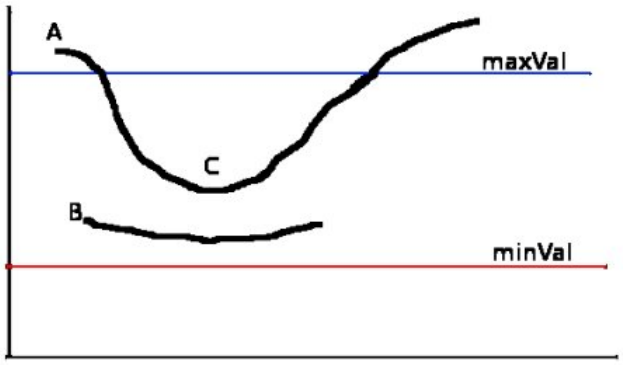


Figure 3.9: Hysteresis Thresholding [2]

Finally, we get strong edges in the image.

[2] https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

Pattern Matching

To match a model image and a search image made up of features, the features have to be represented mathematically. The feature detection in PatMax® produces a geometric description of object boundaries as an ordered list of boundary points that lie along an edge. Each boundary point also specifies the position and orientation as real-valued coordinates. PatMax® uses a range of granularity that is appropriate for each particular image. Geometric descriptions of objects only make sense when the proper granularity is chosen. PatMax® performs a training procedure to choose the range of granularity that makes sense for objects to be found and then to select features at each of the various chosen granularities to represent the object. The geometric descriptions of objects produced by feature detection can be translated, rotated, scaled, stretched and otherwise transformed with no loss in fidelity. PatMax® next performs a pose refinement algorithm to compute translation, orientation and scale of an object to extremely high accuracy. Algorithms that locate objects having various scales and orientations are commonly referred to as RISI (Rotation Invariant Scale Invariant) tools. For complete description of the pattern matching algorithms, the reader is advised to refer to the patents by Cognex: Sarachik *et al.* (2001), Bachelder (2006).

3.3.4 Robot-Vision Guidance

After the object pose is found by the camera, it is necessary to convert this pose from the camera-frame to the robot tool-frame. We will consider a stationary over-the-shoulder camera as in Figure 3.10. Another configuration would be the camera attached to the tool frame, which will not be considered. The objective is to find ${}^{Robot}\mathbf{T}_{\mathbf{object}}$.

We will use the multiplicative rule of transformation matrices in Equation below:

$$ {}^{Robot}\mathbf{T}_{\mathbf{object}} = {}^{Robot}\mathbf{T}_{\mathbf{tool}}.{}^{tool}\mathbf{T}_{\mathbf{camera}}.{}^{camera}\mathbf{T}_{\mathbf{object}} $$

- ${}^{camera}\mathbf{T}_{\mathbf{object}}$: This is the object pose in camera frame. We have this from the previous section on object pose recognition 3.3.3.
- ${}^{Robot}\mathbf{T}_{\mathbf{tool}}$: The transformation from `Robot-base` frame to `Tool-flange` frame is found as the direct kinematic model using the Denavit-Hartenberg parameters [Denavit (1955)]. To find the transformation between `tool-flange` frame and `tool` frame, we make use of the built-in Tool Centre Point (TCP)

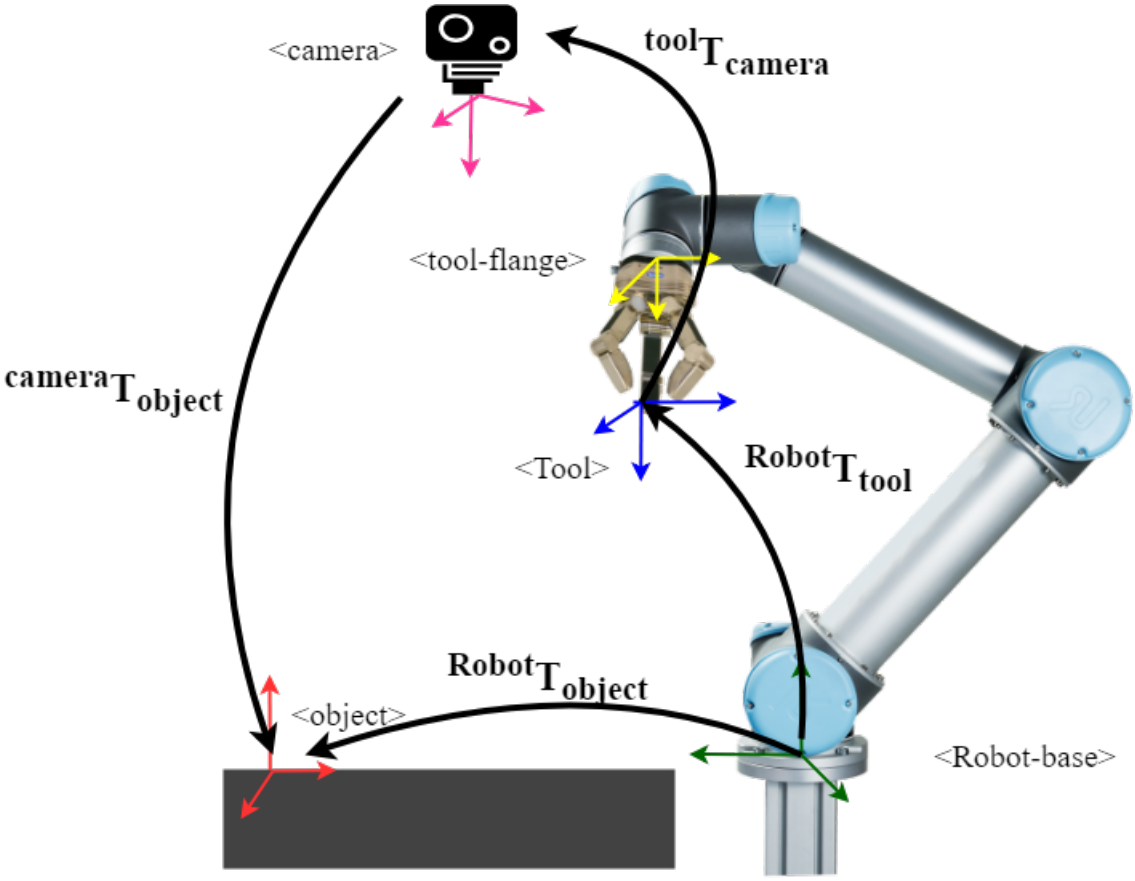


Figure 3.10: The different frames in robot-vision guidance

calibration function present in robot controllers. It is performed with a calibrated tool mounted in the tool of the robot e.g. a tip. The tool tip is brought into contact on a particular point with different joint orientations and the function calculates the active TCP frame.

- $^{tool}\mathbf{T}_{\mathbf{camera}}$: This is known as the *Hand-Eye Calibration*. There are multiple methods in literature exploring methods to perform this process. One method is to ensure the same origin, x- and y-axis of the `camera` frame and tool frame. This can be done by defining a plane with touching the tool with 3 points for origin, x- and y- axis respectively. A fiducial calibration grid can be useful for this procedure. Another method would entail mounting the a calibration target on the tool frame of the robot. The robot is moved to different poses while camera and robot poses are calculated. The camera poses can be estimated with Perspective-n-Point algorithm [Penate-Sanchez

et al. (2013)]. This would result in solving the following equation :

$$A.X = X.B \tag{3.12}$$

where A is the robot tool pose, B is the estimated camera pose and X is the hand-eye transformation. Several research works have been dedicated to find close form solutions for the above problem, such as detailed in Park & Martin (1994) and Horaud & Dornaika (1995)

In the next section, we will look at methods used for robotic manipulation i.e, to pick and place the object and robotic motion planning.

3.4 Robot Interface

In a pick-place task, a robot must be able to manipulate the given object and move to required goal positions. Hence there are two broad categories of robot functions: robotic manipulation and robot motion planning. This section describes the theoretical concepts for these functions.

3.4.1 Robot Motion Planning

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement. In order to move the robot arm to a given position, the individual joints or the position of the TCP can be controlled. The former is known as trajectory planning in **joint space** while the latter is planning in **Cartesian or tool space**. In an industrial setting, there can be few issues controlling the robot in Cartesian space. It depends on inverse kinematics, i.e the mathematical process which determine the joint parameters that provide a desired position for robot's end-effectors. This is computationally intensive and can be time-consuming process. It can lead to very high speeds and accelerations for the individual joints and therefore lead to a shut-down of the system. There are possibilities where the robot can be in a singular configuration while controlling the TCP of the robot. This results in the robot losing one Degree-of-Freedom and enter a **Protective Stop**. Another issue with Cartesian space control is that for every position there are 8 possible robot configurations [1]. So during the trajectory, we are never sure of the posture of the arm (joint configuration). This implies the path needs to be checked and collision avoidance needs to be done. In an industrial setting, the environment is very controlled. Hence

[1] http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf

it is undesirable to have new trajectories generated for each motion. Therefore for a more predictable and safer to use system, we will use joint space for most of the robot movement and controlling in Cartesian space only when the robot needs to perform manipulation.

We will use the ROS-Industrial Universal Robots driver which will be described in Section 4.2.4. The driver essentially exposes an **action interface** connected to a low-level **Joint Trajectory Controller**. The **action interface** provides a standardized interface for interfacing with preemptable tasks. Commanding a joint trajectory for the arm requires the following components:

- A controller that sends commands directly to the joints;
- An action interface to the controller in the form of a ROS action, that takes in a trajectory command expressed as a series of joint angles and sends the appropriate low-level commands to the controller;
- The high-level program that uses the action interface to issue the desired joint trajectories.

The first two requirements are satisfied by the `ur_modern_driver` [4.2.4], the official UR driver from ROS-Industrial. The last component is done by the *Robot Interface* [4.3.3]. This section will introduce the reader to a brief explanation about how the controller defines trajectories given joint angle commands, why we need a **action interface** and finally how the joint commands are recorded.

In joint space control, there are two possibilities: **Joint Position Control** and **Joint Velocity Control**. Joint position control mode is the fundamental, basic control mode for UR arm motion. In position control mode, we specify joint angles at which we want the joints to achieve. Typically this will consist of six values, a commanded position for each of the seven joints, resulting in a full description of the arm configuration. Position control introduces an actuation delay in tracking, but does not require an external controller. Joint velocity control mode is an advanced control mode. In velocity control mode, we specify joint velocities which we want the joints to simultaneously achieve. Typically this will consist of six values, a commanded velocity for each of the seven joints. The velocity mode controller has a lower delay but suffers from drift over time. Hence it is necessary to have an external positional controller to close the loop while using Velocity based joint control. In our application, as we do not require very fast tracking and the actuation delay is not very critical, we will use the **Joint Position Control**.

Given two points in joint space i.e the current and desired goal position, an interpolation is required between these points. This is the task for the trajectory generation. In general, we assume one generic variable u which can be position or orientation variables and use it to define our curves connecting two points. The candidate curves is shown in figure 3.11 [1].

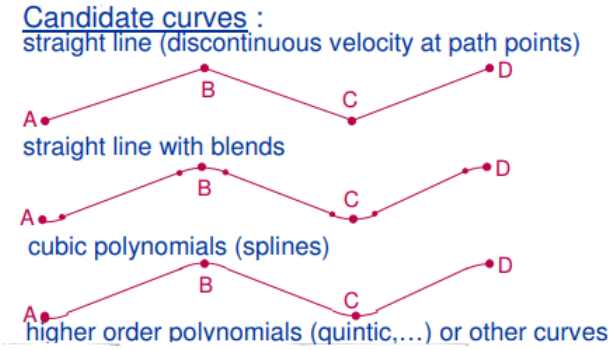


Figure 3.11: Candidate curves for trajectory generation

The mathematical derivation for the trajectories are beyond the scope of this thesis. The `ur_modern_driver` does **cubic interpolation** between two points, given the absolute time between them, the time at which the position should be found, and the velocity at the two points. Note that this function does not adhere to any constraints and, as it is a cubic interpolator, is not jerk continuous.

Now that trajectories are defined and sent to the robot, there may be cases during execution the trajectory needs to be cancelled or changed. These may arise due to obstacles or changing actions by the *Planner*. Hence we require a system that gives feedback and results on the current trajectories, and the possibility to preempt the trajectory with another trajectory. This is done using the **ROS Action Library** through the **action interface** shown in figure 3.12 [2].

The official name for preempting the current trajectory with a new trajectory is called **trajectory replacement**. The steps followed by the controller for trajectory replacement are as follows:

- Get useful parts of the **new trajectory**: Preserve all waypoints whose time to be reached is in the future, and discard those with times in the past. If there are no useful parts (ie. all waypoints are in the past) the new trajectory is rejected and the current one continues execution without changes.

[1] https://see.stanford.edu/materials/aiircs223a/handout6_Trajectory.pdf
[2] http://wiki.ros.org/actionlib/DetailedDescription

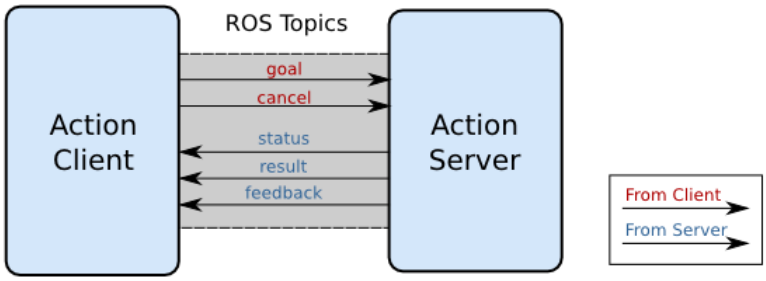


Figure 3.12: Action Interface

- Get useful parts of the **current trajectory**: Preserve the current trajectory up to the start time of the new trajectory, discard the later parts.
- Combine the useful parts of the current and new trajectories.

The trajectory replacement can be done immediately or providing some time in future. In our case, we employ trajectory replacement in situations where the higher-level **Task Planner** decides to change the action command. Hence we need to change the trajectory immediately. The important consideration in such cases is to have a smooth transition and avoid abrupt joint commands causing jerks. This is shown in figure 3.13 [1].

Finally, these waypoints must be recorded and sent to the `Joint Trajectory Action Interface` as joint angle commands. Generally, in industrial setting the robot waypoints are taught by **Programming by Demonstration (PbD)** [Alexandrova *et al.* (2014)]. PbD is characterized by two phases: teaching, in which one or multiple examples are shown by means of physical guidance, and learning, in which examples are generalized in order to obtain a resulting robot behavior. The execution of the task, then, is simply the autonomous repetition of the learned behavior. The teaching is done by **Kinesthetic teaching (KT)**. However in this thesis, we do not directly employ PbD, rather we extract safe waypoints by KT and feed this to the controller to perform the trajectory generation. These waypoints are stored in a database which is accessed by the *Robot Interface*. More about this is explained in Section 4.3.3.

3.4.2 Robot Manipulation

Robotic manipulation refers to the ways robots interact with the objects around them: grasping an object, destroying an object, opening a door, packing an order

[1] http://wiki.ros.org/joint_trajectory_controller/UnderstandingTrajectoryReplacement

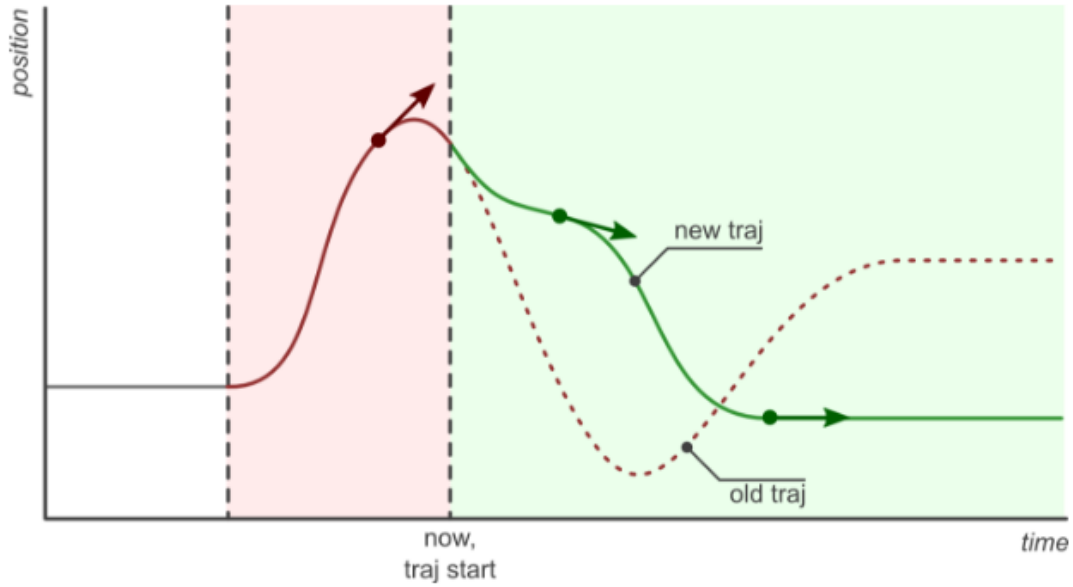


Figure 3.13: Trajectory replacement

into a box, folding laundry and so on. In an industrial setting, manipulation often involves pick-and-place tasks that can be broadly classified into two major categories: structured manipulation and unstructured manipulation.

Structured manipulation is manipulating objects one-by-one in a repetitive, fixed and structured environment. If the objects are identical and the motion is repetitive, gripper design and motion programming can be planned offline. The motion can have variations in output pose, such as palleting objects into a box. Variations in input pose are also possible, provided the pose can be determined accurately enough, typically with a vision system [Mason (2018)]. In general, structured manipulation involves the programmed motion technique. It is defined as the ability to impart a specified motion to any load attached to the robots mechanical interface, with limits on the workspace, speed, acceleration, and load weight and with specified repeatability and accuracy. Furthermore, manipulation can be considered as a direct kinematics problem. Consider moving a set of identical blocks on a table-top: Grasp planning is trivial - align and center the gripper on the block, and squeeze. Placement planning is also trivial - put the block anywhere on the table. Hence, this would require only overall path planning.

Unstructured manipulation on the other hand is far more generic and

more difficult. It involves solving the grasp-dynamics problems and the motion planning problem. Grasp pose planning is the problem of determining the stable grasp poses for a given gripper and object. Hence, to pick up random objects from a cluttered, unstructured scenario we would also need different types of grippers with tool-changing mechanisms or an anthropomorphic gripper capable of grasping different types of objects.

In this thesis, we will only use the structured manipulation technique, as the environment is defined and pre-defined motion paths are preferred. The pick-place technique involves using a vision system to provide the pose of the object. The robot controller solves the IK problem to reach this pose. Intuitively, we align the grippers over the part and close the grippers until grasp has been detected. Also, since the goal-pose for the parts are known apriori, it is only the path-planning problem which needs to be solved to reach these goals. As defined in the previous section, this is done offline by kinesthetic teaching.

As concluding remarks, we explored the proposed architecture designed to complete the given human-robot collaboration task. In particular, we explained the underlying concepts of the various modules involved in the architecture, namely the *Task Representation*, *Task Planner*, *Vision System* and *Robot Interface*. These modules or sub-systems are developed as stand-alone software modules. The performance of the system will rely on accurate communication between the modules and temporal synchronisation of the respective modules. The next chapter on Methodology will explore the implementation details of the architecture.

Chapter 4

Methodology

This chapter describes the steps undertaken to implement the individual modules of the architecture described in Chapter 3. In section 4.1, we describe the hardware setup consisting of the robot, gripper, and vision system. In section 4.2, we describe the various possibilities to program the robot, providing the pros and cons for each choice. The section 4.3 consists of the software techniques utilized to integrate the various software modules of the architecture and finally the section 4.4 describes the experimental procedure and defines the experimental hypothesis we use for the subsequent evaluation.

4.1 Hardware Setup

This section describes the hardware setup for the Human-robot collaboration experiment. We define the hardware requirements for the task:

- A collaborative robot with atleast 6-Degrees-of-Freedom (DOF) necessary to maneuver inside the tight workspace.
- The collaborative robot must be inherently safe or safety sensors/barriers are necessary.
- The robot must be able to carry atleast 7 kg payload. Here it is necessary to distinguish between robot payload and gripper payload. The robot payload refers to the maximum mass that can be attached to the wrist of the robot arm. It comprises of the gripper mass, mass of the part, and bracket mass (to attach the gripper). The gripper payload refers to the maximum mass that can be grasped by the gripper.
- The gripper capable of easy installation on the robot, with a gripping force sufficient for transportable weight of 1kg.

- A 2D-vision system running on dedicated hardware or external computer capable of detecting and recognizing object pose even in presence of clutter.

4.1.1 Universal Robots UR10

In this section, we explore the technical specifications of the Universal Robots UR10. The robot plays a central role in the thesis, hence it is important to look into the capabilities and drawbacks of the robot. Furthermore, we will look into the various ways for communicating with the robot. Finally, although UR10 is designed to be a collaborative robot to work alongside humans freely, it can cause human-injury if the right safety measures are not adopted. This entails a discussion on hardware safety methods as well.

4.1.1.1 Specifications

The UR10 is a collaborative robot designed and produced by Universal Robots, a Danish robotic manufacturer. The robot is designed to be light weight, safe and easy to use. The robot actively checks the forces on the joints by calculating the current usage. In case of contact with obstacles, the robot comes to stop immediately. Hence, it does not need a protective cage around it and is ideal for our topic of research.

The Figure 4.1 shows the key specifications of the UR10 robot. The Figure 4.2 shows the dimensions of the UR10 Robot. The UR10 is running firmware version CB3.0 and PolyScope version 3.5. An important specification is the payload of 10kg. As we will see in Section 4.1.4, this is sufficient to carry the part. Also, the reach of 1.3m affects the workspace design of the robot and the human operator. Also note, the robot is a 6 Degree-of-Freedom (DOF) manipulator and each joint $q_i \in [-2\pi, +2\pi]$ where $i \in (1, 2, \ldots, 6)$. Henceforth, these joints will be referred to as Base, Shoulder, Elbow, Wrist1, Wrist2 and Wrist3. The names are important for the `ROS Parameter Server` as we will see later. Finally, the robot offers a repeatability of $\pm 0.1mm$. From the specification requirement list in Section 4.1, we realize this is sufficient for our task. However we must note that other competitor collaborative robots such as Kuka LBR iiwa 14 R820 offers $\pm 0.15mm$ repeatability while industrial manipulators such as IRB1200 ABB offers $\pm 0.02mm$ repeatability. The rule of thumb is standard industrial manipulators are faster, more precise and can carry heavier payloads compared to the state-of-art collaborative robots, but the latter are inherently safer in a HRC scenario.

Before moving further, it is necessary to perform singularity analysis of the

Performance

Repeatability	±0.1 mm / ±0.0039 in (4 mils)
Ambient temperature range	0-50°
Power consumption	Min 90W, Typical 250W, Max 500W
Collaboration operation	15 advanced adjustable safety functions. TüV NORD Approved Safety Function Tested in accordance with: EN ISO 13849:2008 PL d

Specification

Payload	10 kg /22 lbs
Reach	1300 mm / 51.2 in
Degrees of freedom	6 rotating joints
Programming	Polyscope graphical user interface on 12 inch touchscreen with mounting

Movement

Axis movement robot arm	Working range	Maximum speed
Base	± 360°	± 120°/Sec.
Shoulder	± 360°	± 120°/Sec.
Elbow	± 360°	± 180°/Sec.
Wrist 1	± 360°	± 180°/Sec.
Wrist 2	± 360°	± 180°/Sec.
Wrist 3	± 360°	± 180°/Sec.
Typical tool		1 m/Sec. / 39.4 in/Sec.

Figure 4.1: Specifications table of UR10 [uni (2018)]

UR10 Robot. Singular poses are robot poses where it is not possible to solve the inverse kinematics. In singular areas, the robot loses one or more degree of freedom in Cartesian coordinate system. Thus, movement in Cartesian space is limited in certain directions.

There are three known singularities for the UR10 robot as shown in Figure 4.3. The first is when the tool moves in Cartesian space in the region directly above and below the base of the robot. It causes the joints to accelerate very quickly though the tool is moving slowly as in Figure 4.3 (A). The second type is when wrist 2 joint is 0 deg or 180 deg, 4 axes are co-aligned with the same direction then the robot lost one or more degree of freedom in the certain direction as in Figure 4.3 (B). Finally, as in all standard 6-DOF manipulators, when the robot is fully extended as in Figure 4.3 (C), the robot loses a 1-DOF.

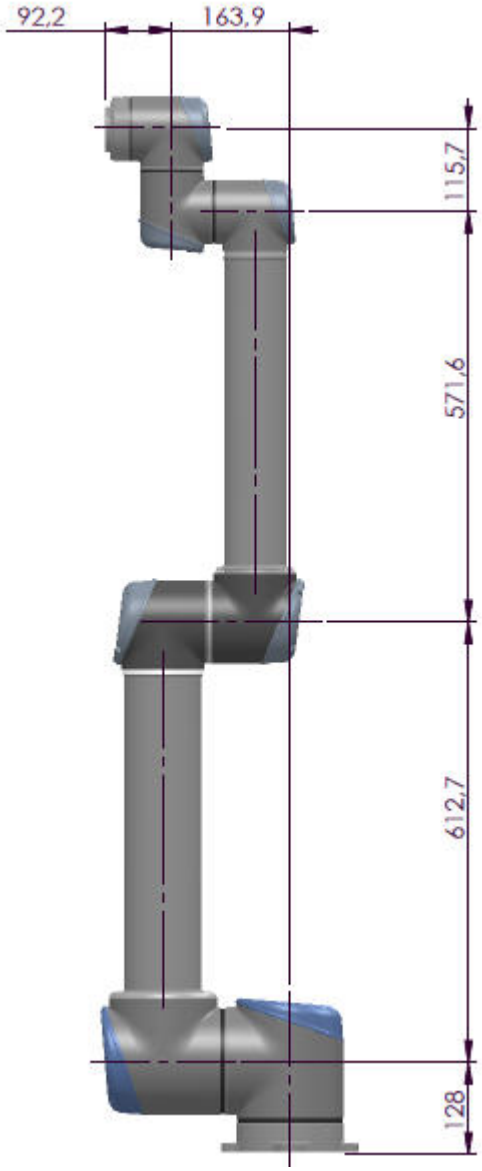


Figure 4.2: Dimensions of the UR10 robot [uni (2018)]

The robot waypoints and motion trajectories must be designed so that they do not go close to joint limits or singularities.

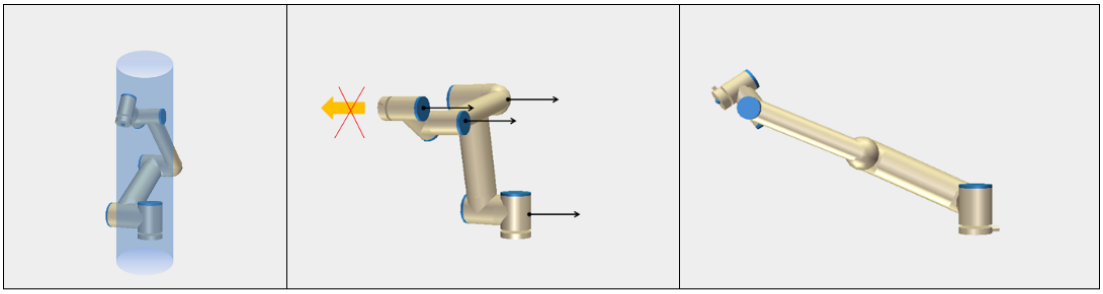


Figure 4.3: Various singularities of the UR10 robot [uni (2018)]

4.1.1.2 Communication with the UR10

There are a few ways of communicating with the UR10 robot. This section will explore the possibilities and in conjuncture with the programming section 4.2 decide the best method for our application.

In most industrial applications, the robots are programmed using teach pendants with their proprietary industrial robot software Neto *et al.* (2010). Similarly, the UR10 robot can be primarily programmed using the teach pendant. The teach pendant offers a highly ergonomic, easy-to-use Graphical User Interface (GUI) which does not require highly specialized programming skills. Figure 4.4 shows the Teach Pendant. Although the teach pendant is sufficient for simple pick-place, palletizing, and manipulating tasks, it is not highly specialized to handle various sensor data, or run Artificial Intelligence (A.I) algorithms. Most importantly, we would like to *decouple* the hardware and software, thus developing programs which are robot-independent and open-source.

The robot controller box running the `URControl` software communicates with the Teach Pendant to receive commands and provide feedback. It also has one **Ethernet connection** through which robot commands can be provided through TCP/IP socket connections and also receive feedback about robot state. Hence we will use an external computer running the Robot Operating System (ROS) connected to the UR10 through Ethernet. This is aptly explained in Figure 4.5. This will be further expanded in Section 4.2.

4.1.1.3 Safety

Following the discussion on safety in Section 2.4, we realize that in case of Human-Robot Cooperation, the force/momentum transfer is more crucial than the tool-

Figure 4.4: UR10 robot with the Teach pendant [uni (2018)]

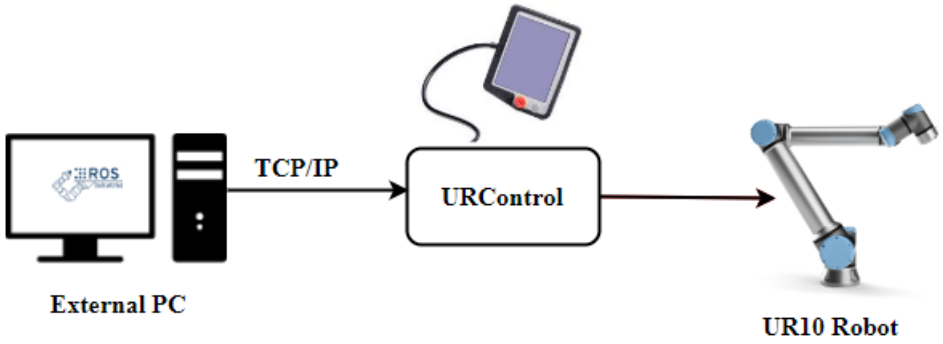


Figure 4.5: Communicating with UR10 Robot

speed regulation. The main safety strategies that we will use are [Robla-Gómez *et al.* (2017)]:

- Minimizing injury during collision by limiting velocity and impact force.
- Modifying robot behaviour by stopping the robot if it enters human workspace

The general safety limits is set on the Teach pendant and serve to limit the linear speed of the robot TCP as well as the force it may exert on the environment. They are composed of the following values:

- **Force:** A limit for the maximum force that the robot TCP exerts on the environment.
- **Power:** A limit for the maximum mechanical work produced by the robot on the environment, considering that the payload is part of the robot and not of the environment.
- **Speed:** A limit for the maximum linear speed of the robot TCP.
- **Momentum:** A limit for the maximum momentum of the robot arm.

The Table 4.1 shows the maximum allowed values for the aforementioned limits and the settings we will use in our application. We must note that all these values are as per the ISO 10218 standards Iso (2011). Therefore we limit the robot velocity and force/momentum transfer, and in case of collision with obstacle, the robot immediately triggers a **Protective Stop** and program execution is halted.

Limit	Maximum	Normal Mode
Force	250 N	100 N
Power	1000 W	300 W
Speed	5000 mm/s	500 mm/s
Momentum	100 kg m/s	25 kg m/s

Table 4.1: Safety Limits

We do not use any external sensors to track humans in robot workspace or utilize virtual barriers. Hence it is essential to ensure that the robot must never enter the human workspace in *any* condition. This is done by using *Safety Planes*. Safety planes can be used to restrict the allowed workspace of the robot by enforcing that the robot TCP stay on the correct side of the defined planes and not pass through them. We define two intersecting planes between the human and the robot as shown in Figure 4.6. The planes have been configured such that it will trigger a **Protective Stop** and abort the program execution should the TCP exceed the plane limits.

Finally according to ISO-TS 105066:2016, contact with the face, skull, forehead, eyes, ears, and larynx areas is not permissible. We take care of this regulation by ensuring that the robot TCP is always below the average head height of a human. This is seen from Table 4.2, notice that throughout the trajectory the robot TCP always remains below average shoulder height of men and women [1].

[1] http://lvtgw.jadephoenix.org/Info_htm/HowTo/bod_num1.htm

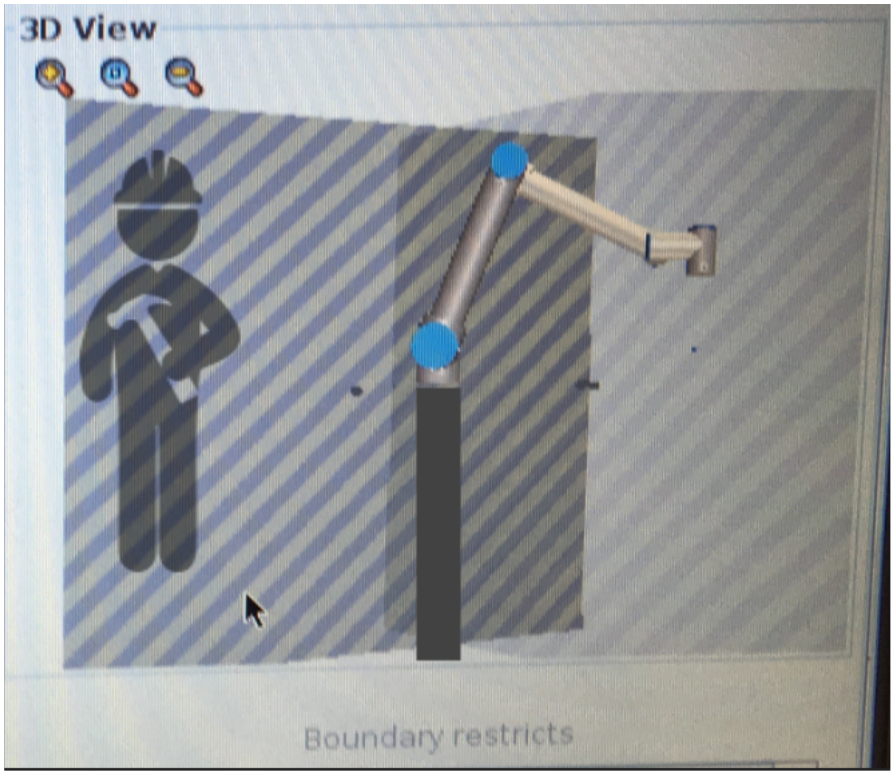


Figure 4.6: Safety Plane configuration

Entity	Value (m)
Average Height Male	1.742
Average Height Female	1.631
Average Height Shoulder	1.386
Robot Base from ground	0.88
Max Height Robot TCP	1.01
Min Height Robot TCP	0.7

Table 4.2: Distance from ground of various safety parameters

4.1.2 RG6 OnRobot Gripper

For the manipulation tasks, we have chosen to use the **OnRobot RG6** gripper. It is a plug-n-play type collaborative gripper. Most importantly, it works without any external cables and power supplies. It fits into the tool flange of the UR10 robot seamlessly and derives power and control signals from the tool output connector of the the robot. The Figure 4.7 shows the technical specifications of the gripper and Figure 4.8 shows the mechanical design of the gripper.

Technical data	*Min*	*Typical*	*Max*	*Units*
Total stroke (adjustable)	0	-	160	[mm]
Finger position resolution	-	0,15	-	[mm]
Repetition accuracy	-	0,15	0,3	[mm]
Reversing backlash	0,4	0,7	1	[mm]
Gripping force (adjustable)	25	-	120	[N]
Gripping force accuracy	±2	±5	±10	[N]
Operating voltage*	10	24	26	[V DC]
Power consumption	1,9	-	14,4	[W]
Maximum Current	25	-	600	[mA]
Ambient operating temperature	5	-	50	[°C]
Storage temperature	0	-	60	[°C]
Product weight	-	1	-	[kg]

Figure 4.7: Technical specifications of OnRobot RG6 Gripper [gri (2018)]

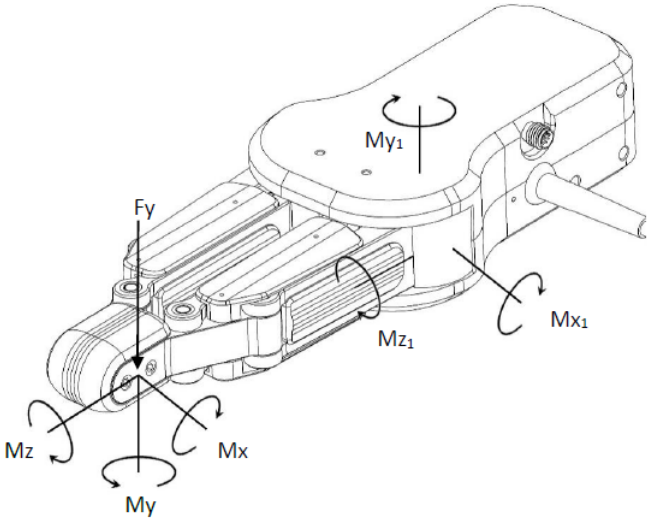


Figure 4.8: Mechanical design of OnRobot RG6 Gripper [gri (2018)]

We will employ the gripping with frictional force. From the specification table 4.7, the necessary gripping force for transportation of the part. Consider the schematic in Figure 4.9[1]

In the case of normal transportation i.e, without high accelerations, the condition that the part does not slip during stable grasp is

$$F.\mu > W \tag{4.1}$$

[1] http://www.intelligentactuator.com/partsearch/robocylinder/appndx74_Model_Selection_by_RCP2_Gripper.pdf

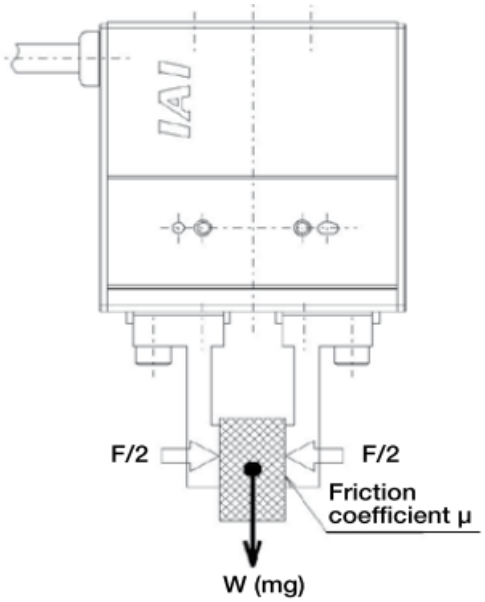


Figure 4.9: Calculation of necessary gripping force

or

$$\mathrm{F} > \frac{mg}{\mu}$$

It is recommended to use a safety factor of atleast 2 for gripping force.

$$\mathrm{F} > 2 \times \frac{mg}{\mu}$$

For most materials, the friction coefficient μ is between $0.1 - 0.2$.

$$\mathrm{F} > 2 \times \frac{mg}{0.1 \sim 0.2} = (10 \sim 20) \times mg$$

where, F is the gripping force, μ is coefficient of friction, $\mathrm{W} = mg$ is the weight of the object.

Thus, the gripping force must be atleast 10 to 20 times the weight of the part. Equivalently, the part must weigh one-tenth to one-twentieth the gripping force. The maximum gripping force of RG6 gripper is $120N$. Hence the maximum part weight can be between **0.6kg to 1.2kg**. Thus we match the hardware specification [4.1] regarding the transportable part weight.

4.1.3 Cognex Vision System

The **Cognex 7802 Vision system** shown in Figure 4.10 is used for the task. A brief look into the specifications of the vision system is provided in the Table 4.3:

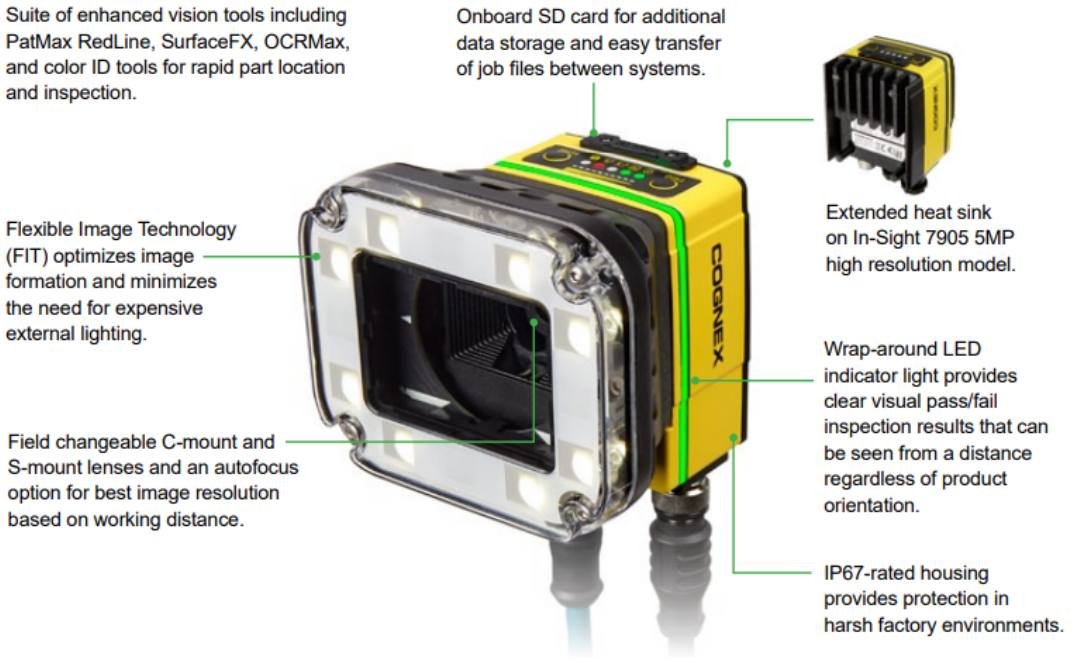


Figure 4.10: Cognex Vision system [cog (2018)]

Specifications	In-Sight 7802
Image type	Monochrome and colour
Job/Program memory	7.8 GB non-volatile flash memory
Image processing memory	512 MB SDRAM
Sensor type	CMOS, global shutter
Resolution	1600 x 1200
Acquisition rate (fps)	53 (monochrome), 33(colour)
Lens focal length	8mm
Light options	External light powered by 7802
Power	24 VDC

Table 4.3: Cognex 7802 specifications

The most crucial take-away from the specification table is the relationship

between the lens focal length, working distance and field of view. The working distance is the distance from lens to the part that needs to be inspected and the field of view is what the vision system can see at that distance.

Given that our working distance is $700mm$ and lens size is $8mm$, our field of view is $630mm \times 472mm$. The calculation is out of scope of this thesis.

4.1.3.1 Setup of Vision system

The vision system is connected to the external PC running through a TCP/IP connection. The IP address is chosen such that it is on the same subnet mask as the PC and the robot. For instance, Figure 4.11 shows the network connections between the vision system, the PC and the robot used in this task.

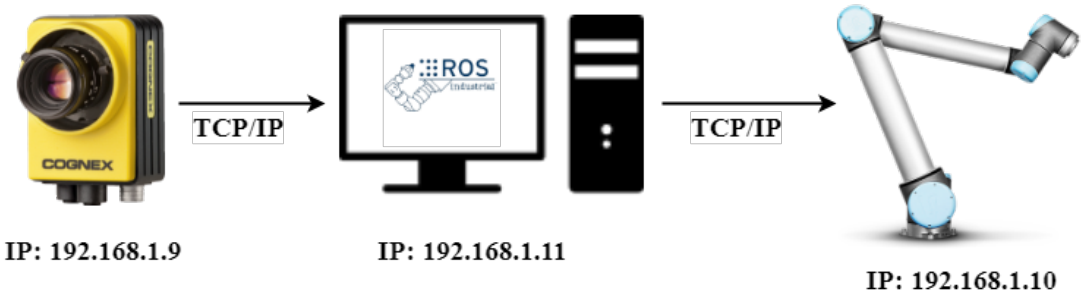


Figure 4.11: Network settings for vision system, PC and robot

The vision system is configured to send the object pose data over the Ethernet connection to the PC. Another important consideration is to check how the vision system will be triggered to capture image and perform pose recognition. Furthermore, the **In-Sight** software from Cognex which houses all the vision libraries works in an event-driven fashion. Thr trigger is an event supplied to the vision system to perform the necessary actions.

The Cognex 7802 features one **acquisition trigger input**, which is optically isolated. The acquisition trigger can be configured to trigger from a NPN (current sinking) or PNP (current sourcing) device. The trigger input requires a Voltage of 24VDC nominal for switching ON and 0VDC nominal for switching OFF. Similarly, it requires 6.6mA to 9.8mA current for ON and less than 1mA for turning OFF. These requirements are met by the General-purpose Input-Output (GPIO) pins on the robot controller. The UR10 manual states that the digital I/O can be used to communicate with other equipment if a common GND (0V) is

established and if the machine uses PNP technology. Hence we will use the **DO0** and **0V** digital output pins on the UR control box. From the breakout cables of the vision system, INPUT COMMON is connected to 0V pin and TRIGGER is connected to DO0 pin, shown in Figure 4.12.

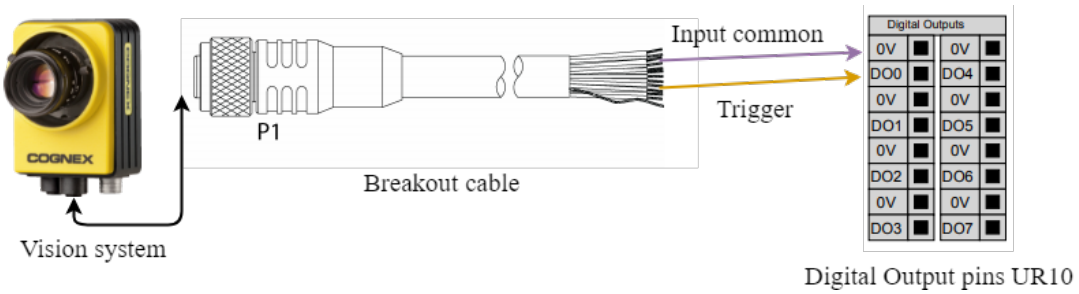


Figure 4.12: Trigger settings between vision system and robot

4.1.3.2 Camera Calibration

We use the non-linear **Grid** calibration wizard tool present in the In-Sight 5.4.0 Cognex Software, Figure 4.13. We use a grid of dots with fiducial (which indicates the X, Y and origin for the sensor).

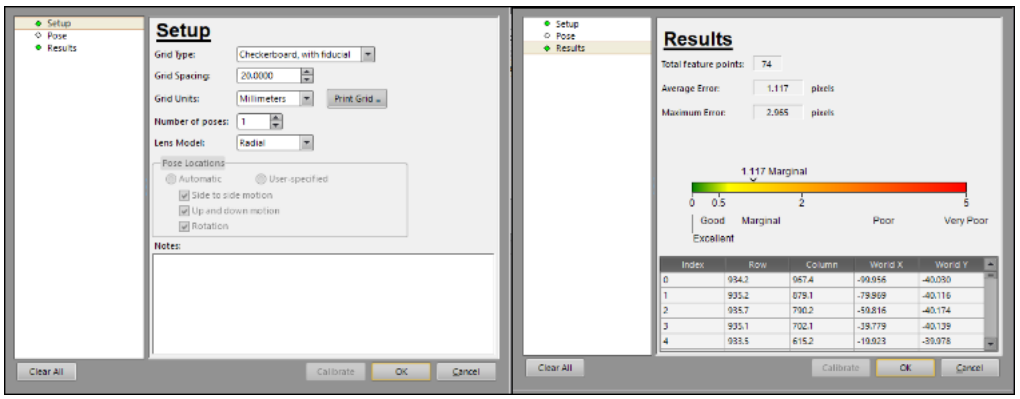


Figure 4.13: Calibration wizard in Cognex

Once the sensor is calibrated it is very important to calibrate the robot according to the same co-ordinates as those of the sensor. If the same co-ordinates were not used, calibration of the sensor and robot would differ, causing orientation problems during operation.

4.1.3.3 Configuring PatMax® algorithm

The calibrated origin in the previous section is stored as a `Fixture` in the PatMax® InSight 5.4 software. All the other pattern matched algorithms are defined with respect to this fixture, i.e we will get the part pose in x,y,$Rot(Z)$ with respect to the fixture in the defined axis of the fixture.

To use the PatMax® algorithm, we need to define two regions:

- **Pattern region:** Region specifying the feature to be trained. The pattern is dragged around the square printed region as this is the region to be trained.
- **Find region:** Region specifying the search area for the pattern within the image. The entire field of view can be used, but to decrease processing time the region has been made smaller

While training the image, it is also necessary to set the **Pattern origin**. This is selected by default as the centre of the circle, but can be changed if needed. We will use the same default centre. Furthermore, we must also set the parameters:

- **Find tolerances:** Used to set the allowable rotation of the object. In this case the allowable rotation is -180 to 180. Anything outside this rotation will not be allowed.
- **Find overlapping:** Sets the allowable overlapping between matches found.

The trained image is shown in Figure 4.14. The input image and PatMax® feature detection is shown in Figure 4.15.

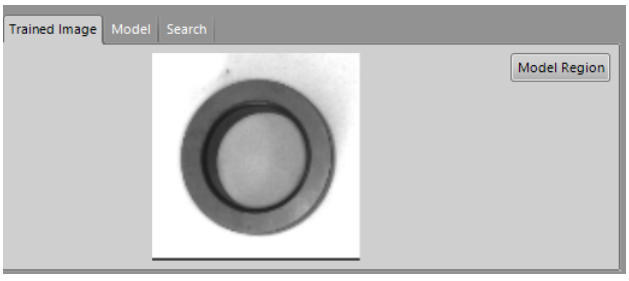


Figure 4.14: The trained image for PatMax®

Finally, the results of the pattern must be sent over the TCP/IP network to the host PC. This is done by the means of `FormatString` function. The `FormatString` is set as (x, y, rot(Z), %score) wherein the %*score* refers to the

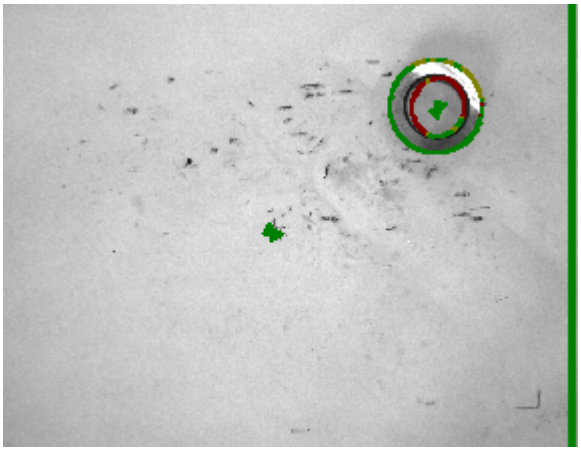


Figure 4.15: The PatMax® feature detection

percentage of match between the trained image and input image. The algorithm works in such a way that, if there are multiple parts found, the image with the largest *%score* is used to `FormatString`.

4.1.4 Parts and Pallets

We use the following ring-like structure in Figure 4.16 as the workpiece for the robot. It is one of the sub-assemblies of wheel bearing assembly. The specifications of the part is shown in Table 4.4.

Figure 4.16: Parts (A) Top view (B) Side view (C) Bottom view

As the part is symmetrical, grasping can be done at any convenient position on the ring in the same way. After the part origin is found through the camera, the robot translates on x-axis of the part frame by 50 mm to grasp on the ring structure. This is aptly explained in Figure 4.17

Entity	Value
Weight	0.118 kg
Outer diameter	60 mm
Inner diameter	40 mm
Thickness	22mm
Material	Steel

Table 4.4: Part Specifications

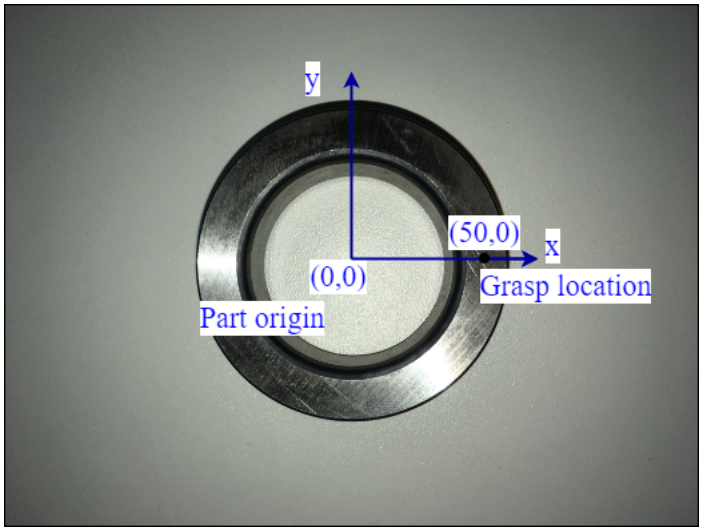


Figure 4.17: Grasp location of the part

The pallet is a standard industrial metal box shown in Figure 4.18. We will consider only one layer for the complete task. One layer in the pallet has 3 columns and 5 rows and holds upto 15 parts.

Figure 4.18: Pallet Box (A) Empty (B) Complete

4.2 Software Setup

In Section 4.1.1.2, we introduced the methods to program the UR10 robot. In this section, we will discuss in detail the various ways the robot can be programmed and validate our choice. There are four different ways of programming the UR manipulators; with a teach pendant, using code generated in URScript or C-API and using ROS drivers (Figure 4.19). Both the teach pendant and URScript runs on the original firmware with a version of a low-level robot controller called URControl. The internal controller sends commands to the joint servos at 125Hz and is thus able to evaluate a new set of instructions every 8th millisecond.

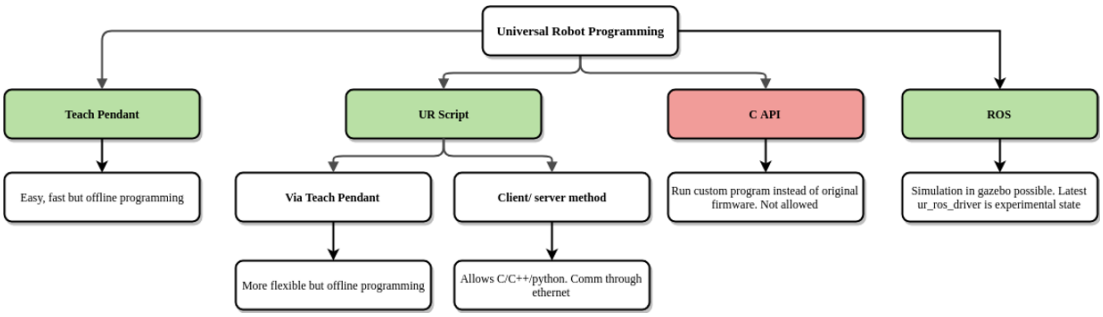


Figure 4.19: Various ways of programming UR10

4.2.1 C-API

Universal Robot provides a C library for direct control of all joint servos in the robot. With direct control of the applied joint torques it's possible to create a custom low level controller that runs on the robot with minimal delay. The fastest implementation will still have a updating speed limited by the joint servos updating rate of 125 Hz. However, a major disadvantage is it is necessary to install the program on the controller box and the Teach pendant is rendered useless. This is not allowed for legal reasons in Schaeffler. Hence, the drivers developed using C-API technique such as `ur_c_api` by Kelsey Hawkins [1] will not be discussed further.

4.2.2 Teach Pendant

The teach pendant is running a program called Polyscope which is a graphical user interface (GUI). It provides control and programming interactions with the

[1] http://wiki.ros.org/ur_c_api_bringup

UR manipulator. Polyscope can be a powerful tool for fast implementation, prototyping and debugging, but it was mainly developed for users without a programming background. For reasons suggested earlier, we will not be using the PolyScope in this thesis.

4.2.3 `URScript` Programming

`URScript` is a robot programming language developed by Universal Robots. Like other robot programming languages, `URScript` provides basic functionality like variables, types, flow of control statements and basic threading. It also provides a lot of robot specific functionality such as built-in variables and functions that monitor and control the I/O and movements of the robot. However, it does not support classes, and the threading functionality is only basic. It supports nested threading and basic mutex operations to avoid race conditions, but it does not support parameters in thread function calls or semaphores. It basically uses four methods for controlling the robot: `move`, `servo`, `speed`, and `force`. For a complete description of these functions, the reader is advised to look into `URScript` API reference [1].

As we will see in Section 4.2.4, `URScript` functions can also be used with ROS-Industrial drivers. In the present task, the URScript functions will be used in two cases:

- **Gripper functions:** To communicate with the RG6 gripper for grasping, un-grasping and grip-detection. See section 4.2.4.1
- **Inverse Kinematics:** Utilizes the built-in inverse kinematics engine in URcontrol. See section 4.2.4.1.

4.2.4 ROS Industrial

ROS-Industrial is an open-source project that extends the advanced capabilities of ROS software to manufacturing automation and robotics. A brief overview on ROS-I can be seen in Figure 4.20.

The `universal_robot` repository [2] is part of the ROS-I repository and contains robot models, parameters, interactions, communication interfaces, standard controllers and visual representation. It supports the UR3, UR5 and UR10 robots with the latest firmware and drivers. We use the `ur_modern_driver` [Andersen

[1] http://www.sysaxes.com/manuels/scriptmanual_en_3.1.pdf
[2] http://wiki.ros.org/universal_robot

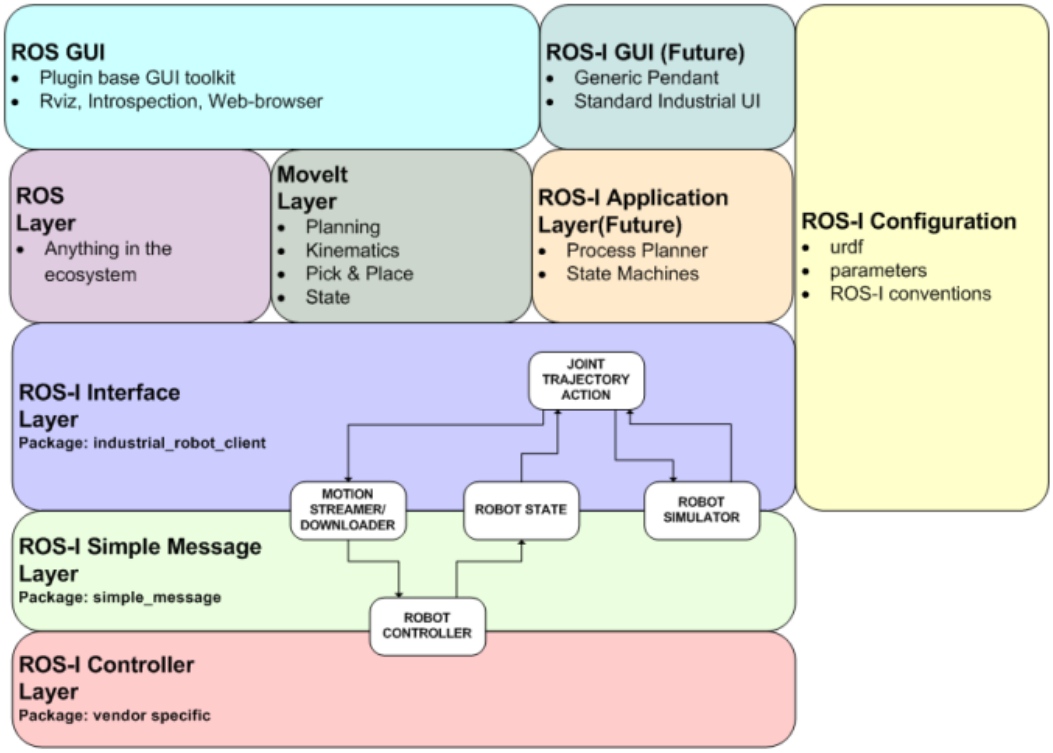


Figure 4.20: Overview of ROS Industrial

(2015)] which is the latest stable driver for the Universal Robots as the basis of this project thesis. The driver is written in C++ and is designed to be a direct replacement for the deprecated `ur_driver` with a lot of improved functionality.

The driver has one script running on the robot controller while trajectory is executing. The teach pendant can be used to move the robot around while the driver is connected. It exposes a ROS Action interface on `/follow_joint_trajectory` for seamless integration with MoveIt. Further functionalities include: Publishing robot joint state on `/joint_states` topic; publishing TCP force on `/wrench` topic; publishing IO state on `/ur_driver/io_states` and spawning a ROS Service to set outputs and payload of robot. The trajectory action interface receives new goals and verifies that the trajectory is valid. If it is approved, the driver does cubic interpolation between the trajectory goals and transmits the intermittent joint position to the custom script running on the UR controller. The driver can operate at 10Hz or 125Hz update rate. The driver supports I/O interactions and all main `URScript` commands for motion, including commands needed for providing force control of the UR manipulator. It is also compatible with large

ROS repositories that provides position and force control.

In this project thesis, the *Robot interface* module sends trajectory goals to the `ur_modern_driver`. The `ur_modern_driver` in turn, sends the required joint values to the robot.

4.2.4.1 Extending the `ur_modern_driver`

We have extended the features present in the `ur_modern_driver` to facilitate our requirements.

Gripper Function

We design and develop a ROS driver using `URScript` functions and integrate it within the `ur_modern_driver`.

At the beginning of the program, a new ROS Service called **RG6** is launched along with the existing services for trajectory action interfaces. The **RG6 service** receives the required target width, target payload and target force. The callback function of the service handles the request and sends the command values to `rg6Control` function in the `ur_driver`. User-defined applications can interact with the gripper only through the service. This methodology enforces data encapsulation. The service also takes care of validating the incoming commands. The algorithm of `rg6Control()` function is as follows:

Algorithm 1 rg6Control()

```
Data: target_width, target_force, target_payload
set cmd_str ;
 cmd_str ← target_width;
 cmd_str ← target_force;
 cmd_str ← target_payload;
 if gripper not busy then
   /* Validate command width                                    */
   if target_width > 160 then
    | target_width = 160
   else if target_width < 0 then
    | target_width = 0
   /* Validate command force                                    */
   if target_force > 120 then
    | target_force = 120
   else if target_force < 25 then
    | target_force = 25
   /* Validate command payload                                  */
   if target_payload > 6 then
    | target_payload = 6
   else if target_payload < 0 then
    | target_payload = 0
   /* Prepare to send bit                                       */
   cmd_str ← rg_data = floor(target_width)*4
    cmd_str ← rg_data = rg_data+floor(target_force/5)*4*161
    cmd_str ← bit(rg_data)
    cmd_str ← set_payload(target_payload)
end
/* Send command to real time interface queue                    */
UR_realtime_interface ← cmd_str
```

Note: while preparing the data to send to the gripper, the `rg_data` is multiplied with constants. These are taken from the manual relating to how the target force and width are mapped within the gripper controller.

Next, we have another ROS service which polls the gripper fingers to check for **grip detection**. This service, `rg6_detect_service` takes an empty request and provides **boolean** response if the grip has been detected or not. The algorithm for `rg6GripDetect()` is shown below:

Algorithm 2 rg6GripDetect()

Data: ip_addr_host, port
set cmd_str
    cmd_str ← grip_detected=False
    `/* Grip detection is checked by digital input [8] state */`
if *get_digital_in(8) == True* **then**
    `/* Grip has been detected */`
    cmd_str ← grip_detected=True;
end
`/* Open TCP/IP socket */`
cmd_str ← open_socket(ip_addr_host, port)
    cmd_str ← socket_send_string(grip_detected)
    cmd_str ← close_socket()
    `/* Send command to real time interface queue */`
UR_realtime_interface ← cmd_str

Inverse Kinematics

After the object pose is received from the camera in robot-base frame, it is necessary for the robot-tool to go to the object pose for manipulation. As described in Section 3.4.1, there are mainly two types of techniques to perform inverse kinematics: closed form solutions and iterative techniques. There are toolboxes available for each of these techniques.

- The Robotics ToolBox by Peter Corke [Corke (2017)] explores iterative solutions for IK by using the Newton-Raphson techniques. This is built for MATLAB.
- There is a closed-form solution IK written in C++ by Hawkins (2013). This solution is ideally suited to be used along with C-API technique [4.2.1].
- The `ur_modern_driver` offers easy integration with MoveIt out-of-the-box. There are IK toolboxes such as `trac_ik` [1] and Kinematics and Dynamics Library (KDL) [2] which can be used. However, there are many technical issues using MoveIt:
  - The trajectory plans generated are not consistent. Sometimes the IK solutions are not found at all.
  - The motion generated from MoveIt causes protective stops often when it is used along with the `ur_modern_driver`. This is due to high velocities sent by MoveIt.

[1] http://wiki.ros.org/trac_ik
[2] http://wiki.ros.org/orocos_kinematics_dynamics

– There are jerky motions caused by MoveIt if wireless connection is used on host PC and TCP/IP latency.

Due to these reasons, we have chosen not to use MoveIt but rather use the built-in Inverse Kinematics `URScript` functions to solve our problem.

A ROS Service `IK_service` is defined which accepts requests with x and y position in mm and calls the `getObjectPose()` function which is shown below:

Algorithm 3 getObjectPose()

Data: x, y, origin
`/* Convert to meters */`
x_pose = x/1000
y_pose = y/1000
set cmd_str
`/* Define the offset pose */`
cmd_str ← global offset=p[%f,%f,0,0,0,0]",x_pose,y_pose)
cmd_str ← global target=pose_trans(origin,offset)
`/* Go 70mm above the object- approach pose */`
cmd_str ← movel(pose_trans(p[0.0,0.0,0.07,0.0,0.0,0.0], target), a=0.5, v=0.25)
`/* Go to object pose */`
cmd_str ← movel(pose_trans(p[0.0,0.0,0.00,0.0,0.0,0.0], target), a=0.5, v=0.25)
`/* Send command to real time interface queue */`
UR_realtime_interface ← cmd_str

Note: we need the `origin` pose which is the world origin. This must be the same origin pose for the camera frame and robot tool frame as discussed in section 4.1.3.2. Also we use the `pose_trans(p_from,p_from_to)` `URScript` function which transforms, that is translates and rotates, p_from_to by the parameters of p_from. This is aptly explained in Figure 4.21

4.2.5 Simulation Environment

This section deals with a short review of the three potential simulation candidates which support Universal Robots: `V-REP`, `Gazebo` and `URSim`. We will justify our choices and describe how simulation of the tasks were carried out.

Creating an accurate simulation environment is one of the goals of this thesis as for the first half period of the thesis, we did not have access to the real hardware. Some requirements from the simulation software are:

- It must be readily compatible with ROS, or atleast through a pre-existng API.

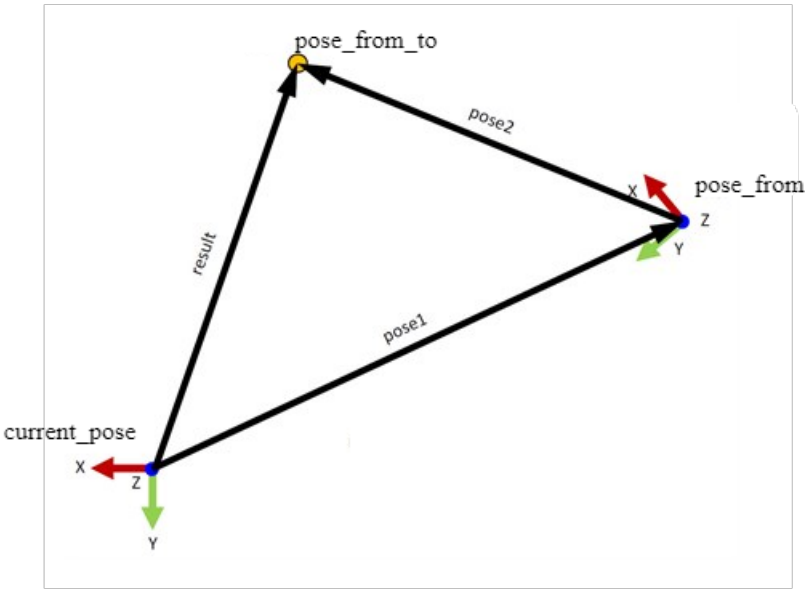


Figure 4.21: `pose_trans()` function

- It must support Universal Robots i.e, UR10 specific robot models, kinematics and dynamics.
- Actively large community for technical support.

4.2.5.1 `V-REP`

`V-REP` is developed by Coppelia Robotics and supports several programming languages and is available for Windows, Mac OS and Linux users. It can be accessed by ROS nodes though a remote API. It provides an intuitive GUI, accurate graphics engine, large community, extensive documentation and good online tutorials. However, it does require some work to setup `V-REP` to work alongside `ur_modern_driver` as compared to `Gazebo` and `URSim`. Also the community support for `V-REP` is not as large as `Gazebo`. Hence, we will use `Gazebo` and `URSim` to create the simulation environment and test the architecture.

4.2.5.2 `Gazebo`

`Gazebo` is a 3D dynamic simulator with the ability to accurately and efficiently simulate robots in complex environments. It is the preferred simulation environment used by the ROS community. Some key features include multiple physics engines, rich library of robot models and environments, wide variety of sensors and an intuitive GUI. It is supported by RVIZ, a powerful robot visualization

tool, that provides visualization of path planning, current states, sensor inputs and logging information for debugging. `Gazebo` runs on Linux, supporting almost all main versions of Ubuntu depending on the chosen ROS version. It has an extensive library of preexisting robot models, a large community and several independent tutorial sources available. ROS-I also has robot models for UR10 in `Gazebo` and supports it natively. Another advantage of using `Gazebo` is the interconnection between simulated and real robot. The same code can be re-used to run on a real robot as in simulation.

To recreate the actual environment, we designed several new models and plugins for `Gazebo`. We will define the terminology before proceeding to describe the simulation environment. It must be noted that designing the simulated environment was done before we had access to the real hardware, hence there is a small difference in the simulated part and pallet. The following concepts are taken from `Gazebo` tutorials [1]

- **World files** The world description file consists of all the elements in a simulation, including robots, lights, sensors and static objects. The file is formatted using the Simulation Description Format(SDF), and typically has a `.world` extension. The Gazebo server reads this file to generate and populate a simulation environment.
- **Model files** The model file uses the same SDF format as world files, but it only contains one specific model. The purpose of these files is to facilitate model reuse and simplify world files. Several models are provided by Gazebos online model database, and even more can be sourced from the extensive community of Gazebo users.
- **Environment variable** Environmental variables are used to locate files, and set up communication between the Gazebo server and clients. A proper software workspace will generate this set up automatically, but it can also be done manually by the user.
- **Gazebo server** Through a command line the server parses a world description file and then simulates the world using a physics- and sensor engine. The server generates all the raw simulation data, but it does not include any graphical representation.
- **Plugins** Plugins provide simple and convenient interfaces with Gazebo. They can be loaded though command lines, or by modifying a world/model

[1] http://gazebosim.org/tutorials

file. Most plugins are loaded by the server, but some can be loaded by the client.

The scenario to recreate is shown in the Figure 1.2. The `Gazebo` simulated scenario is shown in Figure 4.22.

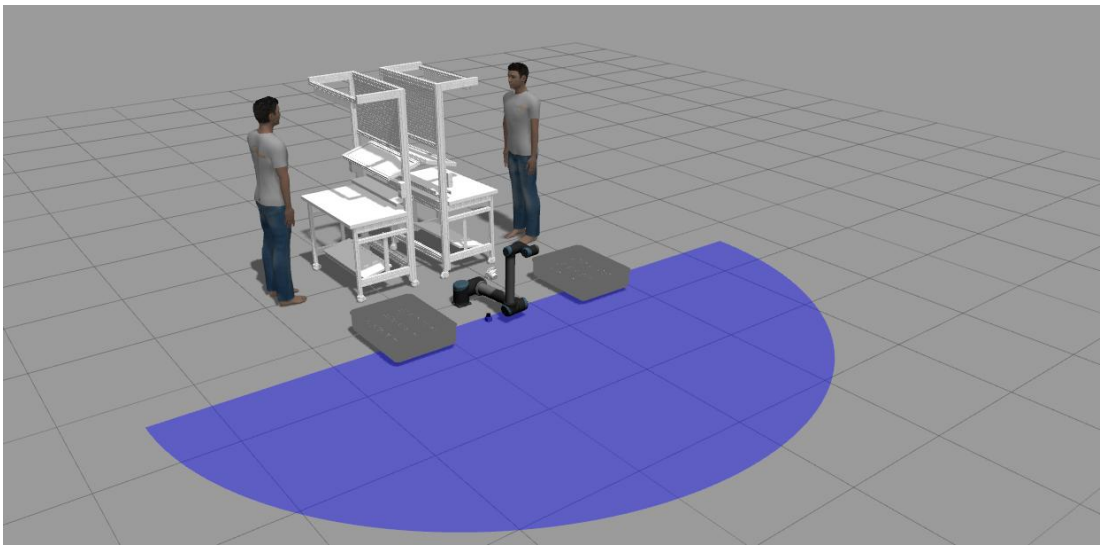


Figure 4.22: Scenario in `Gazebo`

- **World file** The world file includes the SDF models of the part, pallet, table, sensor and by default the ground plane and sun (lights). Note that these models must be present in the `~/.gazebo/models` folder.
- **UR10 Robot** The robot model is provided by ROS-I drivers. It has the same functionality as the real robot. While launching the `roslaunch` file, a parameter `sim` is set to `True` which enables to control the simulated robot.
- **Table, Part, Pallet** The table, part and pallet were 3D scanned and converted to `.STEP` file by the Design department at Schaeffler. For each model, the following structure must be followed in Figure 4.23:

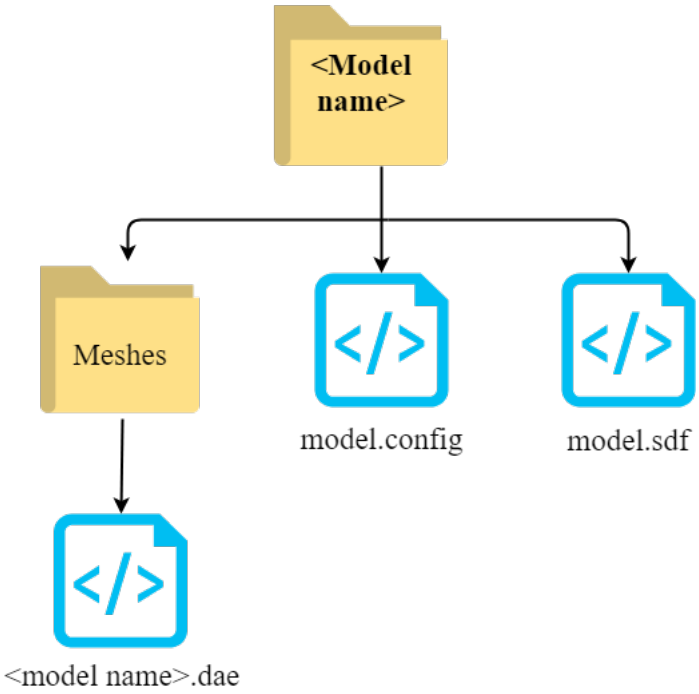


Figure 4.23: Gazebo model file structure

Hence it is necessary to convert the `.STEP` files to `.dae` format. Open-source 3D software called Blender [1] was used for this process. The `.config` and `.sdf` files use the standard XML format. The `.config` file essentially describes the model ,i.e in this case the Table. For instance it is shown in code block 4.1:

Similarly, the `.sdf` file describes several important parameters such as `<link>, <collisions>, <visual> and <pose>`. There are several other parameters, but these aforementioned have to defined by default. This is explained in code block 4.2.

Finally, the model is available in the Models tab in the `Gazebo` GUI. The snapshot of the models is presented in Figure 4.24.

- **Omron Sensor:** In the Figure 4.22, the blue area represents the area covered by the safety sensor called Omron Sensor. It is to mimic the usage of a laser scanner that detects obstacles such as humans walking into the robot workspace. We developed a `Gazebo` plugin to perform this action. Following the steps above, the 3D model is imported into `Gazebo` as seen in Figure 4.24 (D). Once the SDF is created, a sensor must be added to generate

[1] https://www.blender.org/

```
<?xml version="1.0"?>

<model>
  <name>test_table</name>
  <version>1.0</version>
  <sdf version='1.5'>model.sdf</sdf>

  <author>
    <name>Prajval</name>
    <email>prajval10@gmail.com</email>
  </author>

  <description>
    Table in shop-floor
  </description>
</model>
```

Listing 4.1: Writing the `.config` file

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="test_table">
    <static>true</static>
    <link name="link">
      <pose>0 0 0 0 0 0</pose>
      <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://test_table/meshes/test_table.dae</uri>
            <scale>0.55 0.55 0.55</scale>
          </mesh>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://test_table/meshes/test_table.dae</uri>
            <scale>0.55 0.55 0.55</scale>
          </mesh>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```

Listing 4.2: Writing the `.sdf` file

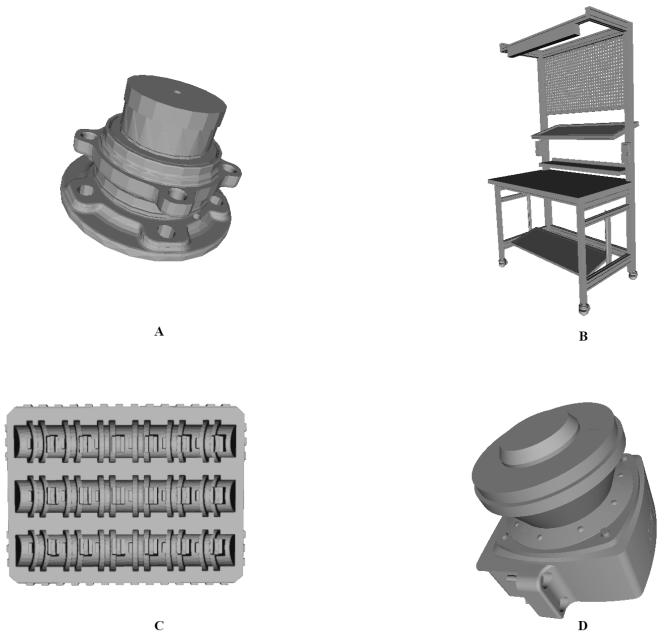


Figure 4.24: `Gazebo` Models (A) Part (B) Table (C) Pallet (D) Omron Sensor

data, from the environment or a model. A *ray* sensor in `Gazebo` consists of one or more beams that generate distance, and potentially intensity data.

A ray sensor consists of one `<scan>` and one `<range>` SDF element. The `<scan>` element defines the layout and number of beams, and the `<range>` element defines properties of an individual beam.

Within the `<scan>` element are `<horizontal>` and `<vertical>` elements. The `<horizontal>` component defines rays that fan out in a horizontal plane, and the `<vertical>` component defines rays that fan out in a vertical plane. These values are plugged in referring to the Omron sensor specifications [1].

Furthermore, a `Gazebo` plugin is created to publish the distances detected by the Omron laser scanner on `\scan` ROS topic. The *Robot Interface* spins a thread which continuously polls this topic to check if the obstacle distance is less than prescribed safety minimum (3m). If so, it immediately stops the robot. A video on the simulation can be seen here: https://goo.gl/dVMcAh.

[1] http://www.ia.omron.com/products/family/2717/specification.html

4.2.5.3 URSim

Universal Robots has created its own offline simulator called URSim. It mainly runs on Linux systems but can also be run in Windows inside a custom Virtual Machine.

URSim is a 3D robot manipulator simulator with the same GUI, called Polyscope, that is used with the physical UR robot manipulators (Figure 4.25). The simulation software comes with preexisting models of the UR3, UR5 and UR10. The major advantage of using the URSim: it guarantees that kinematics and dynamics of the robot is correct. Essentially, the real robot will perform identically as the URSim robot. However, a practical downside is URSim lacks a proper support community.

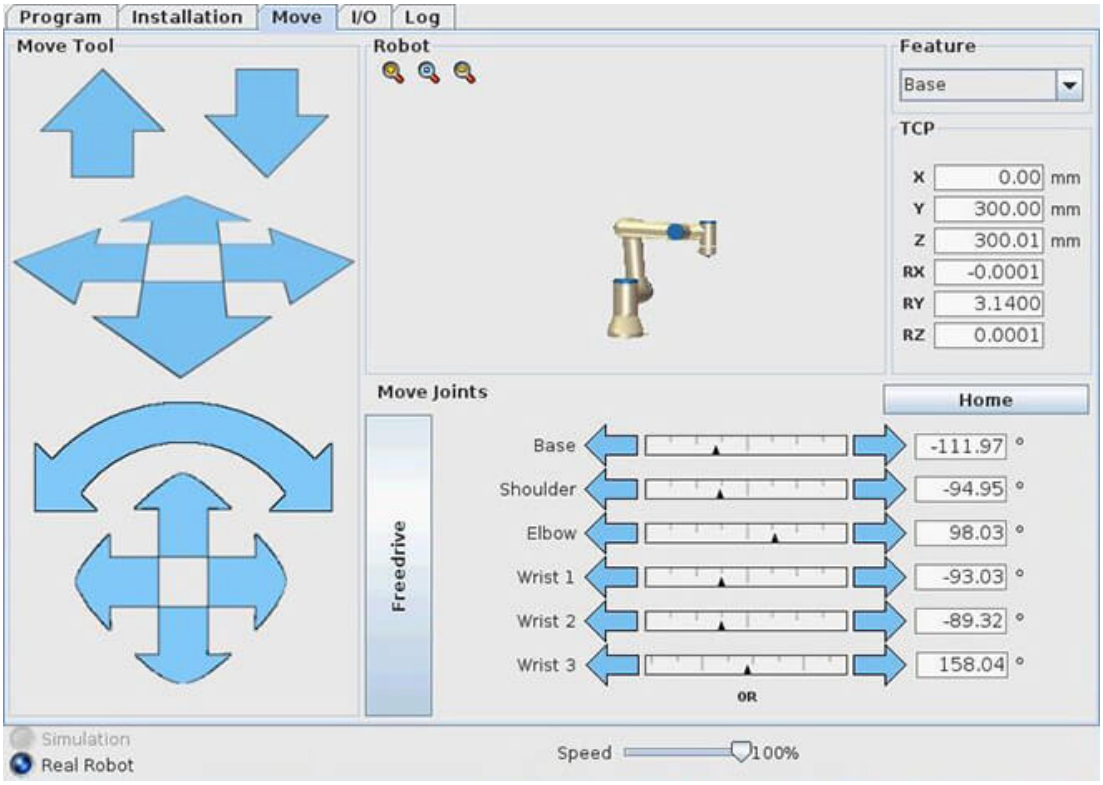


Figure 4.25: URSim Simulator

It is also possible to connect our algorithms developed in ROS with the URSim. While launching the ur_modern_driver, we must specify the IP address of the robot. In case of the real robot, we specify the IP address set in the PolyScope

of the real robot. Also, this is the default setting. In case we need to use the `URSim`, we change the parameter : `robot_ip := localhost` (In case the `URSim` is running on same host machine as the ROS master).

In conclusion, we use `Gazebo` to **test** the robot trajectories and `URSim` to **validate** the trajectories. This two-step verification is done at each stage before we commit to test the algorithm on the real hardware.

4.3 Integration

This section will explore how all aspects of the architecture is glued together and the various communication channels being used.

4.3.1 Implementation of Task Representation

The scenario in the production line at Schaeffler entails a pick-and-place palletizing task involving human-robot cooperation. We represent the cooperation task as a single-long AND/OR graph illustrated in figure 4.26.

Consider the pallet consists of n positions where parts need to be placed. We will choose a pallet where $n = 15$, consisting of 3 columns and 5 rows. The **root node** N_r of the graph G is *pallet_full*, i.e. all positions of the pallet are completed. The **leaf nodes** of G are *part 1* and *pallet_empty*. All other nodes except the leaf nodes are labelled *unfeasible*. At the beginning of the cooperation, the *leaves*: *part 1* and *pallet_empty* are labelled *solved*. The parent node of the leaf nodes,i.e *part_&_pallet_1* is labelled *feasible*. It must be note that however every part is identical to each other, each part has to be placed in the pallet in a specific sequential order. Hence we number the incoming parts so we know their position on the pallet.

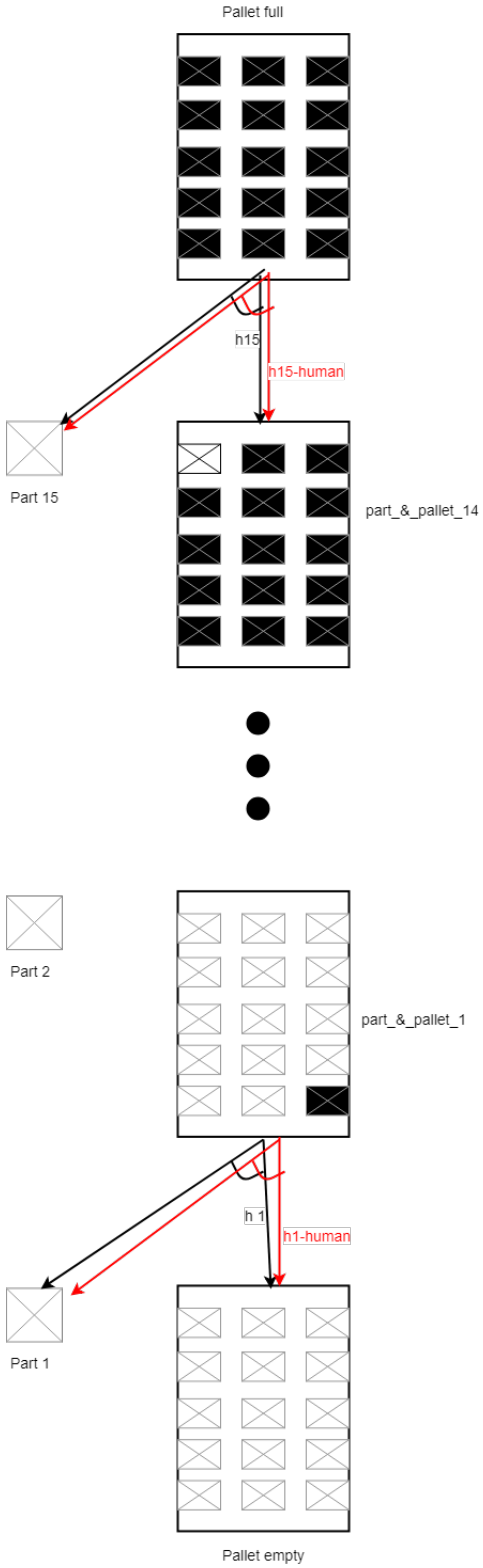


Figure 4.26: Proposed AND/OR graph structure

Every non-leaf node has *two* hyper-arcs *hi* and *hi_human* where $i \in [1, 2, \ldots, n]$. The hyper-arcs *hi* and *hi_human* establish *or* relationship between them and *and* relationship with their *child* nodes. As we can see from Figure 4.26, we have two cooperation paths, P_{black} and P_{red} and corresponding cooperation models M_{black} and M_{red}. By definition, the hyper-arc *hi* corresponds to the performing the pick-place task through *human-robot collaboration*. While the hyper-arc *hi_human* corresponds to the complete task performed by *human* alone. This will be explained in detail in section 4.3.2. As we can see, the *actions* corresponding to each hyper-arc need not be explicitly known to the AND/OR graph. Each hyper-arc and corresponding cooperation model has an associated *traversal cost*. To achieve Objective O1., we realize that to promote ergonomy of human-operators, they must avoid picking and placing of heavy parts. Hence, we must assign *hi_human* (or M_{red}) a much higher *cost* compared to *hi* (or M_{black}).

```
pallet_assembly 3 part_&_pallet_1
part_&_pallet_1 1
part_1 1
pallet_empty 1
h1 2 part_&_pallet_1 1
pallet_empty
part_1
h1_human 2 part_&_pallet_1 5
pallet_empty
part_1
```

Listing 4.3: Implementing the AND/OR graph in code

In the code block 4.3 presented, we demonstrate how the AND/OR graph is encoded for one single part-palleting task. Right at the top, the cooperation task name and the root node is written. This is followed by listing the *nodes*. Similarly the hyper-arcs *h*1 and *h*1*_human* are defined. It is important to note the *cost* provided to each hyper-arc: *h*1 assigned a cost 1 while *h*1*_human* is assigned cost 5. This follows from the discussion above that the cooperation path involving *human alone* must be least *optimal*. That is, the *hi_human* hyper-arc is executed only when *hi* hyper-arc is *infeasible*.

4.3.2 Implementation of Task Planner

As described in Section 3.2, the role of the *Task Planner* is deciding which agent performs which tasks. The *Task Representation* decides which is the current optimal node and hyper-arc and provides this information to the *Task Planner*. The

Planner executes each action corresponding to the Node and Hyper-arc.

We will consider actions corresponding to only hyper-arcs. The nodes do not have any actions themselves. So in Code Block 4.3, $h1$ corresponds to the hyper-arc transforming nodes `pallet_empty` and `part_1` into `part_&_pallet_1`. The hyper-arc $h1$ has corresponding actions which must be performed successfully in sequence to reach the next state. If any of the actions in $h1$ are unsuccessful, the *Planner* informs the *Task Representation* module. The alternative optimal hyper-arc is found by the AND/OR graph, in this case $h1\_human$. The *Task Representation* module informs the *Planner* to perform actions corresponding to $h1\_human$ hyper-arc. In general, this is extended for hi and hi_human, where $i \in 1, 2, \ldots, 15$.

Note: In theory, if actions corresponding to hi_human return unsuccessful, there is no feasible state and the cooperation task fails. This leads to our assumption that human operator will not intentionally fail the actions corresponding to hi_human. These actions and assumptions will be explained further in Section 4.3.4.

The actions are specified to the *Planner* through the `State Action List` and `Action Definition List`. More clearly, the `Action Definition List` defines all the actions that can be performed by every agent in the following way:

```
[action name]
[how the action should be done: "single" agent, "all" agents]
[Number of agents who can perform the action]
[agents name who can perform the action]
[action type: if it is simple, or complex: another andor graph].
```

For the present thesis, the `Action Definition List` can be expressed as follows:

The `simple` keyword denotes the action can be performed by the responsible agent and it is not another AND/OR graph. Furthermore, the actions defined here are very self explanatory to the user. No underlying details such as robot motion planning or manipulation is provided to the *Planner*.

Thus, the architecture decouples action planning from robot motion planning and control.

The `State Action List` defines the list of actions that must be executed

```
Grasp simple Robot
UnGrasp simple Robot
GotoObject simple Robot
startPose simple Robot
approachObj simple Robot
intGoal simple Robot
intGoal1 simple Robot
approachGoal simple Robot
goalPose1 simple Robot
Handover simple Human
PickPlace simple Human
```

Listing 4.4: Action Definition List

sequentially for each hyper-arc. The Code Block 4.5 shows the `State Action List` for $h1$ and $h1\_human$ hyper-arcs. The same is extended to other hyper-arcs.

```
h1 UnGrasp startPose Handover approachObj GotoObject Grasp approachObj intGoal
    approachGoal goalPose1 UnGrasp approachGoal intGoal startPose
h1_human Handover PickPlace
```

Listing 4.5: State Action List

It is interesting to note that the sequence of actions defined in 4.5 are in general valid for all handover and pick-place Human-robot collaboration tasks. There are no robot specific information encoded in the *Planner*.

It can be concluded that hardware independent robot skills facilitates easy instruction by users with no robotic experience, enables skill reuse and allows deployment on different hardware platforms.

4.3.3 Implementation of Robot Interface

The function of the *Robot Interface* is to receive the robot commands from the *Planner* and classify if it is a **Manipulation**, **Motion** or **Object Retrieval** action. The possible actions that can be performed by the robot are as follows:

- **Manipulation action:** This action involves grasping and ungrasping the part by the gripper. The *Robot Interface* uses the Gripper service described in 4.2.4.1 to grasp/un-grasp the part and receive feedback if the grip has been detected in case of a stable grasp. Once the action is complete, the *Robot Interface* provides feedback to *Planner*. Referring to 4.5, these are `Grasp` and `UnGrasp` actions.

- **Motion action:** This action involves robot motion from current position to goal position. We will use a pre-defined motion trajectory after the robot has grasped the part. The waypoints are taught to the robot using kinesthetic teaching by an expert and waypoints values in joint space are stored in a `database` text file. The waypoints to be taught are the `StartPose`, `ApproachPose`, `Intermediate-goal`, `Goal-approach` and multiple `GoalPoses` for each corresponding point in the pallet. It is important to note that names of waypoints in `Database` must be identical to action names stored in `Planner`. When the action is received from the *Planner*, the *Robot Interface* queries the database for the corresponding joint values. This is defined as the `Joint Trajectory Goal`. The *Robot Interface* checks if the current joint values are not same as the goal values. If so, it sends the `Joint Trajectory Goal` to the `ur_modern_driver` to upload to the robot. The action is completed when robot `current joint values = goal joint values`. The feedback is sent to *Planner* as action successful.
- **Object Retrieval action:** This action involves moving the robot to object position. The *Robot Interface* uses the IK service described in section 4.2.4.1. Once the object pose is received in robot frame from the *Human Interface*, the *Robot Interface* calls the IK service sending service request with x and y positions of the object. The robot moves to the requested position and feedback is sent to *Planner* as action successful. From 4.5, these refer to `GotoObject` action.

Furthermore, in all of the above actions in case the robot is protective stopped, required goal is out of robot workspace or object cannot be grasped, an action **failure** feedback is sent to *Planner*. The *Planner* then recomputes the next action to be performed.

In conclusion, the role of *Robot Interface* is to receive *action* commands in semantic format from *Planner* and send corresponding robot joint values to the robot while monitoring if the action is completed successfully.

4.3.4 Implementation of Human Interface

The function of the *Human Interface* is to communicate commands to the human operator through a User Interface and check for *Human Action Recognition*. As discussed in section 4.3.2, human operators can performs two possible actions:

- **Handover**: Inspect and handover the part to the designated grasp location (in camera field of view) in the robot workspace. When the object pose is

recognized by the camera, the Handover task is considered to be successful and feedback is sent to *Planner*.

- **PickPlace**: This task is done when the robot is *Protective stopped*. The operator picks up the part from the grasp location if the part is not grasped by the robot or from the robot gripper if the part is grasped. The operator then places the part in the pallet in the correct sequence. The task completion successful is done through the *User Interface*.

Assumption 1: To enforce Objective O1. we assume that the operator will NOT perform `PickPlace` action before `Handover` action, i.e the operator will not carry the heavy loads to the pallet himself but instead only perform the `PickPlace` when the robot is incapable of performing said task.

Assumption 2: Due to lack of sensors, we cannot clearly monitor the human task in case of `PickPlace` action. Hence it is assumed that human will not try to cheat the system by not placing the part in specified position or placing in another different position. During the `Handover` task, after the vision system has detected the part, the operator must not change the position of the part. We use an open-loop one-shot object pose recognition and grasping technique. Therefore, there is no information to the robot if the object pose has been changed dynamically by the operator.

The aforementioned assumptions are justified in the case of human-robot cooperation that both humans and robots will coordinate with each other to proceed towards the common goal.

The *Human Interface* is programmed in **Python** for easy handling of threads and fast development. There are two threads running in parallel. The main thread runs the *User Interface* and the secondary thread runs the *Human Interface* ROS node. The data is shared between the threads using **Queues** [1] and thread blocking-unblocking is done using **Events** [2].

4.3.4.1 User Interface (UI)

The UI is designed with Objective O2., to ensure a linguistic level of communication between the robot and human operator. The UI is designed with `TKinter`, Python's de-facto standard GUI package [3]. The human tasks are assigned by the

[1] https://docs.python.org/2/library/queue.html
[2] https://docs.python.org/2/library/threading.html
[3] https://wiki.python.org/moin/TkInter

Planner. These tasks are shown in the message text box in Figure 4.28. The human interacts by simple button presses such as `Start`, to start the cooperation task or `OK` to provide feedback after human completes a task. Furthermore, the operator has two language options for convenience: English (default) and Slovak.

4.3.4.2 Human Interface ROS Node

At the beginning, the main thread starts the UI and waits for the human operator to press `Start` button. After which, the secondary thread is spawned. The secondary thread initializes a `human_interface` ROS node and subscribes to \`human_command` ROS topic.

If the command received is `Handover`, it puts the message to the UI via a synchronized Queue. The UI publishes the message on the message text box. The Vision service is called by the ROS node. The camera is triggered every 1 second to check if part has been placed. If the vision system successfully returns the pose of the part, the ROS node sends `Handover_successful` to the *Planner* and also it is published on the UI message box.

Similarly, if the command received is `PickPlace`, the information is sent to operator to Enable the robot as it is *Protective Stopped*. The ROS thread waits for a `thread.Event` to be `set()`. The `thread.Event` is `set()` when the operator presses the `OK` button. By using `thread.Event` we ensure thread safety for synchronization. Continuing if the part has been grasped by the robot, the operator is asked to take the part from the robot gripper. This is again done using the `OK` click on the UI. The operator must place the part in the correct sequence in the pallet. If the part is not grasped, the operator must perform the same task by picking from grasp location and placing in the pallet. Finally, the operator is instructed to come outside the robot workspace and press the `OK` button before the robot continues to the next task. This further ensures industrial safety requirements that robot is not in motion when a human is in close proximity. The process can be seen in Figure 4.29.

In conclusion, the *Human Action Recognition* for `Handover` action is implicitly done by the Vision system and for the `PickPlace` action it is done via the User Interface.

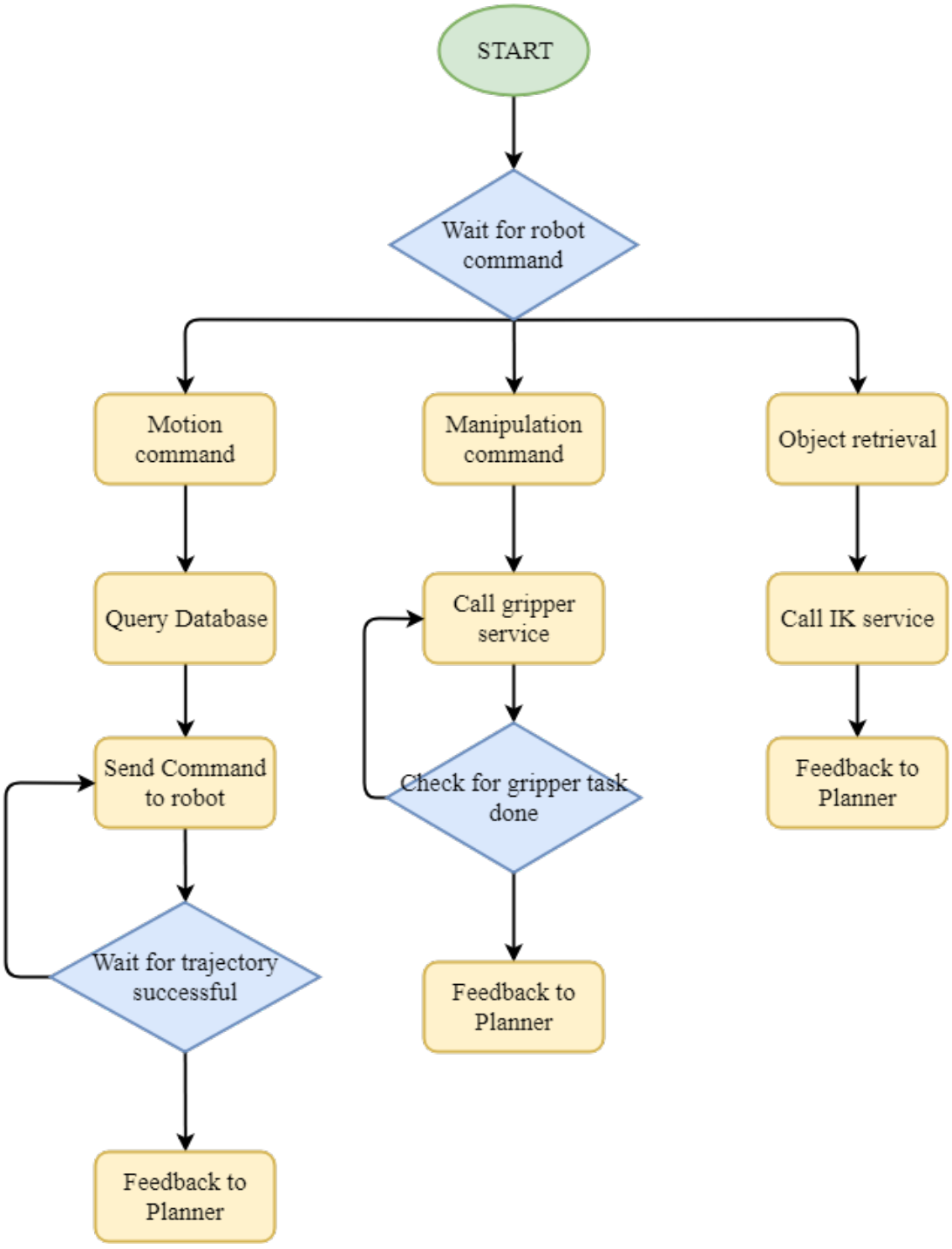


Figure 4.27: Flowchart of the *Robot Interface*

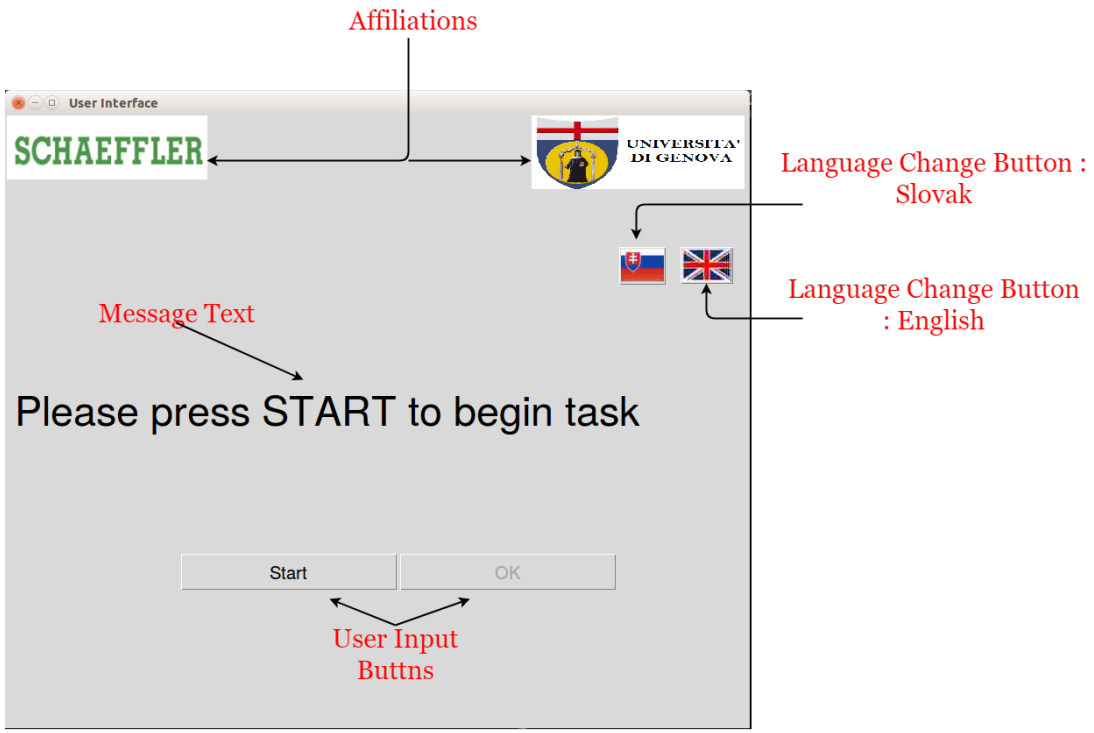


Figure 4.28: Description of the User Interface

Figure 4.29: Various states of the User Interface

4.4 Experiment

The experiments are carried out in the DOJO laboratory [1] in the Special Machine Building (SMB) Department at Schaeffler, Kysuce. The laboratory setup is shown in Figure 4.30. The external workstation is using ROS Indigo based architecture in Ubuntu 14.04 operating system running on 64-bit Intel i7 processor with 8GB RAM.

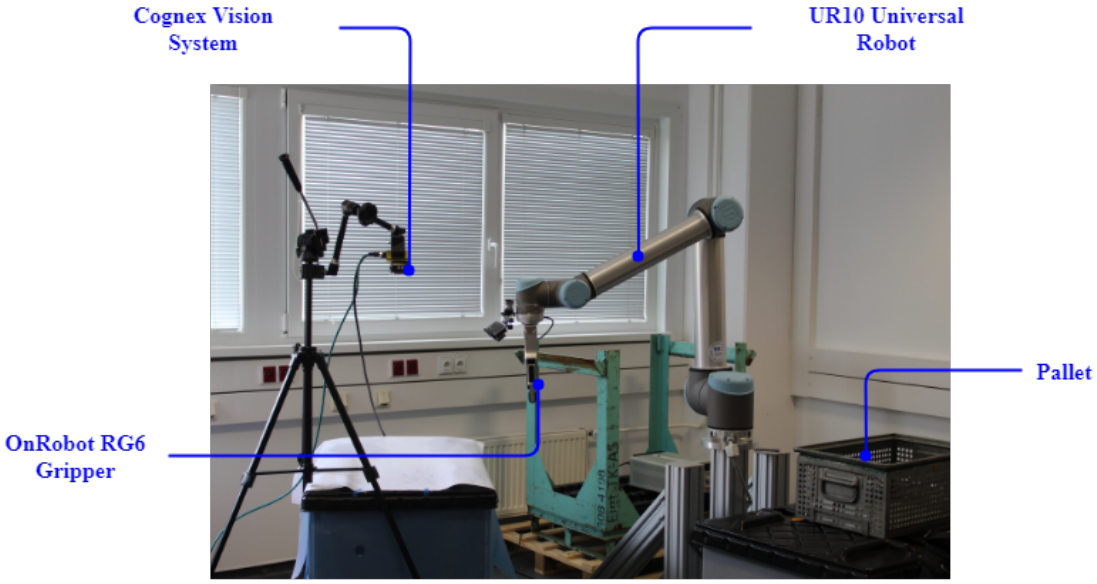


Figure 4.30: Complete hardware setup

The top view of the laboratory setup is shown in Figure 4.31. The dimensions are also shown. Note that robot workspace and human workspace is shown in different colours for illustration, the human can enter the robot workspace and touch the robot during motion. Since it is not possible to setup a conveyor in the laboratory, we use a standard box as the pickup location for the operator.

The experiment involves 4 sub-experiments detailed as follows:

1. **Manual execution:** The operator takes part from the pickup box. He does the visual and tactile inspection. He places it in sequence inside the pallet box. This is repeated until the entire layer of the pallet box is complete. The robot is not used in this task.

[1] https://www.lean.org/balle/DisplayObject.cfm?o=3032

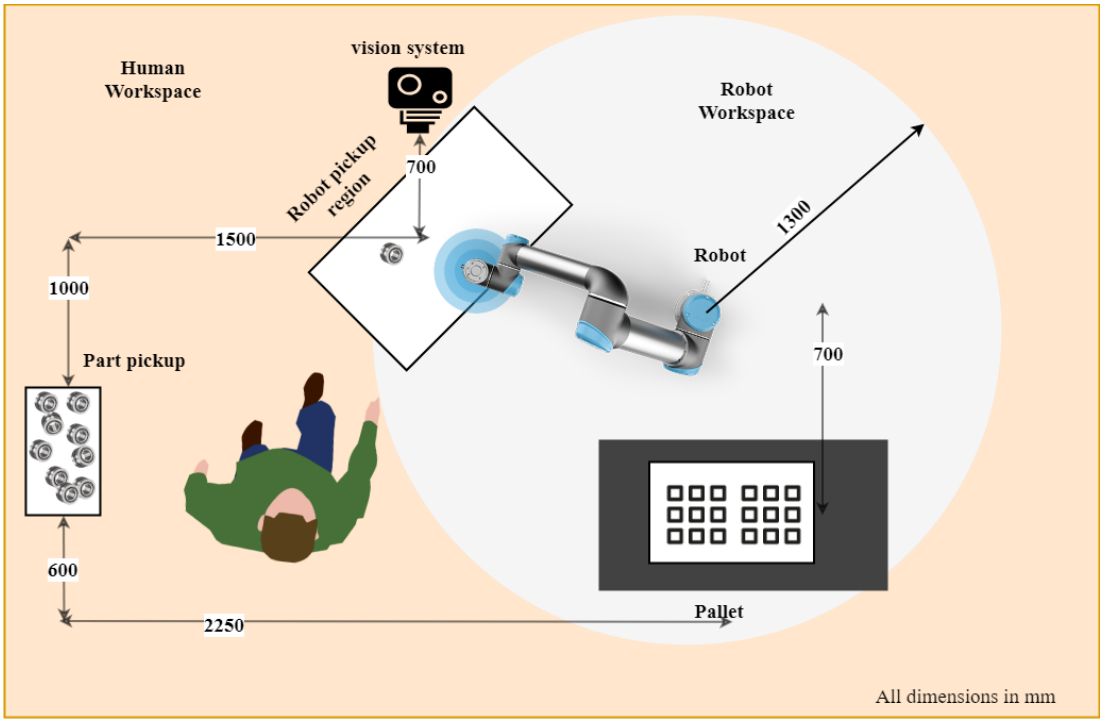


Figure 4.31: Top view of the laboratory setup

2. **Handover and robot pick-place:** The operator picks up part from the pickup box. He performs visual and tactile inspection. He places it in the *robot pickup* area in the field of view of the vision system. The robot performs pick-place in pallet box. The task is repeated one-by-one until the pallet layer is complete.

3. **Robustness:** The same task as above is performed by the operator but instead of placing parts one by one to the robot pickup region, we simulate that parts are arriving at a fast rate to the operator and he places multiple parts in the robot pickup region simultaneously.

4. **Operator intervention:** We simulate the action when the robot is executing the pick-place task and the operator walks into the robot workspace and stops the robot motion by exerting force on the robot body. Once the robot is stopped, the operator interacts with the *User Interface* and takes the part from the gripper and places it in the correct position in the pallet box. The robot continues with next part. The operator intervention is performed at random in between the execution of sub-experiment 2.

The experiments are shown in sequence below in Figures 4.32- 4.35.

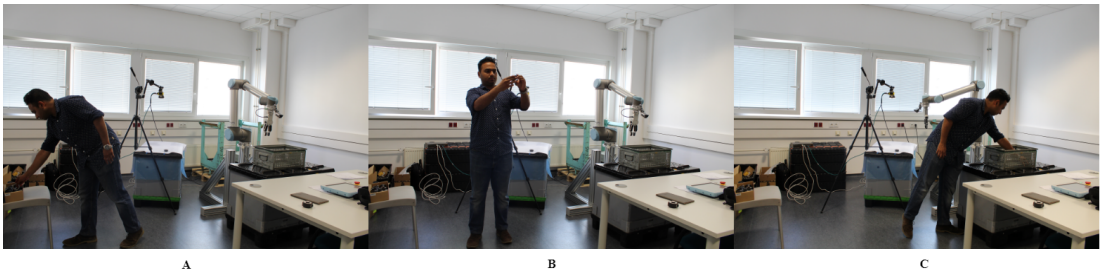


Figure 4.32: Manual execution: (A) Pick up part (B) Visual and tactile inspection (C) Place the part

The Figure 4.32 depicts the manual execution of the task. No AND/OR graph or Planner is used for this task.

Figure 4.33: Handover and robot pick-place: (A) Visual and tactile inspection (B) Place the part in robot pickup area (C) Robot approaches object (D) Robot grasps object (E-H) Robot transports the object to the pallet (I-K) Robot moves back to start position and process continues

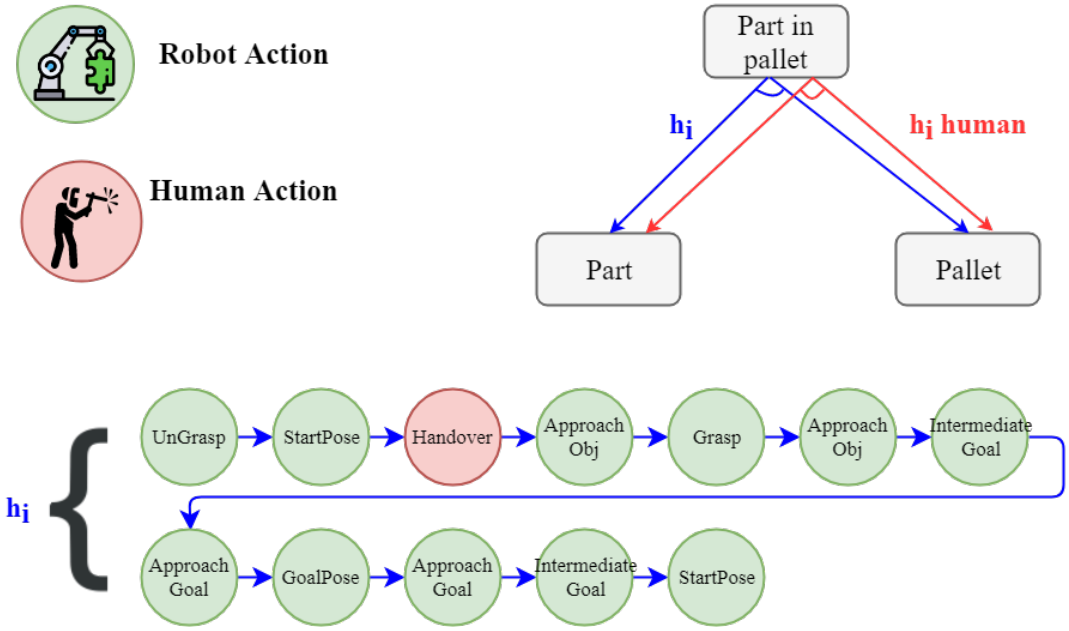
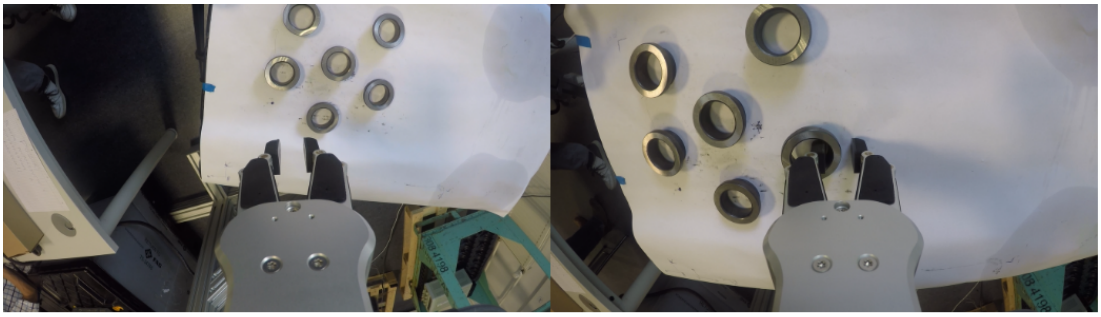


Figure 4.34: Handover and Robot pick-place actions shown using the AND/OR graph

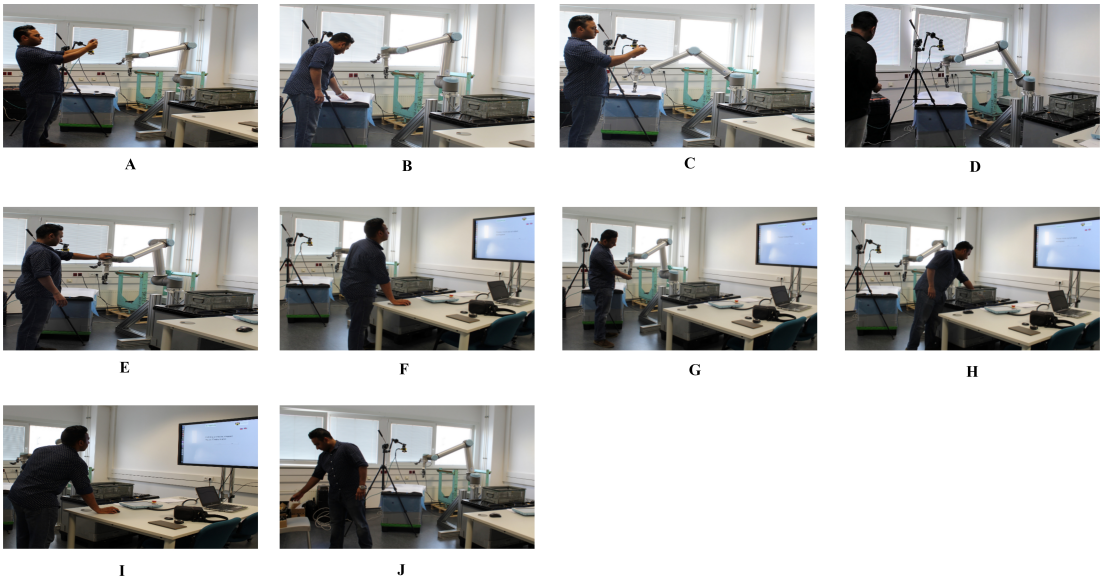
The Figures 4.33 and 4.34 depict the process of h_i hyper-arc in the AND/OR graph. The actions shown in the hyper-arc is sequentially performed. Referring to Figure 4.34, robot performs `UnGrasp` which opens the gripper and moves to `StartPos` shown in Figure 4.33 (A). The operator performs the `Handover` action in Figure 4.33 (B). The robot performs `ApproachObj` by moving above the object in Figure 4.33 (C) and performs `Grasp` in Figure 4.33 (D). The robot moves to an intermediate waypoint by performing action `IntermediateGoal` in Figure 4.33 (F) and moves to `GoalPose` via `ApproachGoal` actions in Figure 4.33 (G-H). The operator continues with the next part while the robot returns to `StartPose` through actions `ApproachGoal` and `IntermediateGoal` in Figures 4.33 (I-K).

The Figure 4.35 shows the robustness experiment where the operator places multiple parts simultaneously in the robot pickup area and the robot performs pick-place one after the other. It must be noted that with respect to the AND/OR graph, there is no difference between the robustness experiment and the previous handover and pick-place experiment.



A B

Figure 4.35: Robustness: (A) Robot waits for vision module to provide grasp location among different parts (B) Robot goes to grasp the part



A B C D

E F G H

I J

Figure 4.36: Operator intervention: (A)-(D) Perform inspection and handover part (E) Stop the robot by exerting force on joints (F) Interact with UI, enable the robot (G) Retrieve part from robot (H) Place the part in pallet box (I) Move out of workspace and interact with UI (J) Continue with next task

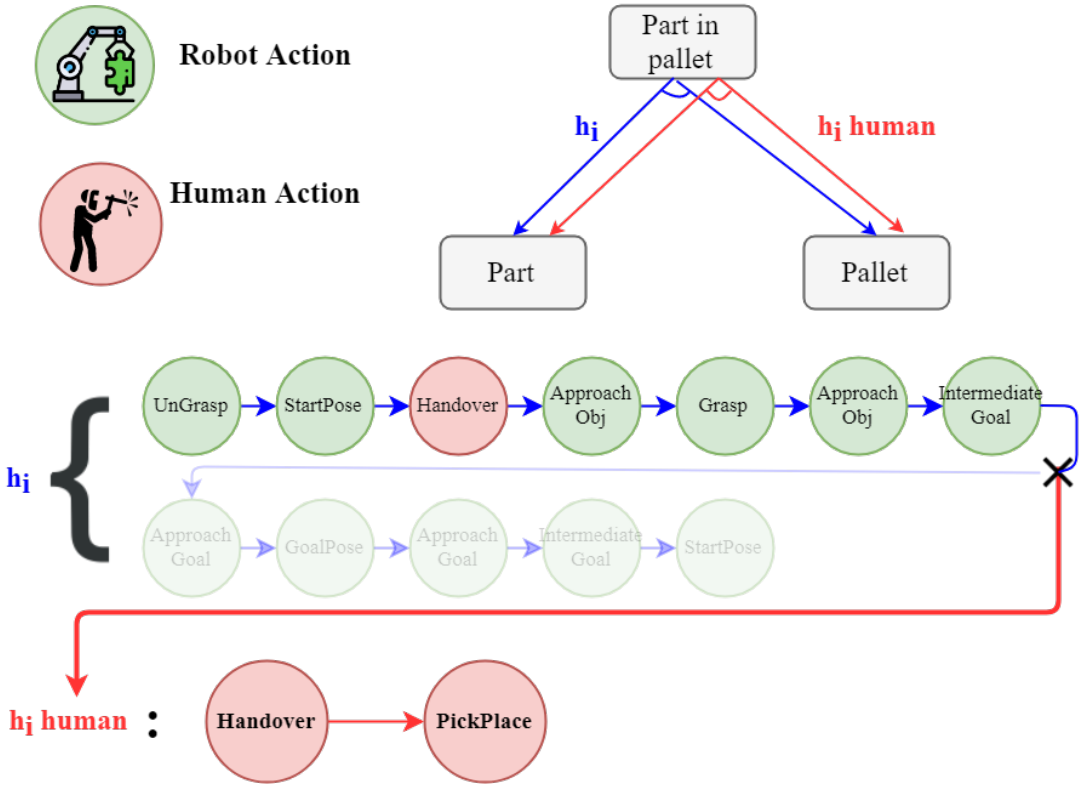


Figure 4.37: Operator intervention actions shown using the AND/OR graph

In the Figures 4.36-4.37, the human intervention experiment is shown. It is interesting to see the process through the AND/OR graph perspective. Initially the operator performs the same visual inspection and handover actions as in hyper-arc h_i in Figures 4.36 (A)-(D). The operator then wants to intervene and stops the robot by exerting force on the joints as in Figure 4.36 (E). The robot is protective stopped and the *Robot Interface* informs the *Planner* about the failure of the motion task. The *Planner* performs the online search in the AND/OR graph for the next feasible hyper-arc which is h_i*_human*. Hence the next action to perform is `PickPlace` in hyper-arc h_i*_human*. The operator enables the robot and interacts with the *User Interface* depicted in figure 4.36 (F). The UI instructs the operator to take the part and place in the pallet in correct sequence to finish the task. The gripper opens after the operator presses a button and he completes the task as in figures 4.36 (G) and (H). The UI informs the operator to move away from the workspace and the robot goes to `StartPos` for the next iteration (Figure 4.36 (I)-(J)).

A complete trial is when the whole layer of the pallet box is completed while a partial trial is defined as when the pallet layer is partially complete, such as 5 out of 15 parts placed in pallet. The authors conducted 25 complete trials and many more partial trials. We also performed experiments with 10 employees for Schaeffler, comprising of production line workers and line supervisors. All the volunteers were male and between age of 29 to 45. Few had experience working with robots prior to the experiment. The volunteers were provided two Likert-scale questionnaire (pre-experiment and post-experiment) which provides subjective evaluation criteria for the HRC experiments. The questionnaires are provided in Appendix A. Furthermore data regarding human action time, robot action time, time taken by *Planner*, human idle time, robot idle time and overall time for execution is also recorded. These provide objective evaluation criteria.

Through the experiments we seek to evaluate the following hypotheses:

H1. After the experiment, the volunteers would feel less intimidated of working alongside a robot. For this hypothesis, we will use subjective measurements through Likert-scale questionnaires before and after the experiment.

H2. On average, time taken to complete the full task would take longer with Human-Robot Collaboration than Human-alone. For this hypothesis, we will use objective execution time measurements.

H3. Operators will feel less stressed physically and mentally when collaborating with a robot co-worker than manually executing the task. For this hypothesis we will use subjective measurements

H4. The architecture will provide more flexibility to the operator during HRC. For this hypothesis we will use subjective measurements.

H5. The architecture will lead to a natural and fluent interaction between the robot and human. For this hypothesis, we will use a combination of subjective and objective measures.

In this chapter, we presented the hardware setup of the experiment by explaining the hardware specifications of the part, pallet, robot, vision system and gripper. We also presented our choice for the software by juxtaposing the pros and cons of the various options to program the robot. Subsequently, we explored the techniques used to integrate the various modules such as *Task Representation*, *Task Planner*, *Robot Interface* and *Human Interface*. Finally, we explained the procedure for conducting the experiment along with defining the experimental hypothesis. In the following chapter, we will explore the results from the experiments and draw conclusions for the thesis.

Chapter 5

Results and Discussion

In this chapter, we evaluate the conducted experiments based on quantitative/ objective measures and qualitative/subjective measures.

5.1 Evaluation

For clarity, we reiterate the definition: *complete task* is palleting 15 parts in the pallet box to complete the layer, *sub-task* is palleting of one part. By definition, the complete task is successful if *all* parts are properly placed in the pallet box and by extension each sub-task is successful if the part is placed in the correct position in the pallet box.

We performed 25 experimental trials of the complete tasks. Among the 25 experimental trials, 12 experiments were of type 2 i.e handover of parts *one-by-one* and robot performing pick-place; 8 experiments involved handover of multiple parts (upto 4) to the robot pickup area, following type 3; 5 experiments involved operator intervention at random stages of the sub-task, following the type 4. There were a total of 6 failed trials. The reason for the failures were: failure to properly pick-up the object due to combination of vision system and grasping module (4 trials), the robot dropping the part during transportation due to improper grasp (1 trial) and human error during the handover stage - the part was accidentally moved after the vision system sent the pose parameters to the robot (1 trial).

Furthermore, to test for robustness (experiment 3), we performed a separate trial of progressively placing 3, 6 and 10 parts. The results are shown in Table 5.1. The success rate of grasping progressively decreases while settling to around 60% for 10 parts. This happens due to various reasons: incorrect pose recognition

by the vision system if the object is placed far away from the world origin frame, ambient illumination, and failure to grasp if parts are very close to each other. To improve robustness, a closed-loop control with feedback from vision system may be used instead of opting for a open-loop one-shot pose recognition as in current system. Furthermore, a camera attached to the tool frame may increase the accuracy of the pose recognition and grasping.

No of parts	Success (%)
3	100
6	66
10	60

Table 5.1: Robustness test

5.1.1 Quantitative Metrics

This section deals with discussion regarding the quantitative metrics namely the overall cooperation time, human and robot action time, and fluency of the task. On average for the 25 cooperation experiments, the Table 5.2 shows the percentage of time needed by the human operators T_h, the robot T_r and Planner T_{plan}. Assuming a sequence of n actions, T_{plan} is defined as the sum of all such $n-1$ contributions (the first being set by default on the optimal path), and each contribution is given by the difference between the time T_{next} when the next action a_i suggestion is ready and the time T_{ack} when an acknowledge for a previous action $a_i 1$ is received, such that:

$$T_{plan} = \sum_{i=2}^{n} T_{next}(a_i) - T_{ack}(a_i - 1) \tag{5.1}$$

The T_h refers to the amount of time the human action is recognized by the vision system. Hence this is not a measure of all the actions done by the human. It is computed in the *Human Interface* module as :

$$T_h = \sum_{i=1}^{n} T_{resp}(a_i) - T_{recv}(a_i) \tag{5.2}$$

where $T_{recv}(a_i)$ is the time instant when the *Human Interface* receives the command from the *Planner* and $T_{resp}(a_i)$ is the time instant when the *Human Interface* sends back the response to *Planner*.

Similarly, T_r refers to the amount of time the robot is in action. It is computed by the *Robot Interface* as follows:

$$T_r = \sum_{i=1}^{n} T_{resp}(a_i) - T_{recv}(a_i) \tag{5.3}$$

Considering the distribution table 5.2, it is interesting to comment on the standard deviation (%) values. In general, the standard deviation (σ) shows the dispersion of the data around the mean value. The lower σ means the values are closer to the mean while a higher σ means a wider range of values. The σ_{plan} for the planner is very low meaning that it takes almost identical time duration during every trial. This shows the temporal deterministic nature of the *Planner*. In contrast, the standard deviation is higher for the human action time and robot action time. This is understandable as the operators takes different amounts of time to perform each sub-task and the robot action time also varies whenever there is operator intervention.

Measure	**Average Time (s)**	**Percentage of Total Time (%)**	**Std (s)**	**Std (%)**
Total Cooperation Time	481.82		24.34	
Human Action Time (T_h)	77.75	17.97	19.2	3.72
Robot Action Time (T_r)	401.57	81.52	10.19	3.51
Planning Time (T_{plan})	2.490	0.51	0.164	0.042

Table 5.2: Cooperation time distribution average and standard deviation (std) for successful tasks

The time distribution is also shown in the Figure 5.1 for the handover and robot pick-place task and Figure 5.2 for the human intervention task. Its noteworthy that the former is sequential processing of tasks, in other words: co-action. Whereas the latter is a collaboration task, illustrated by the marked circles showing both human and robot actions are running at the same time instant.

For one sub-task, the Figure 5.3 shows the time distribution between robot manipulation, robot motion and vision system. Its noteworthy that robot motion takes most of the chunk of the time, hence to speed up the overall process, the motion speed may be increased depending on safety regulations.

Finally we will look at the comparison between manual execution and human-robot collaboration in Table 5.3. We can see from the total execution time that human-alone task execution is faster than Human-robot collaboration, thus proving our experiment hypothesis H2.. Secondly, Team fluency is related to task

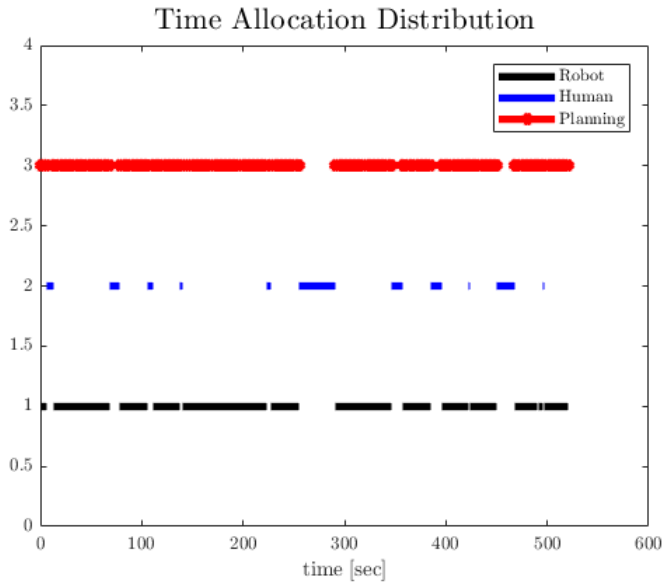


Figure 5.1: Time distribution of handover and robot pick-place task

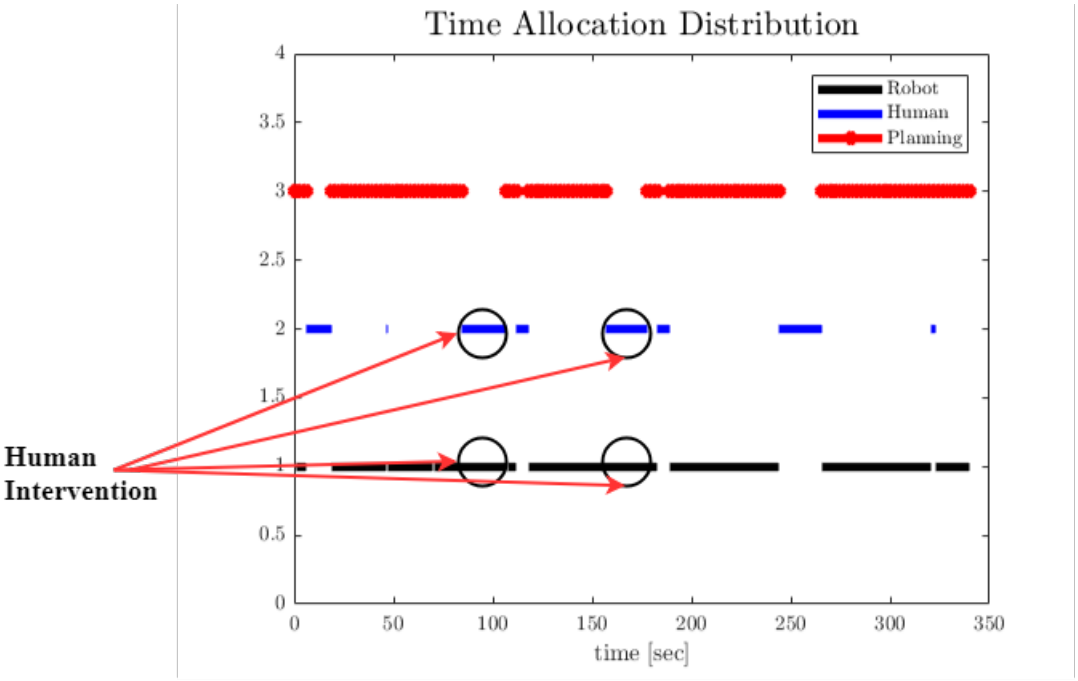


Figure 5.2: Time distribution of task involving operator intervention

efficiency, defined simply as the inverse of the time it takes to complete identical tasks or subtasks [Hoffman (2013)]. To measure fluency, we use a combination of

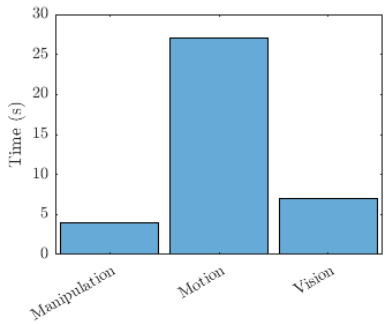


Figure 5.3: Time distribution between the different modules

Property	Human-alone	Human-Robot Collaboration
Total execution time (mins)	3.15	7.83
No# tasks allocated to human	7	5
No# tasks allocated to robot	0	2
Max weight handled by robot (kg)	0	0.1-5.5*
Max weight handled by human (kg)	0.1-5.5*	0

Table 5.3: Comparison between human-alone vs HRC. $*$ The maximum weight that can be handled by the system.

metrics : subjective metrics, which are based on peoples perception of the fluency of an interaction; and objective metrics, which can quantitatively estimate the degree of fluency in a given interaction.

From the Figure 5.1, we see the only time an agent is not performing a task is during planner action, which takes only about 0.54% of entire time. So effectively, there is no situation when all agents are idle during the task cooperation. Sometimes the human finishes his task and waits for the robot action to complete. This can be improved by making the robot move faster but within safety limits for collaboration or using sensors such that robots can operate at full speed when humans are far away and slow down when humans approach the robot workspace. Overall, we can say our system ensures fluency in HRC task (Hypothesis H5.).

5.1.2 Qualitative Metrics

This section studies the responses received from the questionnaires given to volunteers of the experiments.

5.1.2.1 Pre-Experiment Questions

The preliminary set of questions asked to the volunteers before the experiment deals with understanding their outlook towards robot co-workers. Amongst the 10 volunteers, 5 had previously worked with robots (not necessarily collaborative robots) and the remaining 5 did not have any experience with a robot. The results are shown in Figure 5.4.

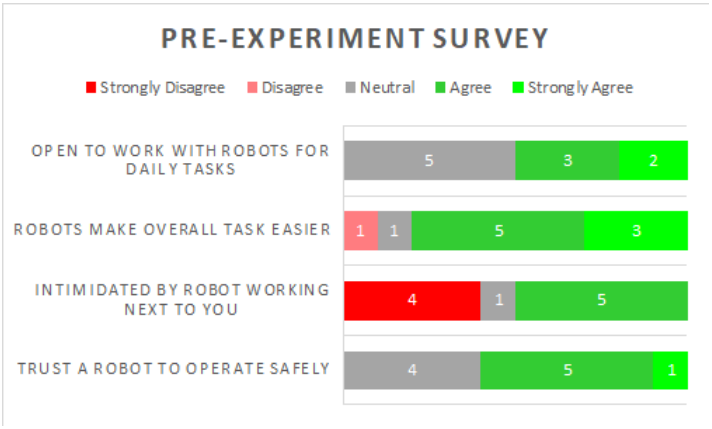


Figure 5.4: Pre-experiment questionnaire results

- Five people said they were **neutral** and the other five tended to **agree** with regards to trusting a robot co-worker to operate safely.
- When asked if they were intimidated by working alongside a robot, we found answers ranging between **agree** and **disagree**. It was also found that people who worked previously with robots tended to be less intimidated having a robot work alongside them without safety barriers.
- With regards to a robot making their overall task easier, general consensus was to **agree**. This meant that even though some were intimidated by the robots, they still consider them a valuable asset to productivity.
- Finally, we asked how open they were to adopt robots in their every-day lives. Half the people were **neutral** while the rest **agreed**, thus following the previous statement regarding finding robots useful.

What we can correlate from this pre-experiment survey is that in general people appreciated and understood that robots are valuable assets and would make their lives easier. However, some were still apprehensive about working next to a industrial robot without a safety barrier.

5.1.2.2 Manual Task Specific Questions

Following the pre-experiment questionnaire, the participants were asked to perform the manual pick-place task. Subsequently they were asked **Task specific** questions regarding the manual task. The results are shown in Figure 5.5.

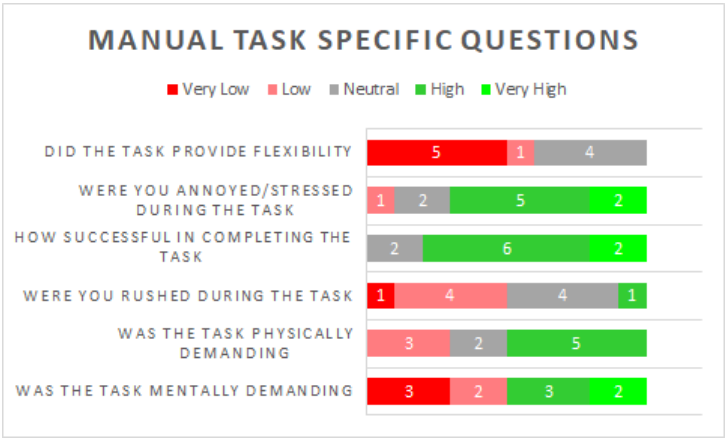


Figure 5.5: Manual Task Specific Questions

- When asked if the task was mentally demanding, we received all answers ranging from **Very Low** to **Very High**. Hence no proper inference could be made.
- In contrast, most of the volunteers seemed to think the task was physically demanding if they were to perform it for 8 hours/day.
- On average, most people felt **neutral** or a **little** hurried or rushed during the task.
- Everyone felt they were highly successful in completing the task.
- Similarly, most people felt the task was annoying or stressful.
- However, since the task had fixed constraints everyone felt the task did not provide flexibility to them.

5.1.2.3 Robot Specific Questions

Subsequently, the volunteers perform each sub-experiment 2-4: Handover and robot pick-place, robustness and human intervention. Due to time constraints, it was performed as partial trials for one section of the pallet box. We asked

each participant robot-specific and task-specific questions similar to the previous exercise. The results are shown in Figure 5.6.

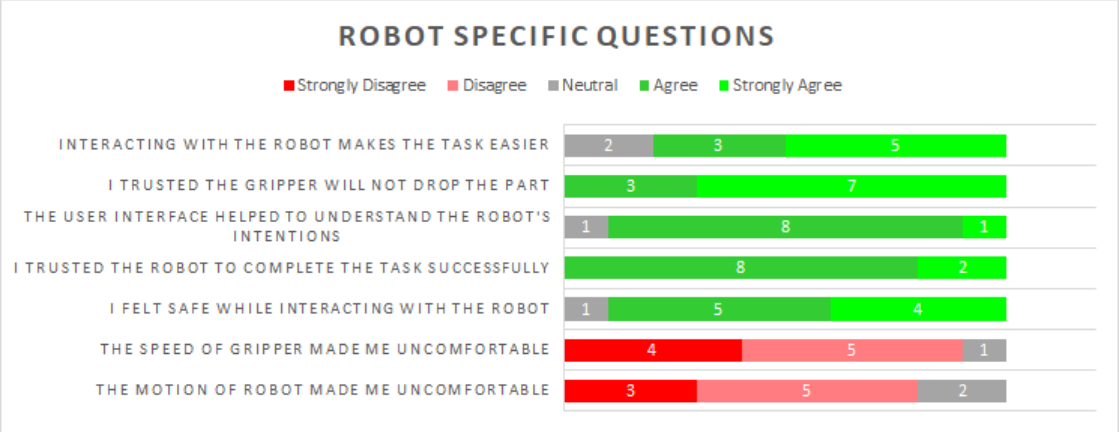


Figure 5.6: Robot Specific Questions

- Everyone disagreed that the robot made them uncomfortable. We can infer that everyone felt comfortable working next to the robot and coming into contact with the robot.
- Similarly everyone disagreed that the gripper made them uncomfortable. This is noteworthy as during intervention, the operator holds the gripper fingers and waits for it to open. Using collaborative grippers was justified, which may not have been the case if we had used traditional high powered pneumatic grippers with metal fingers.
- There was consensus that everyone felt safe while operating next to the robot. Thus even the participants who said they were intimidated were now feeling safe after the experiment.
- Everyone also trusted the robot to complete the task successfully. Hence this follows from the pre-experiment notions that robots can add value to the production environment.
- Interestingly, everyone agreed that interacting with the *User Interface* helped them understand the robot's intentions. This validates the hypothesis H5. for natural interaction between humans and robots.
- Finally, everyone agreed that working next to the robot makes their task easier.

Hence we can validate claim H1., that volunteers would feel less intimidated regarding working alongside robots with no safety barriers.

5.1.2.4 HRC Task Specific Questions

Regarding the Task Specific questions, the results are depicted in Figure 5.7:

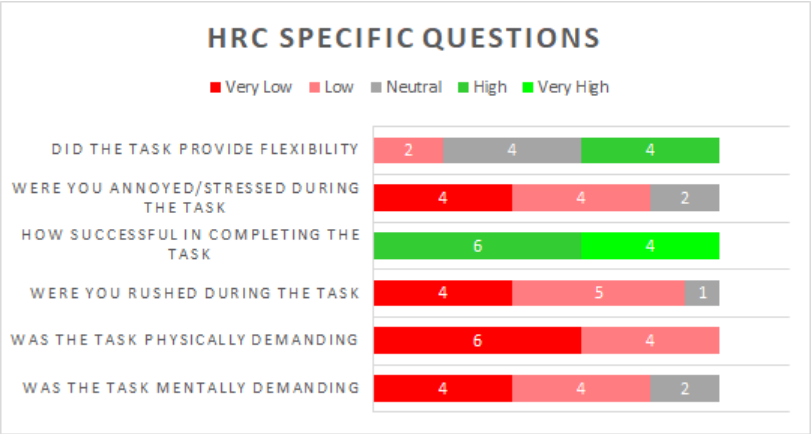


Figure 5.7: HRC Task Specific Questions

- Everyone felt the task was not physically or mentally demanding. Hence validating our hypothesis H3..
- None of the volunteers felt rushed or hurried during the experiment.
- Everyone felt they were highly successful in completing the task.
- On average, there was consensus that they did not feel annoyed or stressed working with the robot.
- Finally, on average four replied positively when asked if the task provided flexibility to them, while four remained neutral and two said it did not provide flexibility. This is a marked improvement from the manual task and hence we can validate our claim to hypothesis H4.

We present some snapshots from the experiment in Figure 5.8:

In this chapter, we discussed the results of the experiments using objective and subjective metrics. Using these metrics, we attempted to evaluate the hypothesis defined in Chapter 4 (H1.-H5.). We proved that human operators would feel less intimidated working alongside a robot without a barrier after performing the

Figure 5.8: Snapshots from the experiment with the volunteers

experiments. Additionally, even the volunteers who had no previous experience with robots felt more confident collaborating with a robot. With the objective measures, we found that HRC task takes more time for completion compared to completely manual (human-only) task. Hence, there is a **trade-off** between production cycle time and ergonomy. Subsequently analyzing the questionnaires, we conclude that operators feel less stressed both physically and mentally while performing the task with a robot co-worker. We also found that the volunteers felt the interaction is more natural if they understood the robot's intentions and this was provided using the *User Interface*. Finally, the outcomes of the experiment does show improvement in terms of the architecture providing flexibility to the operators.

Chapter 6

Conclusions

In this thesis, an architecture for human-robot collaboration is developed and implemented in a real industrial scenario at Schaeffler, addressing a few challenges in shop-floor manufacturing environments. We achieved the overall objectives stated in the beginning of the thesis O1.-O5. The thesis has the following major scientific contributions:

- We explored the possibilities of human-robot cooperation and collaboration in a real industrial scenario. In the process, we attempted to change the mindset and attitude of factory-line operators and higher-level management towards collaborative robots. The operators felt more confident and less afraid of working alongside robots after the experiments.

- The developed architecture ensures ergonomy and flexibility to the human operators. The robot is capable of adapting reactively to the changes in human actions online. In addition, we see a marked improvement in the subjective evaluation of flexibility during the experiments.

- Furthermore, by designing a simple *User Interface* we incorporated a linguistic level of communication between robots and humans. This is a step towards achieving natural and intuitive interaction between humans and robots.

- In most industrial scenarios, the software for robotics is developed using teach-pendant approach. Also, the software is robot dependent and requires expert-knowledge to change and modify the software. We implemented a component-based software architecture which is testable, scalable and robot independent i.e, it can be deployed to many different robots. Additionally, the commands for robot and human actions is provided in the human-semantic level, thus accessible to non-expert users.

- We studied the trade-off between the employed formalisms and the computational performance. This is necessary as in principle, a more flexible and complex system will allow humans to perform different tasks at the expense of overall cycle time. Hence, the AND/OR graphs is developed with few cooperative paths which are designed from common sense and human experience. This allows the operators to choose between the allowed cooperation paths. We found a trade-off between ensuring ergonomy and cycle time. Our architecture ensures that operators have enough flexibility while simultaneously optimizing cycle time.

Apart from these contributions, the architecture also has a few limitations based on our assumptions and hardware restrictions. These limitations stated below are considered for the future work:

- The vision module is used in open-loop, which has a lot of limitations such as imprecise pose recognition, parts moved after image capture causes failure to grasp and so on. Hence a closed-loop vision based control must be considered for the future experiments.

- Grasping has been done using geometric properties of the object and the direct kinematics of the robot. Hence, it is object dependent and not very robust incase on non-symmetric objects. Therefore, a more robust grasping technique must be developed to manipulate various objects in a dynamic environment.

- Safety measures such as robot stop and reduction of impact forces are used in this thesis. However, in a dynamic environment vision-based sensors, wearable sensors or motion capture systems must be used to detect human motion and employ collision avoidance robot motion planning approaches.

- Understanding robot's action intentions increases the trust of human operators in the robots as shown in this thesis. However, instead of employing a user interface to exchange messages between the agents, advanced systems such as verbal communication can be viable options for future research. This increases the naturalness of collaboration between humans and robots.

Appendix A

Questionnaires for HRC

SCHAEFFLER

UNIVERSITA' DI GENOVA

Questionnaire for HRC

Participant ID:

Age:

Sex:

	Yes	No
Do you have previous experience with robots?	☐	☐
Do you consent to video the experiment?	☐	☐

Pre- Experiment	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
In general, would you trust a robot co-worker to operate safely?					
Are you intimidated by the idea of a robot working next to you?					
Do you think a robot will make your overall task easier?					
In general, how open you are to work with robot for daily task?					
Task Specific	Very Low	Low	Neutral	High	Very High
Was the task mentally demanding?					
Was the task physically demanding?					
Were you hurried or rushed during the task?					
How successful were you in completing the task?					
Were you annoyed or stressed during the task?					
Did the task provide flexibility to you?					

X _____

SCHAEFFLER

UNIVERSITA' DI GENOVA

Post-Experiment

Robot-specific	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The motion of Robot made me uncomfortable					
The speed in which gripper picked up and placed part made me uncomfortable					
I felt safe while interacting with robot					
I trusted the robot to complete the task successfully					
The messages in User Interface helped me to understand the robot's intentions					
I trusted the gripper will not drop component					
Interacting with the robot makes the overall task easier					
Task Specific	Very Low	Low	Neutral	High	Very High
Was the task mentally demanding?					
Was the task physically demanding?					
Were you hurried or rushed during the task?					
How successful were you in completing the task?					
Were you annoyed or stressed during the task?					
Did the task provide flexibility to you?					

X _____

References

(2017). Kuka LBR IIWA. https://www.kuka.com/en-gb/products/robotics-systems/industrial-robots/lbr-iiwa. 19

(2017). Rethink Robotics. http://www.rethinkrobotics.com/products-software/. 19

(2017). Universal Robots. http://www.universal-robots.com/GB/Products.aspx. 19

(2018). Cognex Camera Specifications. http://www.cognex.com/support/downloads/ns/1/11/33/is7000G2inst_560.pdf. viii, 57

(2018). OnRobot RG6 Gripper. https://www.universal-robots.com/media/1800257/rg6-gripper-datasheet.pdf. viii, 55

(2018). Universal Robot Specifications. https://www.universal-robots.com/media/1801323/eng_199901_ur10_tech_spec_web_a4.pdf. vii, viii, 49, 50, 51, 52

15066, I. (2010). Robots and robotic devicescollaborative robots. 18

Agostini, A., Torras, C. & Wörgötter, F. (2011). Integrating task planning and interactive learning for robots to work in human environments. In *IJCAI*, 2386–2391. 13

Akella, P., Peshkin, M., Colgate, E., Wannasuphoprasit, W., Nagesh, N., Wells, J., Holland, S., Pearson, T. & Peacock, B. (1999). Cobots for the automobile assembly line. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, 728–733, IEEE. 1

Alexandrova, S., Cakmak, M., Hsiao, K. & Takayama, L. (2014). Robot programming by demonstration with interactive action visualizations. In *Robotics: science and systems*, Citeseer. 44

Andersen, T.T. (2015). Optimizing the universal robots ros driver. Tech. rep., Technical University of Denmark, Department of Electrical Engineering. 64

Argall, B.D. & Billard, A.G. (2010). A survey of tactile human–robot interactions. *Robotics and autonomous systems*, **58**, 1159–1176. 18

Atkeson, C.G. & Schaal, S. (1997). Robot learning from demonstration. In *ICML*, vol. 97, 12–20. 15

Bachelder, I. (2006). Gauging based on global alignment and sub-models. US Patent 6,993,177. 39

Bauer, A., Wollherr, D. & Buss, M. (2008). Human–robot collaboration: a survey. *International Journal of Humanoid Robotics*, **5**, 47–66. 8, 19, 20, 21

Bobick, A.F. (1997). Movement, activity and action: the role of knowledge in the perception of motion. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, **352**, 1257–1265. 21

Caccavale, R. & Finzi, A. (2017). Flexible task execution and attentional regulations in human-robot interaction. *IEEE Transactions on Cognitive and Developmental Systems*, **9**, 68–79. 14

Chen, F., Sekiyama, K., Sasaki, H., Huang, J., Sun, B. & Fukuda, T. (2011). Assembly strategy modeling and selection for human and robot coordinated cell assembly. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 4670–4675, IEEE. 12

Cherubini, A., Passama, R., Crosnier, A., Lasnier, A. & Fraisse, P. (2016). Collaborative manufacturing with physical human–robot interaction. *Robotics and Computer-Integrated Manufacturing*, **40**, 1–13. 1

Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised*, vol. 118. Springer. 68

Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F. & Casalino, G. (2017). Flexible human-robot cooperation models for assisted shop-floor tasks. *arXiv preprint arXiv:1707.02591*. vii, 3, 14, 22, 25, 29, 30

De Mello, L.H. & Sanderson, A.C. (1990). And/or graph representation of assembly plans. *IEEE Transactions on robotics and automation*, **6**, 188–199. 24

De Santis, A., Siciliano, B., De Luca, A. & Bicchi, A. (2008). An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, **43**, 253–270. 16

Denavit, J. (1955). A kinematic notation for low pair mechanisms based on matrices. *ASME J. Appl. Mech.*, **22**, 215–221. 39

Faber, M., Bützler, J. & Schlick, C.M. (2015). Human-robot cooperation in future production systems: analysis of requirements for designing an ergonomic work system. *Procedia Manufacturing*, **3**, 510–517. 8

Gleeson, B., MacLean, K., Haddadi, A., Croft, E. & Alcazar, J. (2013). Gestures for industry: intuitive human-robot communication from human observation. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, 349–356, IEEE Press. 20

Goodrich, M.A., Schultz, A.C. *et al.* (2008). Human–robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, **1**, 203–275. 1

Haigh, K.Z. & Veloso, M.M. (1998). Planning, execution and learning in a robotic agent. In *AIPS*, 120–127. 14

Hawkins, K.P. (2013). Analytic inverse kinematics for the universal robots ur-5/ur-10 arms. Tech. rep., Georgia Institute of Technology. 68

Hawkins, K.P., Bansal, S., Vo, N.N. & Bobick, A.F. (2014). Anticipating human actions for collaboration in the presence of task and sensor uncertainty. In *Robotics and automation (ICRA), 2014 ieee international conference on*, 2215–2222, IEEE. 13, 16

Hoffman, G. (2013). Evaluating fluency in human-robot collaboration. In *International conference on human-robot interaction (HRI), workshop on human robot collaboration*, vol. 381, 1–8. 98

Horaud, R. & Dornaika, F. (1995). Hand-eye calibration. *The international journal of robotics research*, **14**, 195–210. 41

IEC, I. (1998). 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. *International electrotechnical commission*. 17

Iso, I. (2006). 13849-1. safety of machinery, safety-related parts of control systems, part 1: General principles for design. *International Organization for Standardization*. 16

ISO, I. (2010). 12100: Safety of machinery–general principles for design–risk assessment and risk reduction. *CEN, Bruxelles*. 16

Iso, I. (2011). Iso 10218-1: 2011: Robots and robotic devices–safety requirements for industrial robots–part 1: Robots. *Geneva, Switzerland: International Organization for Standardization*. 17, 53

Johannsmeier, L. & Haddadin, S. (2017). A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, **2**, 41–48. 12, 14

Kanade, T., Cohn, J.F. & Tian, Y. (2000). Comprehensive database for facial expression analysis. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, 46–53, IEEE. 20

Khamis, A., Hussein, A. & Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*, 31–51, Springer. 12

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, 396–404, Springer. 19

Kittiampon, K. & Sneckenberger, J.E. (1985). A safety control system for a robotic workstation. In *American Control Conference, 1985*, 1463–1465, IEEE. 18

Koppula, H.S. & Saxena, A. (2016). Anticipating human activities using object affordances for reactive robotic response. *IEEE transactions on pattern analysis and machine intelligence*, **38**, 14–29. 13, 15

Korsah, G.A., Stentz, A. & Dias, M.B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, **32**, 1495–1512. 12

Kosuge, K. & Kazamura, N. (1997). Control of a robot handling an object in cooperation with a human. In *Robot and Human Communication, 1997. RO-MAN'97. Proceedings., 6th IEEE International Workshop on*, 142–147, IEEE. 21

Krüger, J., Lien, T.K. & Verl, A. (2009). Cooperation of human and machines in assembly lines. *CIRP Annals-Manufacturing Technology*, **58**, 628–646. 8

Lacevic, B. & Rocco, P. (2010). Kinetostatic danger field-a novel safety assessment for human-robot interaction. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2169–2174, IEEE. 19

Lim, H.O. & Tanie, K. (1999). Collision-tolerant control of human-friendly robot with viscoelastic trunk. *IEEE/ASME transactions on mechatronics*, **4**, 417–427. 18

Magrini, E., Flacco, F. & De Luca, A. (2015). Control of generalized contact motion and force in physical human-robot interaction. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2298–2304, Citeseer. 19

Martinet, P. (2016). Lecture notes in computer vision. vii, 34

Mason, M.T. (2018). Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, **1**, 1–28. 45

Michalos, G., Makris, S., Papakostas, N., Mourtzis, D. & Chryssolouris, G. (2010). Automotive assembly technologies review: challenges and outlook for a flexible and adaptive approach. *CIRP Journal of Manufacturing Science and Technology*, **2**, 81–91. 9

Michalos, G., Makris, S., Spiliotopoulos, J., Misios, I., Tsarouchi, P. & Chryssolouris, G. (2014). Robo-partner: Seamless human-robot cooperation for intelligent, flexible and safe operations in the assembly factories of the future. *Procedia CIRP*, **23**, 71–76. vii, 9

Munzer, T., Mollard, Y. & Lopes, M. (2017). Impact of robot initiative on human-robot collaboration. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 217–218, ACM. 3

Neto, P., Norberto Pires, J. & Paulo Moreira, A. (2010). High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition. *Industrial Robot: An International Journal*, **37**, 137–147. 51

Nikolaidis, S., Gu, K., Ramakrishnan, R. & Shah, J. (2014). Efficient model learning for human-robot collaborative tasks. *arXiv preprint arXiv:1405.6341*. vii, 15

Nunes, E., Manner, M., Mitiche, H. & Gini, M. (2017). A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, **90**, 55–70. 12

Park, F.C. & Martin, B.J. (1994). Robot sensor calibration: solving ax= xb on the euclidean group. *IEEE Transactions on Robotics and Automation*, **10**, 717–721. 41

Pedrocchi, N., Vicentini, F., Matteo, M. & Tosatti, L.M. (2013). Safe human-robot cooperation in an industrial environment. *International Journal of Advanced Robotic Systems*, **10**, 27. vii, 9, 10

Penate-Sanchez, A., Andrade-Cetto, J. & Moreno-Noguer, F. (2013). Exhaustive linearization for robust camera pose and focal length estimation. *IEEE transactions on pattern analysis and machine intelligence*, **35**, 2387–2400. 40

Pinto, C., Amorim, P., Veiga, G. & Moreira, A. (2017). A review on task planning in human-robot teams. 14

Robla-Gómez, S., Becerra, V.M., Llata, J.R., Gonzalez-Sarabia, E., Torre-Ferrero, C. & Perez-Oria, J. (2017). Working together: a review on safe human-robot collaboration in industrial environments. *IEEE Access*, **5**, 26754–26773. 3, 19, 52

Roitberg, A., Perzylo, A., Somani, N., Giuliani, M., Rickert, M. & Knoll, A. (2014). Human activity recognition in the context of industrial human-robot interaction. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, 1–10, IEEE. 21

Rozo, L., Silvério, J., Calinon, S. & Caldwell, D.G. (2016). Exploiting interaction dynamics for learning collaborative robot behaviors. *Proceedings of the 2016 Interactive (a)(b)(c)(d)*. 13

Sakita, K., Ogawara, K., Murakami, S., Kawamura, K. & Ikeuchi, K. (2004). Flexible cooperation between human and robot by interpreting human intention from gaze information. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 1, 846–851, IEEE. 20

Sarachik, K.B., Bachelder, I.A., Marrion, C.C. & Chang, Y.L. (2001). Object image search using sub-models. US Patent 6,324,299. 39

Schank, R.C. (1972). Conceptual dependency: A theory of natural language understanding. *Cognitive psychology*, **3**, 552–631. 20

Sølund, T. & Aanæs, H. (2017). Towards plug-n-play robot guidance: Advanced 3d estimation and pose estimation in robotic applications. vii, 31

Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, **3**, 323–344. 33

Tsuji, T. & Kaneko, M. (1999). Noncontact impedance control for redundant manipulators. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, **29**, 184–193. 19

Ulmen, J. & Cutkosky, M. (2010). A robust, low-cost and low-noise artificial skin for human-friendly robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 4836–4841, IEEE. 18

Ulrich, M. & Steger, C. (2002). Performance evaluation of 2d object recognition techniques. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, **34**, 368–374. 35

Wilcox, R., Nikolaidis, S. & Shah, J. (2013). Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. *Robotics*, 441. 13

Woods, W.A. (1991). Understanding subsumption and taxonomy: A framework for progress. *Principles of Semantic Networks: Explorations in the representation of knowledge*, 45–94. 20

Xie, S., Haemmerle, E., Cheng, Y. & Gamage, P. (2008). Vision-guided robot control for 3d object recognition and manipulation. In *Robot Manipulators*, InTech. 33