Delft University of Technology

MULTIDISCIPLINARY PROJECT
RO47007

Ahold Delhaize case: Relocating products left at checkout back to their shelves

Team 10:
Rubén Martín Rodríguez (5507391)
Max Polak (4570677)
Pim de Ruijter (4580699)
Frankie Yuezhe Zhang (5390664)
Gioele Buriani (5629888)

June 20, 2022

TU Delft Delft University of Technology

Ahold Delhaize

Contents

0 Team roles

The present section aims to provide a description of the background of the different components of the group, as well as the personal and overall group objectives for the project. Additionally, the role distribution will be presented with additional insights provided by the results of the Belbin test.

0.1 Team introduction

Rubén: completed his BSc in Industrial Electronics and Automation at UC3M (Spain). He is interested in the potential of robots in healthcare, both for rehabilitation and treatment. He will take the role of HRI expert.
Max: has completed his BSc in Mechanical Engineering at Delft University of Technology, the Netherlands. He is interested in robotics solutions in horticulture to contribute to building smart and sustainable greenhouses. He will take on the role of navigation expert.
Pim: completed his BSc in mechanical engineering at the TU Delft. After which he mainly focused on machine learning, deep learning, and computer vision during his master's robotics.
Frankie: completed his BSc in Mechatronics Engineering at Beijing Institute of Technology, China. He is interested in the application of mobile manipulation. He will take the role of high-level planning.
Gioele: completed his BSc in Automation Engineering at the University of Bologna, Italy. He is keen on intelligent control applied to bio-inspired robots (humanoid, quadrupeds). He will be the motion-control expert.

0.2 Expectations

The overall expectation of the group is to obtain a robotic solution that works effectively in the human-inhabited environment. That is, in addition to effectively completing the task, to have a robot that effectively moves, shows itself and interacts with users.
Following the recommendations for the course, we will aim for a simple, yet effective solution that will satisfactorily achieve the proposed objective and later on, we will further develop and improve it. We will do this according to the evolution of the project and the feedback received by the end of the different sprints. To do so, we have planned to work twice per week, right after the lectures. In the meantime, we will aim to develop the solution for every speciality on our own.

0.3 Learning goals

Rubén: all robot-centred projects he's done so far did not include the human factor. In this project, he aims to get a better insight into the factors that improve the interaction of the robot in a human-inhabited environment.
Max: this is the first project to simulate a real work scenario. He would like to gain experience to collaborate with other experts using tools such as Scrum and Git, and also to improve his skills in ROS.
Pim: aims to further deepen his understanding of vision algorithms and the implementation of such algorithms in various programming languages. C++ and Python in this case.
Frankie: in this project, he aims to get hands-on experience with mobile manipulation in the retail market. He wants to further develop a planning framework in addition to the course of RO47014.
Gioele: the main goal he set for this project is learning how to perform in a work-like scenario while coordinating with other experts. Also, he wants to improve his motion control skills in a humanoid robot framework.

0.4 Roles

In addition to the specialized roles, we will define a set of roles in accordance with the results of our Belbin tests, as shown in Appendix B.
Rubén: from his results of the test, Rubén fits with the Expert, Director and Finalizer tags, which matches his personality of attention to detail and making sure the final result is as nice as possible.
Max: according to the test results, Max fits into the roles of Connector, Director and Coach, which reflects his team-leading skills, focus on priorities, negotiation techniques and motivating team spirit.
Pim: according to his test results, Pim best fits the role of judge/appraiser. A role not filled by any other team member and essential to the evaluation of ideas.
Frankie: according to the results of his test, Frankie is a good Finalizer, Diplomat and Connector, which matches his dedication to the things he is focusing on and negotiation skills with different people.
Gioele: according to his test results, Gioele is a good Director, Coach and Finalizer. This reflects his team-leading skills, his enthusiasm and his (sometimes excessive) perfectionism.

Additional roles: Max will also take the role of sustainability expert, Ruben will take charge of documentation proof-checking, Frankie will do the code maintenance, Gioele will take charge of scheduling and Pim of handing in the deliverables.

1 Project goal

Now that we have introduced the team, we proceed to analyze the task at hand, determine the exact solution that we want to implement and how we envision it.

1.1 Problem definition

It is well known that the number of customers in the supermarket fluctuates throughout the day, fewer people will do the groceries at noon, but many people will do them as they get out of their jobs on their way back home. Although such trends and patterns are well-known by the company, it is usually unfeasible to hire additional employees for such short periods. As a result, the employees suffer from problems in handling the sudden peaks of workload when a large volume of customers comes at once.

In addition, these crowds of customers can cause stress and confusion to the employees as products are left at the cashier, on a shelf that does not match the product or accidentally fall on the floor. The employees will then need to spend time correcting these "flaws" by performing undefined tasks, such as cleaning the floor or putting the misplaced products back in place. All these actions fall outside the clearly defined tasks that an employee must perform, preventing them from properly attending to the customers, the check-out, or restocking the shelves. This ultimately results in increased discomfort and workload of the employees; increased dissatisfaction of the customers; and an overall reduction in performance and profits for the company.

Ahold Delhaize (AD) has found in this situation an opportunity for a new "role" which could be taken over by an autonomous robot, which would allow the employees to better concentrate on their designated tasks. According to all the presented information, we propose the following problem definition:

"AD (*problem owner*) needs a way to free their employees from correcting the aforementioned "flaws" (*problem*) during periods of maximum customer flow (*context*) in order to increase employees performance and customer service (*goal*)"

1.2 Vision

In order to determine a fix to the problem presented in the previous section, we have come up with a series of possible solutions: **(1)** make the robot scan the shelves looking for misplaced items via the QR code and put them back in place; **(2)** make the robot take products left in the register back in their shelves and **(3)** make the robot grasp products on the floor and put them back in their shelves.

To better define the idea we will carry on with, we will now take a look at the objectives of the different **stakeholders** in this context. **AD**: its main objective is the increase in productivity and the benefit, which can be attained through efficient allocation of humans or increasing the quality of customer experience. **Employees**: their objective is to perform their job effectively, without the aforementioned distractions and too high a workload. **Customers**: they want to do their groceries in a safe environment and as quickly as possible, without distractions.

Regarding the **financial** aspect of the project, the deployment at a large scale of these robots translates into a significant initial investment. However, improved software can be built into the robots, extending their capabilities and translating into an increased long-term benefit for the company. In other words, even though the robot will only be capable of accomplishing one of these tasks at the beginning, it can be further extended for more complex operations. Therefore, we will implement one of the three aforementioned solutions. From an **ethical** perspective, all the proposed solutions can be implemented in a morally acceptable way. In any case, the robot is not replacing the human workers, but freeing them from distracting tasks which prevent them from effectively doing their job. **Safety** is also an important factor because of the fact that the robot is deployed in an unstructured environment and shall not cause harm to people at all times. Last but not least, there is an additional component related to **users' comfort**. By this, it is meant that the presence of the robot in the environment is such that it translates neither into a burden for the employees nor an annoyance/perturbing factor for the customers.

With all these factors we consider that the robot can be designed to *return products left at the register back to their original shelves*. The robot will do so in the following way:

- It will be considered that an add-on has been implemented in the cashier's current software to establish a channel of communication with the robot. The Graphical User Interface will be designed by us, intuitive to use and quick to handle and retrieve information, aligning with the objective of not disturbing nor increasing the workload of the employees.

- The robot will move through the environment avoiding the customers and in a way that clearly states its intentions so that the robot can be avoided by the customers themselves during operation. To do so, the robot will reconfigure its trajectory upon detected humans and will avoid them. In order to state its intentions, the robot will move with its grippers pointing forward (and the arms folded) and looking a certain distance ahead so as to declare the direction in which it intends to move.
- Once at the pick-up location, the robot will detect the object to be grasped through its corresponding tag and will execute the arm and gripper movement, recomputing the trajectory upon moving or newly detected objects.
- The robot will then take the product to the placing location, which will be determined through the tag attached to the product and an internal map of the supermarket (assumed to be specified upon robot deployment). The robot displacement will follow the same considerations as mentioned before.
- Once in the placing location, the robot will follow the same requirements and considerations as for the picking, performing the placing task in an analogous way.
- Finally, upon task fulfilment, the robot will move back to the charging station or to another cashier for picking up a new product.
- In line with the intention declaration, the robot should be programmed in the most autonomous way possible, with a proper diagnosis of the problem it may encounter. In particular, this will be done by:
  - Detecting the absence of an object at the pick-up location. This will be informed through a message to the employees, which will tell the robot to return to waiting mode or will provide it with the product to place back.
  - Detecting the presence of an obstacle or obstruction in movement. In that case, the robot will attempt to move back and recompute the trajectory. Even in the case of failing at recomputing, if the obstruction is caused by a human, the robot will politely ask them to give way, after waiting a reasonable amount of time for them to move.
  - Detecting the absence of space on the shelves, upon which a picture will be sent to the employees for them to indicate the location of an empty space or to send one of the employees for help.
- In the case of an unidentified problem, the robot will send a help message to the employees and move safely into a safe idle position. Even though this solution strays away from the objective of the employees of strictly focusing on their jobs, there will be some scenarios in which there will be no choice but to ask for help.

1.3 Operational scenarios

The system will present a series of operational scenarios, listing all the possible situations and processes the system will find itself in.

Deployment or Adaptation phase	
S1: Deployment	Robot is located at the store, the charging equipment is deployed as well as the terminals for giving instructions to the system and receiving information from it.
S2: System configuration	The robot is loaded with the map of the store and the shape and location of the different products.
S3: Store map adaptation	Whenever the location of products within the store is rearranged, the updated version of the product locations needs to be uploaded to the robot's database.
Operation phase	
S4: Waiting/charging mode	Charge when the battery is insufficient and waits at the charging station for a request by the cashier to pick up one or several unwanted products.
S5: Product pick-up	Robot moves toward the register and picks up the product.
S6: Product placing	The robot takes the product to the designated shelf.
S7: Back to charging	Upon task completion or low battery level, move back to the charging station.
S8: Customer interaction	When obstructed, the robot will face and ask the customer for collaboration.
S9: Movement obstruction	Whenever an interruption of the movement is detected, the robot will try to free itself or ask for help.
S10: Pick & place error	If it is impossible to perform the picking or placing action, send an informative message.
S11: Self-stop	When unexpectedly running out of battery or anomalous behaviour takes place, send an informative message to the employees.
S12: Manual stop	The robot is shut down by an employee or by a customer through an emergency stop button.

Table 1: Operational scenarios for TIAGo

1.4 Nodes

The system will be developed using *Robot Operating System (ROS)*. This tool is structured in loosely-coupled nodes that execute small-sized tasks and communicate with each other to reach a final objective. Therefore, at least a single node will be developed for each of the five specializations within the project, breaking down the objective proposed in subsection 1.2 into smaller problems that will be tackled separately and that will coordinate with each other towards the final goal.

Text-To-Speech (tiago_tts)
As part of the human-robot interaction aspect, TIAGo will be able to communicate with humans asking for their collaboration in certain scenarios, such as when the humans are standing in TIAGo's way and there is no means to recompute a valid alternative trajectory. This will be achieved by the package **tiago_tts** and its node **tts_client** which convert a message provided through a topic into speech via the already-available tts package from PAL-robotics.

Graphical-User-Interface (tiago_gui)
As part of the human-robot interaction aspect, a graphical user interface will be created for enabling the communication between the robot and the employees. The latter will be able to command instructions to the robot and get feedback on its status in an intuitive and quick manner. The design of the interface will be done via Qt designer by the package **tiago_gui** and node **tiago_gui**.

Error communication (dyn_reconf_mdp)
As part of the human-robot interaction aspect, TIAGo will be able to communicate its error state to operators when that occurs. To do so, a dynamically reconfigurable server package **dyn_reconf_mdp** and node **server** will be created to establish bidirectional communication between the GUI and the rest of the nodes. Information from the nodes can be retrieved and displayed to the employee, and commands can be sent to the robot.

Human detection and product detection (detection)
As part of human detection, TIAGo can scan the environment and recognize a person based on the camera. Machine learning algorithms will be used for person or face detection which is useful for human-robot interaction. As part of the product detection, TIAGo can recognize a product based on the QR marker attached to it. To do this, TIAGo will first rotate his head and scan the area using the cameras to see if there is a desired product exists. If so, TIAGo will look at the QR code and compute the relative position and orientation which will be later sent to the grippers for grasping. The design of the detection functionalities can be done by the package **detection**.

Navigation (tiago_director)
As part of the navigation aspect, TIAGo should navigate in the retail environment and avoid pedestrians and static obstacles. The navigation of the robot base is completed using the package move_base [1]. After receiving the goals from the planner, TIAGo will navigate in the environment to reach the destination. Thus, the package **tiago_director** will be created, which contains a node to receive the commands from the GUI and error messages and execute corresponding actions.

High-level reasoning (planning_control)
As part of the planning aspect, Planning Domain Definition Language (PDDL) [2, 3] and ROSPlan [4] are utilized to generate a sequence of actions based on our problem setting. The PDDL files can generate a sequence of actions given specific goals based on symbolic reasoning and send it to the action interfaces. The action interfaces are some nodes that can gather commands from PDDL and send them to the action servers to execute. In addition, the node **update_goals** can deal with the PDDL request online and modify the PDDL goals at run time.

Control actions (planning_control)
As part of the pick action, after having detected their position and orientation, TIAGo will be able to grasp objects with its free end-effector and keep hold of them until needed. As part of the place action, after knowing the intended location, TIAGo can release a previously grasped object in the correct place.

2 Functional architecture

In the previous section, we defined our solution and how we envision the robot solving the problem. We also made an initial draft of the system's operational scenarios, as well as the nodes that will be implemented. In this section, we will dive into the operational scenarios, as well as the different functions and requirements of the system, finalising with the activity diagram which describes the sequence of events of the solution.

2.1 Extended operational scenarios

In the current section, we will present an extended description of the operational scenarios presented in Table 1. For each of the scenarios in the *Operation phase*, we will present the different preconditions and postconditions of the scenario, as well as the events that trigger them and an extended description of the scenario itself.

S4: Waiting/charging mode	
Preconditions	• System is in idle state/no command has been given • Battery is insufficient
Triggering event	Robot at the charging station.
Description event	The robot will stay in the charging station and recharge its battery.
Postcondition	• Battery charged • Ready for operation

S5: Product pick-up	
Preconditions	• At least one gripper is empty • Pick-up location is known • Ready for operation • Battery sufficiently charged
Triggering event	Picking action is requested
Description event	The robot moves to the requested object location and picks it up with an empty gripper.
Postcondition	• Object in the gripper and being processed • Robot in location of picked product • Ready for operation

S6: Product placing	
Preconditions	• At least one gripper is full • Target placing location is known • Ready for operation • Battery sufficiently charged
Triggering event	Placing action is requested
Description event	The robot moves to the target placing location and places the object in the target place.
Postcondition	• Object in the target location and product processed • Gripper is empty • Ready for operation

S7: Back to charging

Preconditions	• No gripper operation is being performed
Triggering event	Battery level insufficient or task finished or commanded by operator.
Description event	The robot will move back to the charging cradle.
Postcondition	• The robot is at the charging station • The robot enters scenario S4

S8: Customer interaction

Preconditions	• The robot is in scenario S9.
Triggering event	Human is hampering robot's movement and dodging maneuvers have failed.
Description event	Whenever the robot detects that a customer is impeding robot's movement, avoidance maneuvers have failed and a small amount of time has passed without changes, the robot will face the human(s) and ask them to make space for it.
Postcondition	• The robot is capable of moving again to its goal location. • Or, if the obstruction is still present, it will move to S11.

S9: Movement obstruction

Preconditions	• Robot moving to some location as part of S5, S6 or S7
Triggering event	Robot position does not change unexpectedly over some time.
Description event	Whenever the robot detects an interruption of the movement, that is, stays in an unexpected location for too long, the robot will try to free itself by slightly moving in the opposite direction of the obstruction and re-planning the trajectory. Upon failure, the robot asks for help.
Postcondition	• Resuming the operation (back to S5/S6/S7) • Or, if the obstacle is a human and movement is still obstructed, move to S8. • Otherwise, waiting for help.

S10: Pick & place error

Preconditions	• Robot executing S5 or S6 • Robot executing the picking or placing sub-task
Triggering event	No product in the picking location or product target location complete.
Description event	Whenever the shelves are complete or it is impossible to perform the action, send an informative message or take the product back.
Postcondition	• Robot resumes operation (S5 or S6 respectively) or moves back to charging (S4)

S11: Self stop	
Preconditions	None
Triggering event	Anomalous behaviour detected.
Description event	When unexpectedly running out of battery or an another anomalous behaviour takes place, stop and send an informative message to the employees.
Postcondition	• Stopped • Waiting for Operator

S12: Manual stop	
Preconditions	None
Triggering event	Emergency stop button is pressed.
Description event	The system is shut down by an employee or by a customer through an emergency stop button.
Postcondition	• Stopped • Waiting for operator

The following preconditions listed below apply to all nodes. In order to keep the overview, they have not been listed in each table separately.

General preconditions:
- System is initialized

2.2 Needs and functional requirements

In this section, we present a detailed description of the operational needs for properly carrying out the tasks corresponding to the different operational scenarios. From namely operational needs, the requirements of the system will be derived.

Operational needs for S4 Waiting/charging mode	
ON_S4_01:	The system shall be able to inform the operators about the state of the battery and its own availability to operate again.
ON_S4_02:	The system shall be able to increase its battery level while at the charging station.
ON_S4_03:	The system shall stay in the charging area if no request has been sent, so it doesn't hinder the movement of people around it.
System requirements for S4 Waiting/charging mode	
SR_S4_01:	The robot can detect its battery status and has a way to communicate it to operators (either visual or auditory)
SR_S4_02:	The robot charging speed is at least twice the discharging speed.
SR_S4_03:	The charging station is placed in a location where it does not create discomfort to customers or employees.

Operational needs for S5 Product pick-up	
ON_S5_01:	The system shall be capable of moving towards the pick-up location avoiding collisions with static objects in a limited time.
ON_S5_02:	The system shall be able to perform the pick-up operation avoiding all static obstacles in a limited time.
ON_S5_03:	The system shall be able to pick up all intended products with at least one of its free parallel grippers.
ON_S5_04:	The system shall be able to detect the existence of a product at the pick-up location, or the absence of it.
ON_S5_05:	The system shall perform all the operations described in the scenario ensuring the safety of the humans around it.
ON_S5_05.1:	The environment shall be considered shared at all times, with the associated possibility of encountering moving obstacles.
ON_S5_05.2:	The system shall monitor at all times the obstacles in its surroundings, which will vary attending to a whole-body displacement or an actuator displacement, ensuring that a minimum distance is maintained with moving objects and stopping itself otherwise.
ON_S5_05.3:	The system shall perform its actions in such a way that ensures stopping the motion in time when needed.
System requirements for S5 Product pick-up	
SR_S5_01:	The robot is capable of computing a base trajectory from the start to the goal location and can move following it in less than 300 seconds if no obstacle impedes its movement.
SR_S5_02:	The robot is capable of computing an end-effector trajectory from the start to the goal location and can move following it in less than 30 seconds if no obstacle impedes its movement.
SR_S5_03:	The robot gripper is able to pick products of cylinder or box shape, with a maximum dimension of $20cm$, a minimum dimension of $4cm$ and a maximum weight of $0.2kg$.
SR_S5_04:	The robot has a front camera with which it can perceive the presence of objects in front of it.
SR_S5_05:	The robot has a front camera and a lidar for the perception of the surrounding environment and a way to interrupt its motion upon request.

Operational needs for S6 Product placing	
ON_S6_01:	The system shall be capable of moving towards the placing location avoiding collisions with static objects in a limited time.
ON_S6_02:	The system shall be able to perform the placing operation avoiding all static obstacles in a limited time.
ON_S6_03:	The system shall be able to place products in the intended locations.
ON_S6_04:	The system shall be able to detect the tag indicating the placing location and the presence or absence of the placing location above it.
ON_S6_05:	The system shall perform all the operations described in the scenario ensuring the safety of the humans around it.
ON_S6_05.1:	The environment shall be considered shared at all times, with the associated possibility of encountering moving obstacles.
ON_S6_05.2:	The system shall monitor at all times the obstacles in its surroundings, which will vary attending to a whole-body displacement or an actuator displacement, ensuring that a minimum distance is maintained with moving objects and stopping itself otherwise.
ON_S6_05.3:	The system shall perform its actions in such a way that ensures stopping the motion in time when needed.
System requirements for S6 Product placing	
SR_S6_01:	The robot is capable of computing a base trajectory from the start to the goal location and can move following it in less than 300 seconds if no obstacle impedes its movement.
SR_S6_02:	The robot is capable of computing an end-effector trajectory from the start to the goal location and can move following it in less than 30 seconds if no obstacle impedes its movement.
SR_S6_03:	The robot can place products in locations that are at least 10cm wider than the product's width to allow for end gripper movement.
SR_S6_04:	The robot has a front camera with which it can detect the tags of the location and the presence of free space above it.
SR_S6_05:	The robot has a front camera and a lidar for the perception of the surrounding environment and a way to interrupt its motion upon request.

Operational needs for S7 Back to charging	
ON_S7_01:	The system shall be capable of moving towards the charging station avoiding collisions with static objects in a limited time.
ON_S7_02:	The system is able to reach the charging station before the battery is empty.
ON_S7_03:	The system shall perform all the operations described in the scenario ensuring the safety of the humans around it.
ON_S7_03.1:	The environment shall be considered shared at all times, with the associated possibility of encountering moving obstacles.
ON_S7_03.2:	The system shall monitor at all times the obstacles in its surroundings, which will vary attending to a whole-body displacement or an actuator displacement, ensuring that a minimum distance is maintained with moving objects and stopping itself otherwise.
ON_S7_03.3:	The system shall perform its actions in such a way that ensures stopping the motion in time when needed.
System requirements for S7 Back to charging	
SR_S7_01:	The robot is capable of computing a base trajectory from the start location to the charging station and can move following it in less than 180 seconds if no obstacle impedes its movement.
SR_S7_02:	The robot has a way to detect when the battery is too low and it is time to go back to charge.
SR_S7_03:	The robot has a front camera and a lidar for the perception of the surrounding environment and a way to interrupt its motion upon request.

Operational needs for S8 Customer interaction	
ON_S8_01:	The system shall be capable of detecting whether a human is blocking the movement.
ON_S8_02:	The system shall be able to localize the position of the customer and face them.
ON_S8_03:	The system shall be able to clearly ask the customer for collaboration.
System requirements for S8 Customer interaction	
SR_S8_01:	The robot has a front camera that allows it to recognize if the unexpected obstacle is a human.
SR_S8_02:	The robot can determine the position of the human from its detection and can rotate its head to face them.
SR_S8_03:	The robot can communicate with the customer with clear and reassuring voice asking for collaboration.
SR_S8_03:	The voice of the robot needs to be sufficiently loud so as to be clearly heard by the customer (40dB).

Operational needs for S9 Movement obstruction	
ON_S9_01:	The system shall be able to detect when it has stopped moving and the source of the obstruction.
ON_S9_02:	The systems shall be able to gently move back in the opposite direction of the obstruction.
ON_S9_03:	The system shall be able to communicate with an operator in case of an impossible resolution of the obstruction.
System requirements for S9 Movement obstruction	
SR_S9_01:	The robot has a lidar and force sensors on its body that allows it to understand where the possible obstruction comes from.
SR_S9_02:	The robot's actuators allow TIAGo to move in the opposite direction of the perceived obstruction.
SR_S9_03:	The robot can send a signal to a control panel signaling the error related to the obstruction.

Operational needs for S10 Pick & place error	
ON_S10_01:	The system shall be able to detect whether a product cannot be grasped.
ON_S10_02:	The system shall be able to detect whether a product cannot be placed.
ON_S10_03:	The system shall be able to effectively communicate to the operator the origin of the problem.
System requirements for S10 Pick & place error	
SR_S10_01:	The robot has a front camera with which it can perceive whether a product is missing or misplaced and cannot be grasped.
SR_S10_02:	The robot has a front camera with which it can perceive whether a designed location is not completely empty and thus the product cannot be placed.
SR_S10_03:	The robot can send a signal to a control panel signalling the error related to the pick and place actions specifying which inconvenience occurs.

Operational needs for S11 Self stop	
ON_S11_01:	The system shall be able to move into a safe position with its arms close to its body.
ON_S11_02:	The system shall perform the previous operation ensuring the safety of the humans around it.
ON_S11_03:	The system shall be able to communicate to the customers its non-operating state and it is safe to pass by.
ON_S11_04:	The systems shall be able to communicate the event to an operator.
System requirements for S11 Self stop	
SR_S11_01:	The robot can stop any other action it's performing and tuck its arms after checking whether it is possible.
SR_S11_02:	The robot in this state performs the tucking operation extremely slowly in order not to rely too much on possibly faulty sensors.
SR_S11_03:	The robot can communicate its state to customers with either a visual and/or an auditory signal.
SR_S11_04:	The robot can send a signal to a control panel signalling the error related to the self stop specifying which inconvenience occurred.

Operational needs for S12 Manual stop	
ON_S12_01:	The system shall be able to be stopped at any moment by either a customer or an operator.
ON_S12_02:	The system shall be able to communicate to the customers its non-operating state and it is safe to pass by.
ON_S12_03:	The systems shall be able to communicate the event to an operator.
System requirements for S12 Manual stop	
SR_S12_01:	The robot has an easily reachable emergency stop button on its body.
SR_S12_02:	The robot can communicate its state to customers with either a visual and/or an auditory signal.
SR_S12_03:	The robot can send a signal to a control panel signaling the error related to the manual stop.

2.3 Functional hierarchy tree

In this subsection, the functional hierarchy tree is presented. The full functional hierarchy tree can be found in Appendix C. Below, all the main functions are discussed and described. Also, the children functions are stated and described.

2.3.1 Function 1: Motion

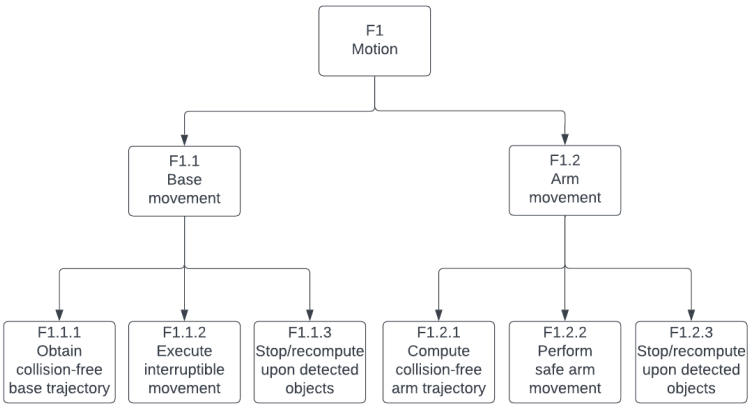


Figure 1: F1 Motion

Function:	**F1 Motion**
Description	The system moves different parts of its body according to the tasks
Inputs	Motion command
Outputs	Motion status (Running, Success, Failure)
Parent function	None
Children functions	F1.1, F1.2

Function:	**F1.1 Base movement**	**F1.2 Arm movement**
Description	The system moves its base to the directed location without collisions or harming people around it	The system moves its arms & grippers to the directed location without collisions or harming people around it.
Inputs	Desired base location	Desired end-effector location
Outputs	Reached the desired base location	Reached the desired end-effector location
Parent function	F1	F1
Children functions	F1.1.1, F1.1.2, F1.1.3	F1.2.1, F1.2.2, F1.2.3

Function:	**F1.1.1 Obtain collision-free trajectory**	**F1.1.2 Execute interruptible movement**	**F1.1.3 Stop/recompute upon detected object**
Description	The system computes a trajectory for the base to reach the given base location whilst avoiding obstacles.	The system executes the planned movement of the base and can be stopped at any time.	The system has detected an object within the reference trajectory, so it stops execution and reroutes a new collision-free trajectory.
Inputs	Target base location and Octomap	Base trajectory	Object detected
Outputs	Base trajectory	System base at target location	New base trajectory
Parent function	F1.1	F1.1	F1.1
Children functions	None	None	None

Function:	**F1.2.1 Compute collision - free arm trajectory**	**F1.2.2 Perform safe arm movement**	**F1.2.3 Stop & recompute upon detected objects**
Description	The system computes a trajectory for the arm to reach the given end-effector location by avoiding obstacles	The system executes the planned movement of the arm and can be stopped at any time	The system has detected an object within the reference trajectory, so it stops execution and reroutes a new collision-free trajectory.
Inputs	Target gripper location and Octomap	Arm & gripper trajectory	Object detected
Outputs	Arm & gripper trajectory	System gripper at target location	New arm & gripper trajectory
Parent function	F1.2	F1.2	F1.2
Children functions	None	None	None

2.3.2 Function 2: Interaction

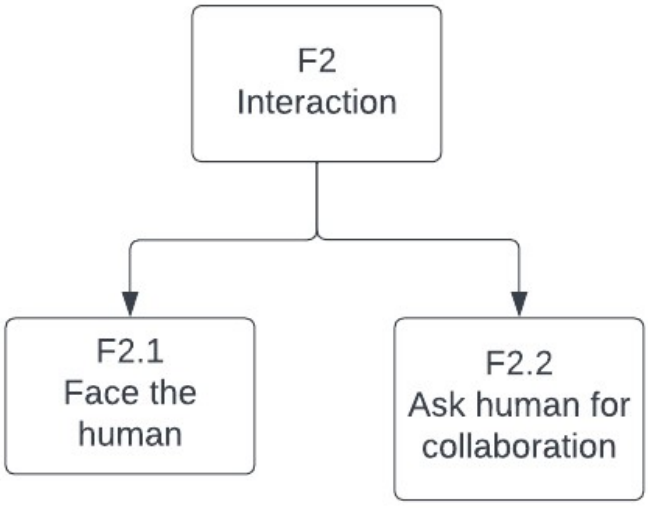


Figure 2: F2: Interaction

Function:	**F2 Interaction**
Description	The system interacts with the user to ask for collaboration upon persistent movement obstruction.
Inputs	Customer location.
Outputs	Interaction status (Running, Success, Failure)
Parent function	None
Children functions	F2.1, F2.2

Function:	**F2.1 Face the human**	**F2.2 Ask human for collaboration**
Description	The robot turns its head towards the human in order to speak with them	The robot communicates to the human asking for collaboration.
Inputs	Human bounding box	Face towards human
Outputs	Face towards human	Message conveyed
Parent function	F2	F2
Children functions	None	None

2.3.3 Function 3: Detection

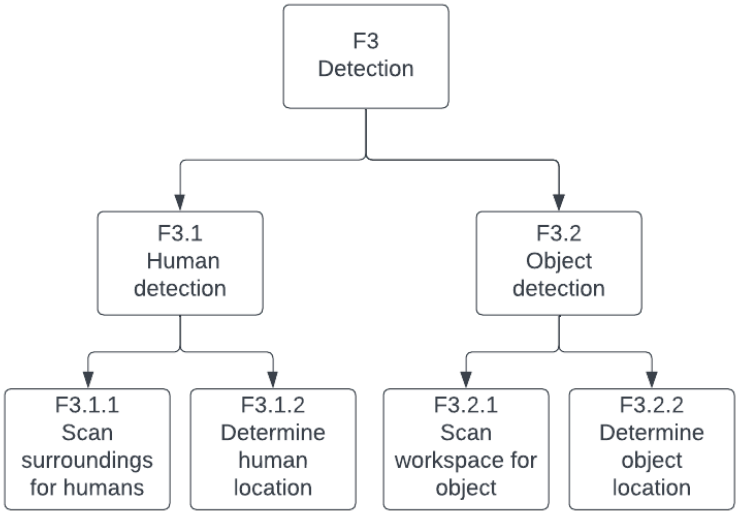


Figure 3: F3 Detection

Function:	**F3 Detection**
Description	The robot uses its camera to detect and recognize elements in the environment
Inputs	Camera images
Outputs	Elements detected
Parent function	None
Children functions	F3.1, F3.2

Function:	**F3.1 Human detection**	**F3.2 Object detection**
Description	The robot detects the presence of humans in front of it thanks to its front camera	The robot detects the presence and the type of products in front of it thanks to its front camera
Inputs	Camera images	Camera images
Outputs	Human bounding box and location	Object type and location
Parent function	F3	F3
Children functions	F3.1.1, F3.1.2	F3.2.1, F3.2.2

Function:	**F3.1.1 Scan surroundings for humans**	**F3.1.2 Determine human location**
Description	The robot constantly scans the front camera image to detect human shapes	The robot determines the location of the human based on its previous detection
Inputs	Camera images	Human bounding box
Outputs	Human bounding box	Human location
Parent function	F3.1	F3.1
Children functions	None	None

Function:	**F3.2.1 Scan workspace for objects**	**F3.2.2 Determine object location**
Description	When at the desired position the robot scans the workspace in front of it to detect objects' shapes and their respective QR markers	The robot determines the objects' type and location based on the previous detection
Inputs	Camera images	Objects detection and QR markers information
Outputs	Objects detection and QR markers information	Objects type and location
Parent function	F3.2	F3.2
Children functions	None	None

2.3.4 Function 4: Error handling

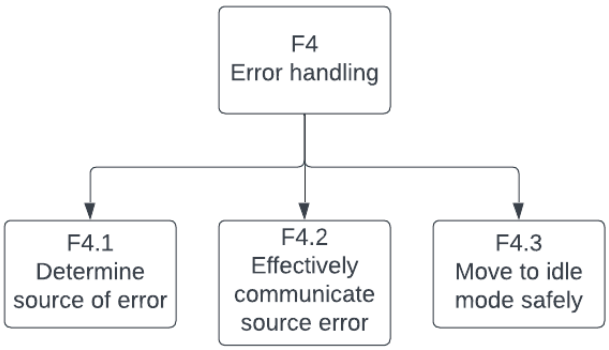


Figure 4: F4 Error handling

Function:	**F4 Error handling**
Description	The system will sufficiently handle errors by being transparent of its actions. It will try out a number of specified error handling procedures on its own. If it can't solve the problem, it will notify the human operator about this error and moves into the safe idle state.
Inputs	Failure in one of the tasks
Outputs	Task succeeded or idle mode
Parent function	None
Children functions	F4.1, F4.2, F4.3

Function:	**F4.1 Determine source of error**	**F4.2 Effectively communicate source of error**	**F4.3 Move to idle mode safely**
Description	The system detects the location of the error to reason and figure out how to fix it on its own.	The system describes the cause of the error in detail to the operator so so that the operator can be used as fallback.	The system cannot resolve the error on its own and turns into safe idle mode and waits until the operator's command.
Inputs	Error	Error	Unsolvable error
Outputs	Location of the error	Send detailed description of the error	Safe idle mode
Parent function	F4	F4	F4
Children functions	None	None	None

2.4 Activity diagram

The activity diagram depicted in Appendix D depicts the sequence of the different actions and functions of the solution. In addition to the different connections between the functions in the system, it also contains the main components of the system, as we stated in subsection 1.2, which include planning, navigation, perception, motion control and human-robot interaction. Additionally, an error-handling component was included in the diagram, as it spans through all the components of the solution. In the incoming lines, a description of the activity diagram flow is provided.

2.4.1 Description

The system initializes upon an order request by one of the employees. The request will be collected through the GUI provided, by pressing a "START" button. The request will contain the information about the cashier from which the product is to be picked.

Once received, the robot will depart from its charging station and move its base to the corresponding cashier (F1.1.1 and F1.1.2). On the way there, the robot may need to re-adapt its trajectory (F1.1.3) upon newly detected humans (F3.1). The robot will do it while stating its intentions, anticipating the direction of the movement by turning its head in such direction beforehand. On the other hand, the robot may find itself in a deadlock, from which it will try to free itself by moving back and recomputing the trajectory (within the scope of F4). Upon failure, if the obstruction is caused by a human, the robot will politely ask for collaboration (F2.1 and F2.2) or request the help of an employee through an informative message if nothing else works (handled in the "GUI & errors" parallel flow). A certain decision will be made by the employee and transmitted to the robot through the GUI. After the intervention of the employee, the robot may resume its corresponding operation (moving to the cashier, shelf or charging station), or move back to the charging station.

Upon arrival at the picking location, the robot will determine the presence or absence of a product, informing the employees through the terminal in the latter case (once again, within the scope of F4.1 and F4.2). In the event of an absent product, the employee may provide the robot with the product or will ask the robot to cancel the current operation. If the product is detected in the first place (F3.2.2), the robot will grasp the object (F1.2.1 and F1.2.2), recomputing the trajectory upon a possible human in the workspace (F3.1 and F1.2.3).

Once the product is picked, the robot will move to the placing location, determined by the product's tag and the internal database. The robot will follow the same protocol for moving through the supermarket as described above. Once at the shelf, the robot will determine whether a space is available for the product. In the absence of it, the robot will identify such absence (F4.1) and send an image to the employees through the terminal (F4.2) who will indicate in the image the location of the gap (if existent) or abort the operation otherwise. If there is a gap, the robot will follow a procedure similar to the one for picking, but in this case, placing the object.

Finally, once the placing operation has terminated, the robot will move back to the charging station or to a cashier if a new order is waiting (unless the battery level is too low and moving to the charging station is necessary, handled by F4.1 and F4.2).

Two parallel flows regulated by a timer can be found in the diagram, in addition to the main diagram/flow described above. The first of them, denoted as "Perception", will take charge or continuously scan the environment for humans and detect their location. Upon such event, the information is transmitted to the system for recomputing the trajectories. The second parallel flow, also regulated by a timer, corresponds to the periodic check of a request for relocating a product, or possible errors in the system. The former, when one order is detected, will initialize the aforementioned main flow. The error handling will consist of displaying an informative message on the GUI and collecting the employee's decision. The diagnosis of the error will have been performed by the overall system.

3 Full functional description of the system software

Finally, after breaking down the functions and requirements of the system, we will present our actual implementation of the system. For a more detailed description of the packages, click on this link

3.1 Function graph

The complete functional graph of the system can be found in Appendix F. The simplified version, in which only our nodes are displayed, is shown in Figure 5

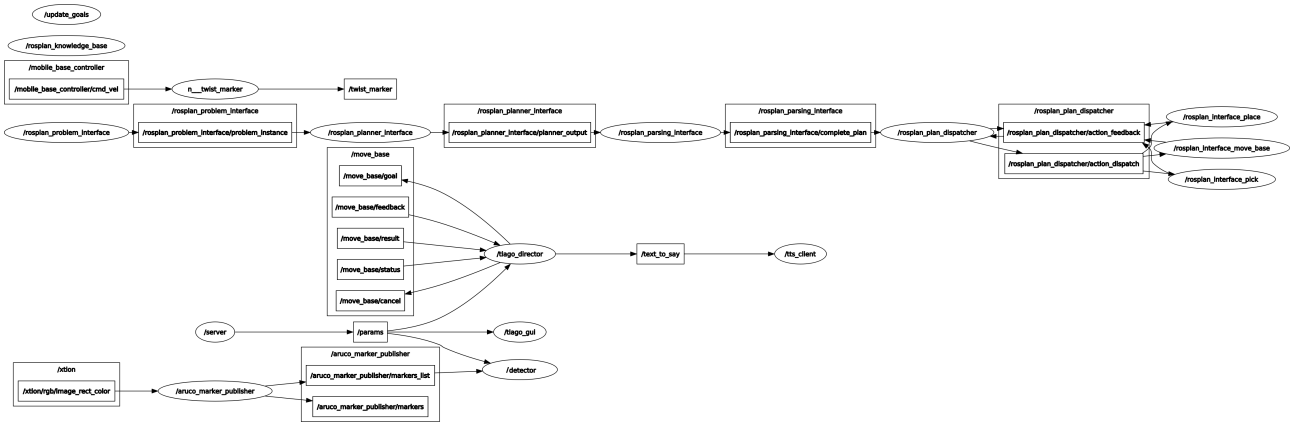


Figure 5: Rqt graph of our solution including exclusively our nodes and how they are related to each other

Our nodes can be observed to be working in synchrony by subscribing to the "/param" topic, in which information about the current status of the system is published. The information is kept up to date employing the **"/server"** node, part of the *dyn_ reconf_ mdp* package developed by the HRI expert. Three nodes are listening to the aforementioned topic. The first of them, **"/aruco_detection"**, developed by the perception expert, takes charge of detecting the aruco markers and extracting the information about the product that they correspond to (communicated through dynamic reconfigure to the "/server" node, not visible on the graph). The second of them, **"tiago_gui"**, developed by the HRI specialist, displayed the information from the topic and displayed it to the user. At the same time, it gathers information from the user through the GUI and updates the system status in the "/server" node (done via dynamic reconfigure, not visible on the graph). The third of them, **"/tiago_director"**, created by the navigation expert, determines the actions that the robot should do, based on the current status of the robot and the information received from the "/params" topic. It will initialize the *rosplan* PDDL file for performing the pick and place actions, displace the robot to the required location, and trigger the interaction upon detected human (done via a service with the "/aruco_detector" node, not visible in the graph) by sending a speech message through the "text_to_say" topic. The latter will be received by the **"/tts_client"**, developed by the HRI specialist, which will interact with a Text-to-speech action server to reproduce the sounds using the *soundplay* package (see full rqt_graph for an insight into the nodes used for the sound reproduction).

On the other hand, we can see at the top a main branch corresponding to the *rosplan* package. Though such nodes were not developed by us, they are in charge of executing the plan defined on the PDDL files created by the planning and motion control experts. The PDDL files will dynamically change the goals during execution via the **"/update_goal"** package, created by the motion control expert.

3.2 Recorded behaviours

Now that we know how the different nodes interact with each other, we will proceed to demonstrate the functioning of the solution in different scenarios for the individual components. The videos of all the recorded behaviours can be found in the project's GitLab repository: see here

3.2.1 tiago_director: navigation

The tiago_director node is the most extensive one among our nodes because of its dual functionality. On the one side, it acts as the node for navigation, by establishing itself as a client to the "move_base" action server, for displacing the robot to the different target locations of the environment. Secondly, by inspecting the values of the parameters contained in the **server** node from the *dyn_ reconf_ mdp* package, and keeping an internal

record of other values within itself, the node is also capable of determining where the robot should go, when to perform the pick-and-place action, or when to interact with humans.

Upon termination of a certain action, the node will inform the **server** node about the changes in the status of the system, so that the GUI is notified and can keep the employee updated about the robot status. In the following paragraphs, we will try to break down different simulations of the robot functioning in different scenarios.

Normal operation

Upon initialization of the simulation, the GUI presents a similar appearance to the one depicted in Figure 16a, informing the employee about the availability of the robot for performing a task. The robot will remain in the "charging station" (which in our simulation corresponded to the centre of the environment) until an order is sent to it Figure 32a. It can be observed in Figure 32b that all the parameter values corresponding to a possible command (order_req, charge and scan) are set to *false*, that the product number is set to -1 (no product detected) and that the level of the system (used for warnings or errors) is set to zero to indicate normal operation. Upon a request sent by the employee, the robot will move to the checkout location (desk in the simulation). This event corresponds to the macro state "MoveTo (Cashier)" in the Appendix D. During execution, the system will detect and avoid any humans in its surroundings, according to the description provided in the detail of the "MoveTo" macro state, also shown Appendix D. During the execution of this displacement, as it can be checked in Figure 32c, the GUI changes to the status depicted in Figure 16b, in which the status of the robot switches to "busy". The GUI will not change during the execution of the rest of the operation unless an unpredictable event occurs. In Figure 32d it can be observed that, upon interacting with the user interface, the values of the parameters have been updated. Now, the "order_req" value has switched to *true*, which is understood by the **tiago_director** node as an indication to move to the checkout to perform a relocation operation.

Once the robot reaches the pickup location, the "product" parameter will change to the ID of the product present at the pick-up location (see Figure 32f). This value will be used by the node **update_goals** to adjust the PDDL behaviour according to the detected product. After that, the PDDL planning will be executed with the updated goal, performing the scanning of the desk to determine the exact location of the product and the pick-and-place operation (see subsubsection 3.2.3 for more details on the rosplan/pddl-based pick-and-place operation). Finally, upon the termination of the pick-and-place, the robot will move back to the charging station, switching the informative message displayed on the GUI to "Going back to charging!". This last displacement corresponds to the state "MoveTo (Charging Station)" of the Appendix D. Upon arrival at the charging station, the parameter values and the GUI appearance return to the state depicted in Figure 32a.

Extra: scan shelves

In addition to order request order action, the robot can also perform the task of scanning the shelves. The robot will scan each shelf one by one and will skip the part where people are standing in front of that particular shelf as can be seen Figure 34. Even though there is ArUco detection of the product on the shelf, the scanning operation doesn't perform any tasks with this information due to time constraints. However, it shows the flexibility and opportunity to expand the robot's operational tasks for future work, which include detecting available spots on the shelf for products to place; detecting the shelf emptiness for updating inventory and restocking notifications; and detecting the shelf for incorrectly placed products. The OctoMap of the shelves after the operation can be seen in Figure 6. As you can see, it shows a very nice map of the shelves at eye level but failed in the lower part. In future updates, TIAGo should be programmed to look down as well.

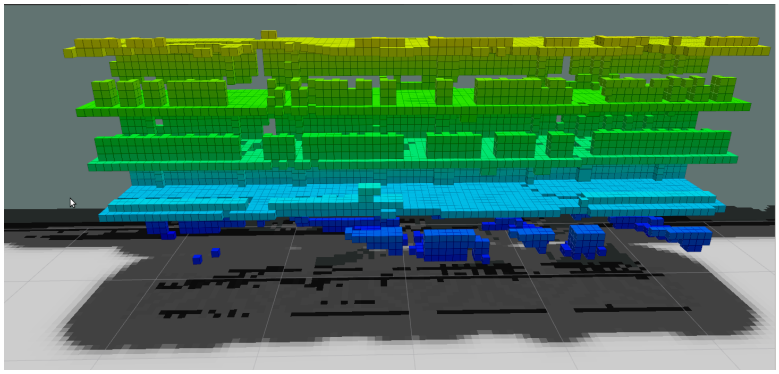


Figure 6: OctoMap of the shelves in RViz.

3.2.2 tiago_director: error handling

Error handling: no product at cashier

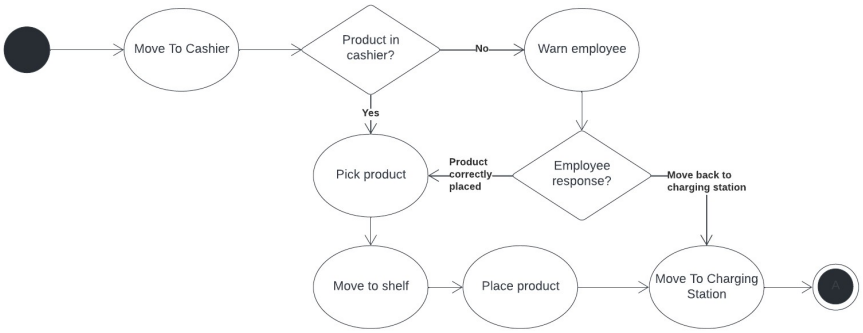


Figure 7: Simplified activity diagram of the picking action error handling.

One of the possible errors that the system must handle is the event in which, upon arrival at the pick-up location, there are no products there to relocate. This scenario, which takes place after the displacement to namely location depicted in Figure 32c, is depicted in Figure 35a. In this case, the **detection** package will not detect any product, leaving the value of the *product* parameter to the default value, as appreciated in the Figure 35b. It can also be observed that the *level* parameter has changed its value to 1, and the *resume* parameter to *false*, indicating that the system finds itself upon an anomaly that can be handled by receiving instructions from the operator of performing some corrective action. The status of TIAGo is changed to orange, and the informative message is displayed (see the section on GUI for more details).

Upon this event, the product can either be placed at the pick-up location and the "Resume" button be pressed to continue operation, or the robot can be sent back to the charging station by pressing the "Go to charge" button. The state of the system and the updated values of the parameters can be found in Figure 35e, Figure 35c and their adjacent details.

The behaviour described above corresponds to part of the functions F4.1 and F4.3 of the activity diagram and, in particular, to the decision branch after F3.2.1.

Error handling: navigation interruption

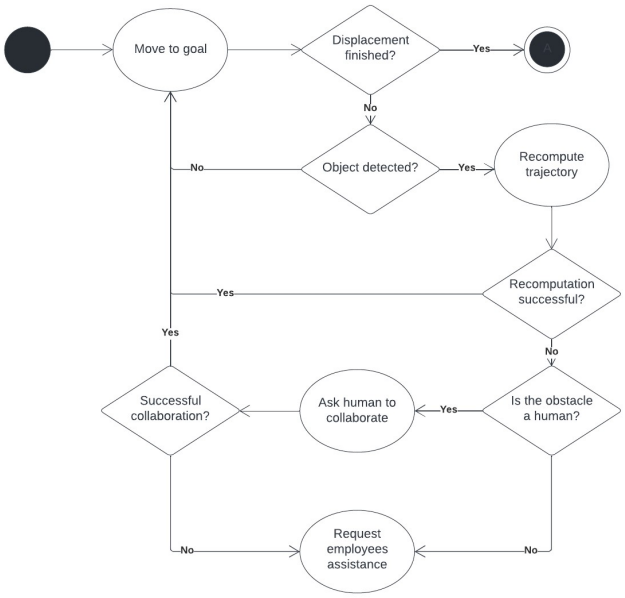


Figure 8: Activity diagram for the error handling during displacements

The second error that is automatically handled by the system is the possibility of encountering an obstacle during a displacement. This behaviour corresponds to the interruption and upper branches of the macro state "MoveTo" depicted in the Appendix D. Upon encountering the object, the system will attempt to recompute the trajectory to avoid the object, falling within the scope of F4, as depicted in the diagram. If the system fails at recomputing the trajectory, it will evaluate whether a human is impeding the movement. If such is the case, the interaction will start, as described in subsubsection 3.2.7.

Once the interaction has been completed, the robot will try once again to reach its goal. If the movement does not succeed, the robot will switch to an error state and will require an employee's intervention. The state of the system in this scenario can be appreciated in Figure 36. It is important to mention that due to the limitation of the current human detection algorithm as described in subsubsection 3.2.4, we assume that if TIAGo can't go to the prescribed location, a human will be standing there such that it starts the interaction, even if it might just be any obstacle (e.g. a restocking car or a wet floor warning board). After this, it will wait a few seconds and try to move again to that specific location. If it again failed, then we can say that it is probably an obstacle (or a human ignoring TIAGo) and we programmed that it then will not start the interaction once more, but rather send a warning Figure 16d to inform the cashier. In this section subsubsection 3.2.4, an alternative solution will be discussed to detect humans instead of these hand-crafted assumptions. With a better human detection model, we could program TIAGo to use different approaches to humans and potential other obstacles or even make a distinction between employees and customers to improve interaction for different purposes.

3.2.3 Planning: PDDL and update_goals

PDDL

The schematic of the planning can be seen in Figure 9. The planning problem is solved using PDDL 2.1 with the POPF 1.1 solver and ROSPlan. The ROSPlan will construct the knowledge base by combining the problem file and domain file. Based on this, the PDDL solver can generate a sequence of actions given specific goals and send it to the action interfaces. The action interfaces are some nodes that can gather commands from PDDL and send them to the action servers to execute different actions.

The POPF 1.1 solver supports modelling logical axioms, predicates and durative actions and is the only readily available solver in ROSPlan. Although the POPF 1.1 solver is not an optimal solver with respect to the consuming time, which means in some cases the robot may take detours to perform a series of actions. We solve this issue by adding predicate `can_move(tiago)` to the actions. We assume that the knowledge base contains the following knowledge:

- the robot can move when it is given the destination and free.
- the robot can pick a product when it is at the waypoint where the product locates. And the product can be picked up and held by the corresponding free gripper.
- the robot can place a product by grippers when the robot is at the desired location - shelf. And then the robot is free.
- the pick and place actions can only be done after the move action.
- it is not possible to move to another location immediately after a move action.

Thus the `move`, `pick` and `place` action can be described using the first order logic:

$$\texttt{can\_move(tiago)} \wedge \texttt{robot\_at(start\_pos)} \Rightarrow \texttt{robot\_at(end\_pos)} \wedge \neg\texttt{can\_move(tiago)}.$$

$$\texttt{can\_pick(tiago)} \wedge \texttt{object\_at(obj, pos)} \Rightarrow \texttt{is\_holding(gripper, obj)} \wedge \texttt{can\_move(tiago)}.$$

$$\texttt{is\_holding(gripper, obj)} \Rightarrow \texttt{object\_at(obj, pos)} \wedge \texttt{is\_classified(obj, pos)} \wedge \texttt{can\_move(tiago)}.$$

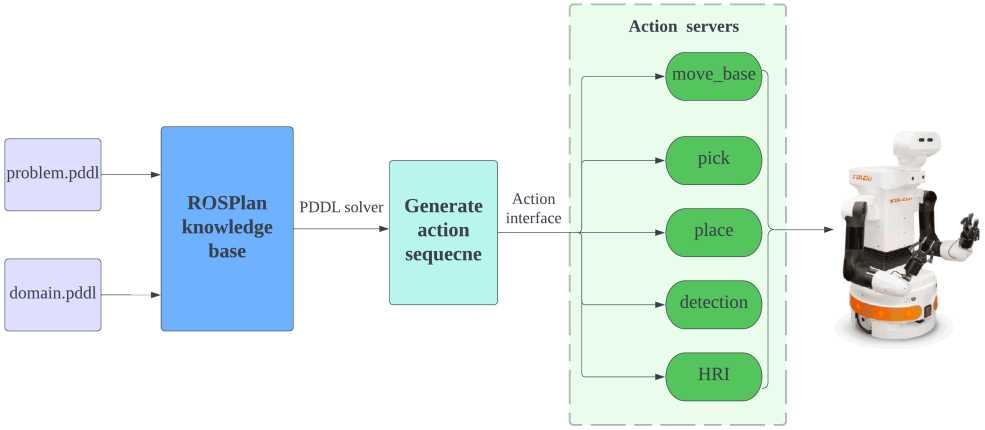


Figure 9: Schematic of planning

The design details of the PDDL domain file can be seen in Table 2. The domain file defines the comprehensive aspects of the problem and applies to different specific problem situations. The file normally contains object types, predicates and actions to define the model. The predicate `can_pick(g, obj)` is used to determine which gripper can pick which item. The predicate `is_classified` describes the state the item is sorted well, which means the item is placed on the shelf where we want.

Table 2: Details of PDDL domain file

Name	Contents
Types	waypoint-wp, object-obj, robot-v, gripper-g
Predicates	visited(wp), gripper_free (g), robot_at(v, wp), object_at(obj, wp), can_pick(g, obj), is_holding(g, obj), is_classified(obj)
Durative actions	move, pick, place

The design details of the PDDL problem file are shown in Table 3. The `Objects` define the items we will use in the world model, including the TIAGo robot, a few location points, two grippers and 4 items of hagelslag. The `Initial states` describe the situations when the program starts. In our case, the robot is already at the table and two grippers are free. All items are at the table. The `Goal` describes what we want to achieve in this task. Here the goal is to make all the items classified. The preconditions of `is_classified(obj)` have been explained above.

Table 3: Details of PDDL problem file

Name	Contents
Objects	tiago - robot wp0 wp_table wp_shelf - waypoint leftgrip rightgrip - gripper AH_hagelslag_aruco_17 AH_hagelslag_aruco_0 - object
Ground truth	(can_pick rightgrip AH_hagelslag_aruco_17) (can_pick rightgrip AH_hagelslag_aruco_0)
Initial states	(robot_at tiago wp0) (free leftgrip) (free rightgrip) (object_at AH_hagelslag_aruco_17 wp_table) (object_at AH_hagelslag_aruco_0 wp_table)
Goal	(is_classified AH_hagelslag_aruco_0 wp_shelf)

The solution of the PDDL planning can be shown in Figure 10. To accomplish the goal, the robot will pick the item using the right gripper. And then it will move to the shelf and place them. The durative actions are all in 2 seconds.

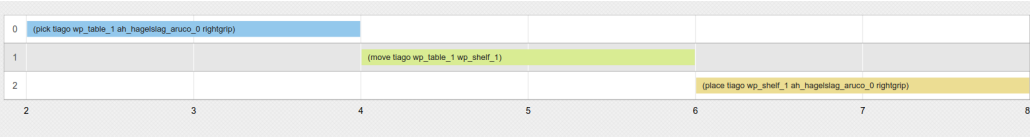


Figure 10: Solution of PDDL planning

update_goals

It is worth mentioning that the PDDL can only deal with the request offline and have prior knowledge about the environment. However, the robot cannot have fixed goals a priori and therefore needs to be able to react to the changes in scenery and adapt to unpredictable events. In other words, TIAGo can only decide its final goal once it knows which product is at the cashier. In order to deal with this problem, we created a node called `update_goals` which allows the `problem.pddl` goal to be updated online, the schematic can be shown in Figure 11.

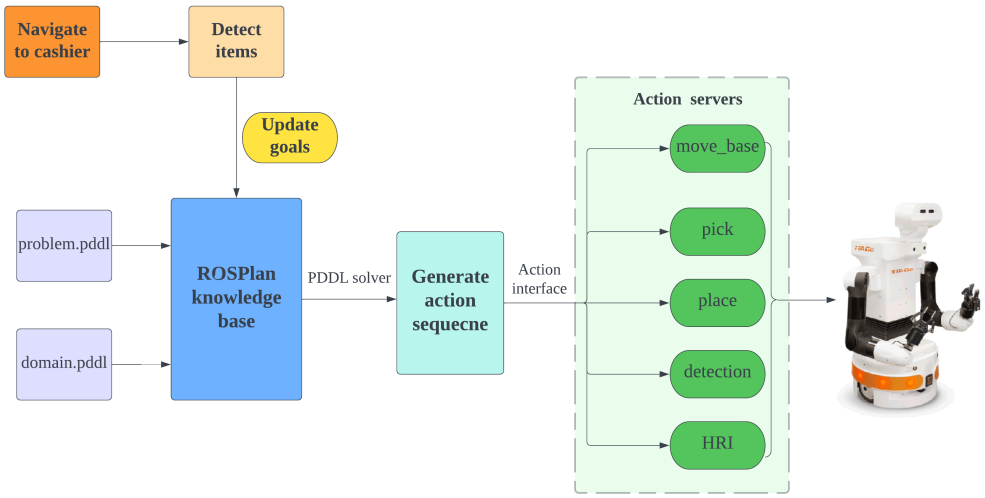


Figure 11: Schematic of planning with update_goals

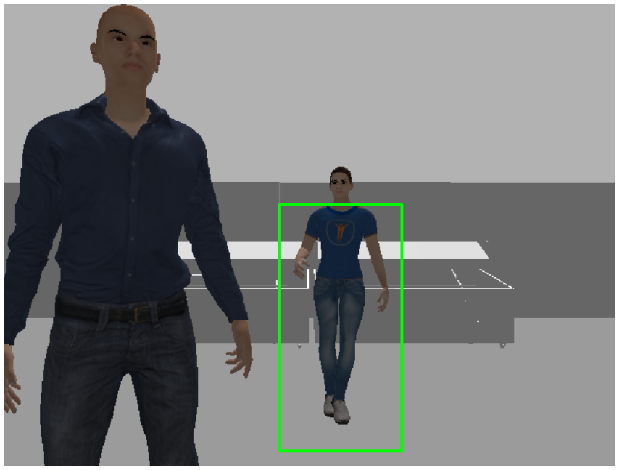
We will now briefly explain the logic behind this node. The `problem.pddl` is first initialized with no goal. While the robot moves towards the cashier, thanks to the detection node it will be able to understand what type of product is at the check-out and, according to that, will update the value of a parameter called `place_product`. The `update_goals` node, when called, will then read the value of this parameter and update the goal of the `problem.pddl` accordingly. To do so, we used the /rosplan_knowledge_base/update server with 1 as the argument to add the goal. When this action is finished, the file automatically generates the problem, calls the planner and then parses and dispatches the plan. One last notice: whenever a new goal is added, its values are stored in some parameters that are then read whenever a new goal needs to be added to remove the previous one. Apart from the first time, every time the goal needs to be updated, the previous goal is removed by reading the previous values from the parameters server and the new one is added according to the value of the parameter `place_product`.

3.2.4 Perception: detection

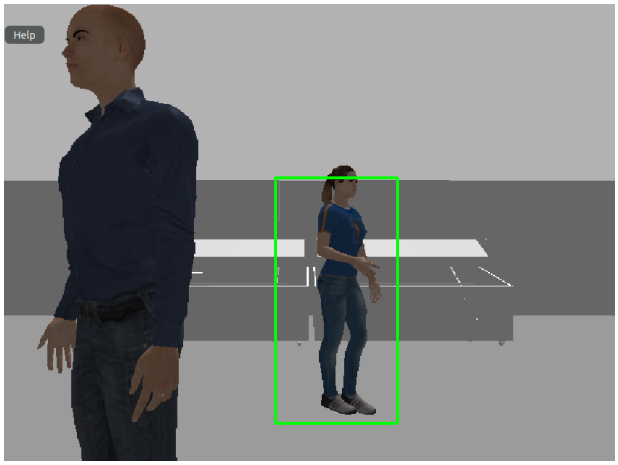
Human detection

The detection node has two main purposes. Firstly, it has the goal to update the parameter server with detection information. Secondly, it has the goal of detecting humans on its trajectory. Obstacle avoidance is included in the tiago_director node. However, for proper human-robot interaction, it is required to know if the obstacle faced is a human or inanimate. The OpenCV HOG person detector provided by PAL robotics did not function as desired since the entire human had to be in the field of view of the camera. The field of view of TIAGo's camera caused parts of the human to go out of view from around 2.5 meters distance. Using the face detector instead solved the problem of proximity, however, would only work when people face TIAGo. Ideally, an upper-body detector would be used to allow for detections from all angles at close range. Training this model, unfortunately, proved unfeasible within the scope of this project. The detection node therefore continuously

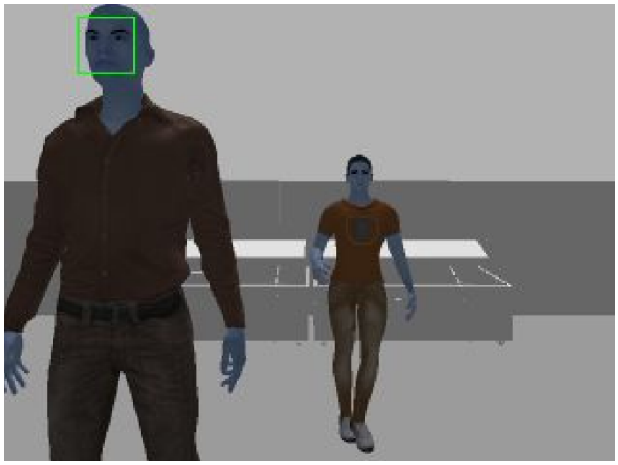
outputs a positive human detection, to allow for the human-robot interaction to function in the test environment.



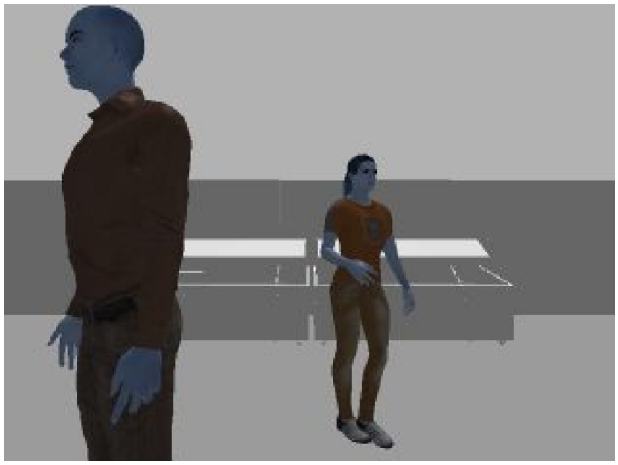
(a) Person detection



(b) Rotated person detection



(c) Face detection



(d) Rotated face detection

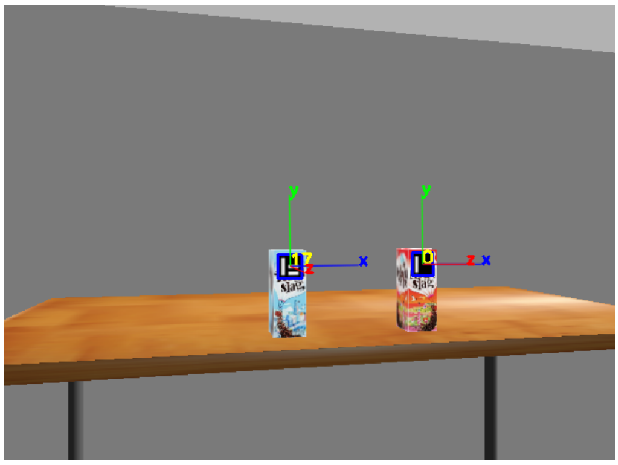
Figure 12: Person and face detection at different distances and angles

ArUco detection

The detection node is also responsible for detecting the QR tags in the simulation or from the actual TIAGo RGBD camera. Two packages were considered for the detection. Namely ArUco tags and AprilTags were evaluated [5]. AprilTags proved to be more easily detectable from a greater distance when compared to the ArUco tags. AprilTags however seemed to create slightly more false detections. In the end, both options worked sufficiently well for the desired application. Considering parts of the pick action server was already made to work with ArUco, it was deemed unnecessary to switch to AprilTags. ArUco detections from the detection node are sent to the parameter server so that other nodes have access. To update the parameter server with the identified tag information the following steps have to be performed. The detection node firstly establishes itself as a client to the parameter server. It continuously monitors the /aruco_marker_publisher/markers_list topic. If products are detected at the checkout the product ids are stored. The node then continues by sending the lowest product-id to the parameter server. This would allow for a simple priority system to be applied. Products that are frozen or chilled could be given a lower product-id to indicate urgency. Therefore the product that is at the checkout with the highest urgency will be picked up first.

The ArUco detection can also help get the position and orientation of the detected item. The related file look_to_point.py was borrowed from the course Knowledge Representation and Symbolic Reasoning and later used in the pick action server. Every time the robot gets close to the cashier, it will rotate its head and see whether there are desired items within its view. If it detects an item in the target frame, it will first use the tf transformation to get the transformation matrix between the target frame and the base_link frame, and then it will transform the point in the target frame to the base_link frame. Thereby, the item's position and

orientation are obtained.



(a) Simulated environment, ArUco

(b) Simulated environment, AprilTag

Figure 13: Tag detection

3.2.5 Motion control

The files that manage the motion control aspects of the robot were not implemented by us but were instead borrowed from the course Knowledge Representation and Symbolic Reasoning where they were already implemented by people with much higher expertise.
We will now give a very brief overview of the functioning of the main actions.

Gripper control
First, we compute a goal position based on what gripper is concerned (right or left) and the required final state (open or close). Then we calculate a simple trajectory to reach the final goal position. Finally, we send the trajectory to the follow_join_trajectory_action server to execute the movement.

Look to point
After deciding a goal point position, use the point_head_action server to make TIAGo rotate its head in the direction of that point. In this case, we use the action instead of publishing to the /head_controller /point_head_action/goal topic.

Move base
For this case, we simply get a goal position and we make use of the move_base_action server to reach that position by navigating the base of the robot in the environment and also avoiding obstacles.

Pick
Together with the place action, this is the most complicated action to be performed. It makes use of both the gripper control and the look-to-point actions previously mentioned. It also makes use of two other servers, namely the collision_object server and the grasp_pose server. The first one is used to add and remove objects as collision objects in Moveit, while the second one is used to calculate the grasp position for the arm.
As for the nominal working of the action, as shown in Figure 14, the robot starts by scanning the area in front of it to find the aruco marker. This part is performed by using the look-to-point action. After finding the aruco marker, TIAGo looks slightly below the object position to capture the table in the octomap. At this point, we add the object as a collision object using the collision_object server and then use the grasp_pose server to calculate the grasp position. Then, a pre-grasp pose is defined based on the calculated grasp pose. TIAGo then moves to the pre-grasp position first and then to the grasp position. This addition is done to facilitate the planning of the action with Moveit. Now the robot closes the gripper with the gripper_control action and then moves to the post-grasp pose, once again calculated from the grasp pose for the same reason as before. Finally, the robot tucks back the arm thanks to the play_motion_action server and places the head back to the central position with the look_to_point server.

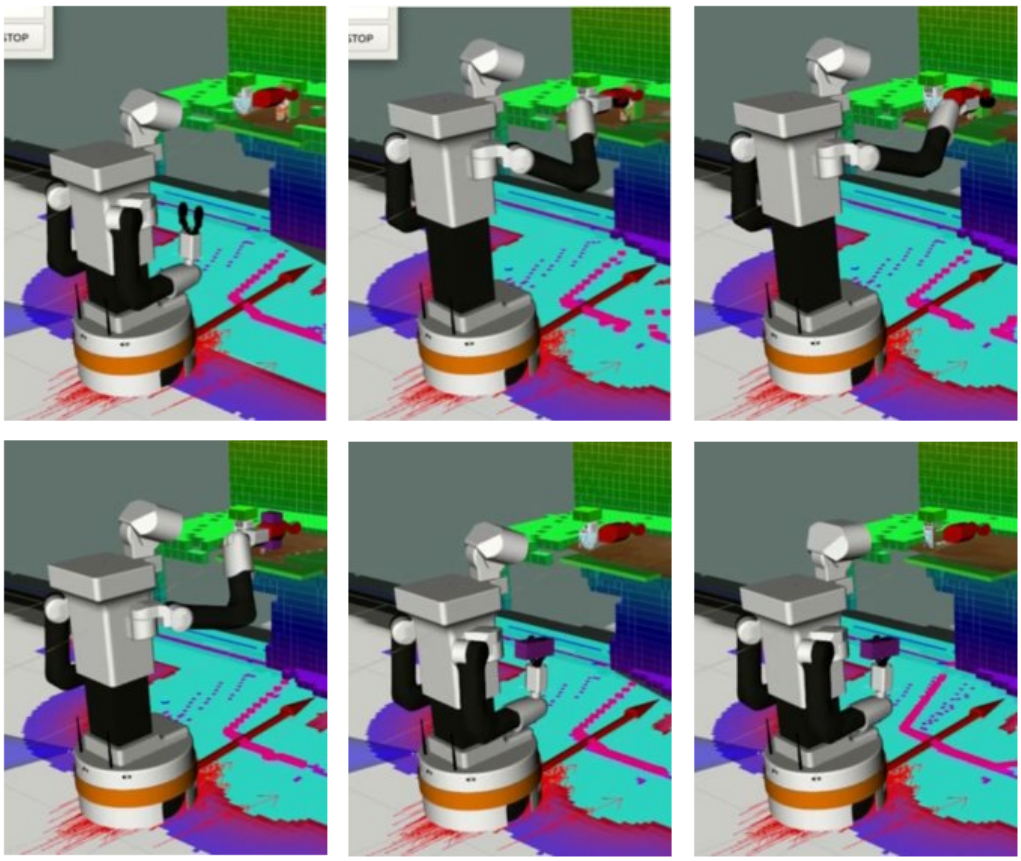


Figure 14: Sequence of screenshots of the Pick action

Place

The place action is rather similar to the pick action from various points of view. In our implementation, the placing position of each shelf is hard-coded, so TIAGo does not have to go through the process of extracting the placing position from aruco markers. Therefore, as shown in Figure 15, after reaching the correct shelf, TIAGo simply has to look around to fill the octomap to avoid collision when using the Moveit library. Then, the robot moves to the pre-place pose and then to the actual place pose (considering the height difference). At this point, TIAGo needs to open the gripper using the gripper_control action to let the product fall in the correct spot. After this, it simply moves back to the post-place position and then once again tucks its arm back with the play_motion_action server. Finally, also in this case, TIAGo moves its head back to the central position thanks to the look_to_point server.

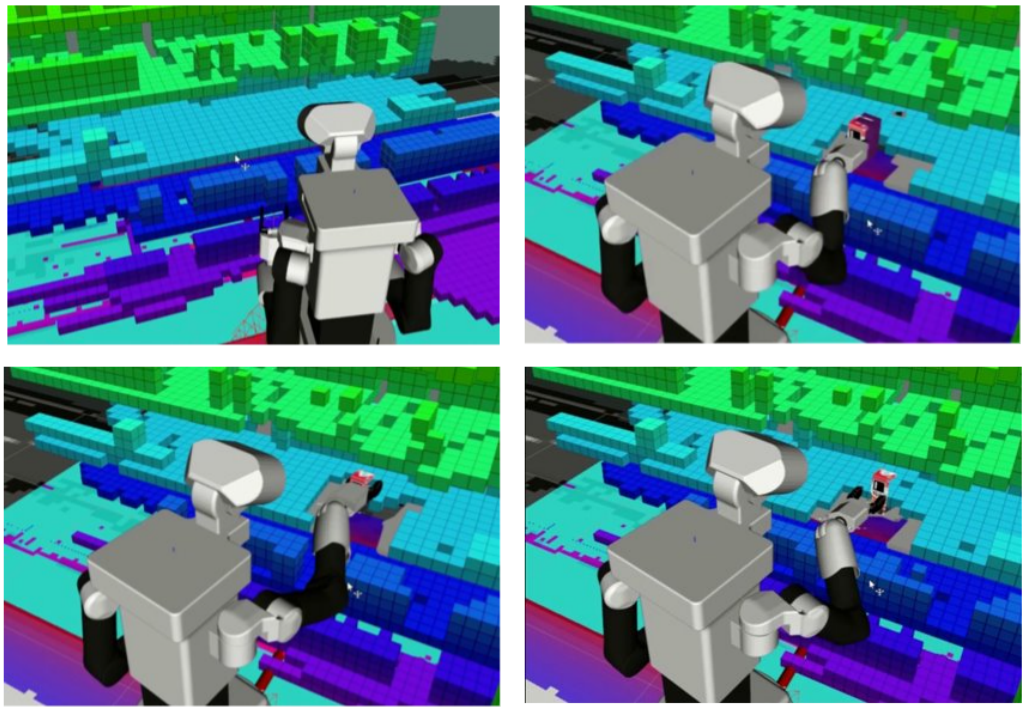
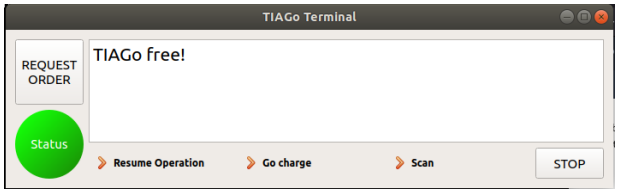


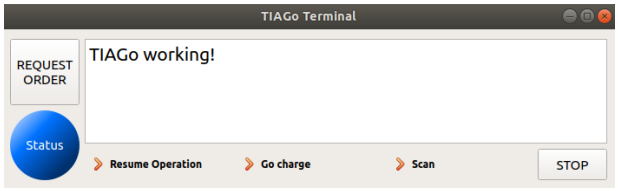
Figure 15: Sequence of screenshots of the Place action

3.2.6 Human-Robot Interaction: interface (tiago_gui)

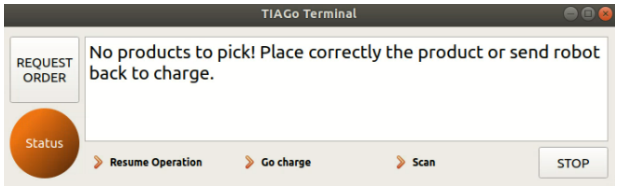
The final GUI can be observed in Figure 16. The GUI was designed in such a way that the commands are as intuitive as possible to the employee, with short messages describing what every button does. Note that the most convenient place to use the GUI is at checkout, controlled by the cashier who can send TIAGo to pick up a product left at the checkout. In this case, the GUI can be installed on top of the general laptop cash register software. The "REQUEST ORDER" button, commands TIAGo to move to the checkout for picking a product; "Resume Operation" to command TIAGo to try again to perform a certain action that was interrupted due to an exception; "Go charge" to command TIAGo to move back to the charging station; "Scan", to command TIAGo to scan the shelves; and "STOP", which can be used as an emergency button to stop the execution of TIAGo's operation. The emergency stop button shuts down all of the nodes related to PDDL and tiago_director while maintaining the action movement servers, the GUI and the server nodes working. It is important to notice that, by doing this we prevent the system from fully shutting down, which could result in dangerous outcomes, it also has the downside of not being able to stop immediately a movement that is being executed. Similarly, the information about the status of the robot was attempted to be conveyed in the least time-consuming way, using an LED whose colour matched the robot's condition and an informative message pannel, describing the status of the system: green for robot availability (Figure 16a), blue for robot performing some task (Figure 16b), orange for a warning state in which employee is required to perform some action (Figure 16c), and red for the error state in which the robot will require the physical intervention of the employee, or an emergency stop has occurred (Figure 16d).



(a) GUI appearance when TIAGo is available



(b) GUI appearance when TIAGo is working



(c) GUI appearance when a warning is issued

(d) GUI appearance when an error occurs

Figure 16: Different states of the GUI

The messages displayed to the user depend on the current functioning of the system. A list of displayed messages and the circumstances under which they are shown can be found in the continuation:

- **TIAGo free!**: displayed when TIAGo is at the charging station and no other task has been assigned to it.
- **TIAGo Working!**: displayed when TIAGo is moving to the picking location or performing the pick and place action.
- **Going back to charging station!**: displayed when TIAGo is displacing back to the charging station.
- **"No product to pick! Place correctly the product or send the robot back to charge.**: displayed when the robot has reached the pick-up location but no product was found to relocate. In this case, the message contains information about the reason for the warning, as well as concise instructions for fixing the fault.
- **An obstacle is impeding the product picking!**: shown when an obstacle is impeding the movement and all attempts to resolve the problem have failed.
- **Emergency STOP! Assistance required! The state of the robot is unknown**: displayed when either the STOP button on the terminal or the emergency stop button on the robot has been pressed.

With all this, the GUI operates in parallel with the rest of the nodes, performing the actions in the secondary branch "GUI & errors" of the Appendix D, mainly performing the function F4.2 and the interruptions for "Order Request" and "Collect human response".

3.2.7 Human-Robot Interaction: vocal interaction (tiago_tts)

Vocal human-robot interaction was only required if a human is hampering the movement of TIAGo. In that circumstance, TIAGo will politely ask the human to step aside so that it can perform the commanded task.

The message with which the interaction will be performed will depend on the current situation of the robot. For example, for an obstruction during the movement to the pick-up location, the robot will tell the user: "Hello, would you be so kind to step aside so that I can pick up this product?". An example of the simulation and the output generated on the terminal upon execution of the speaking can be found in the following figure:

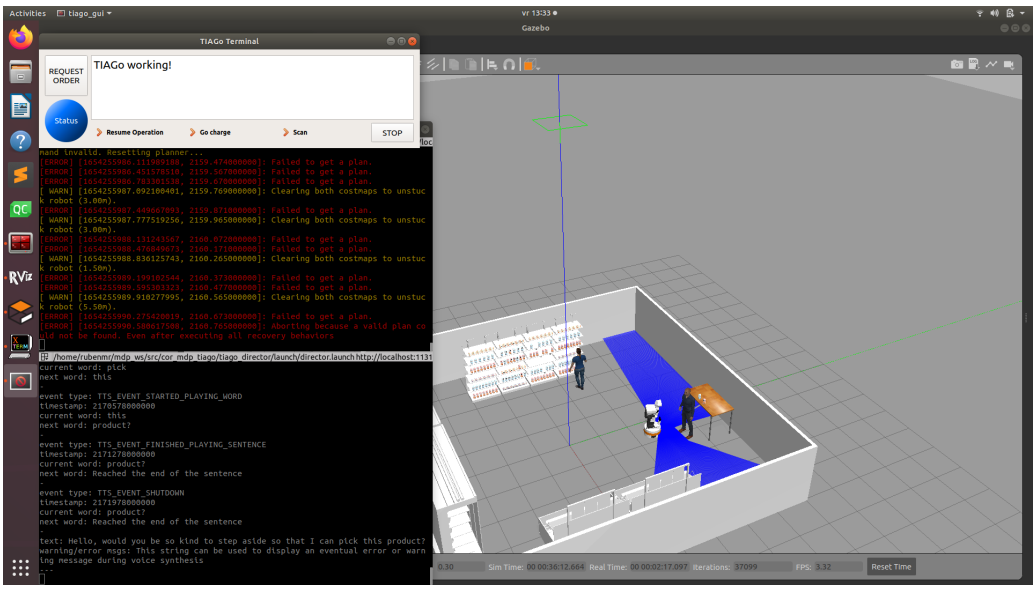


Figure 17: Example output on the terminal upon execution of the vocal interaction with the human on the way

In this way, we have managed to implement the function F2.2 of "Ask for human collaboration", required in the macro state "MoveTo" of the Appendix D.

4 Summary of real robot results

After performing all the described tests in the simulation, we had the opportunity to evaluate whether our solution worked correctly in the real system. In this section, we provide a description of the changes that we had to implement and an analysis of the differences in behaviour that were observed.

4.1 Functional graph

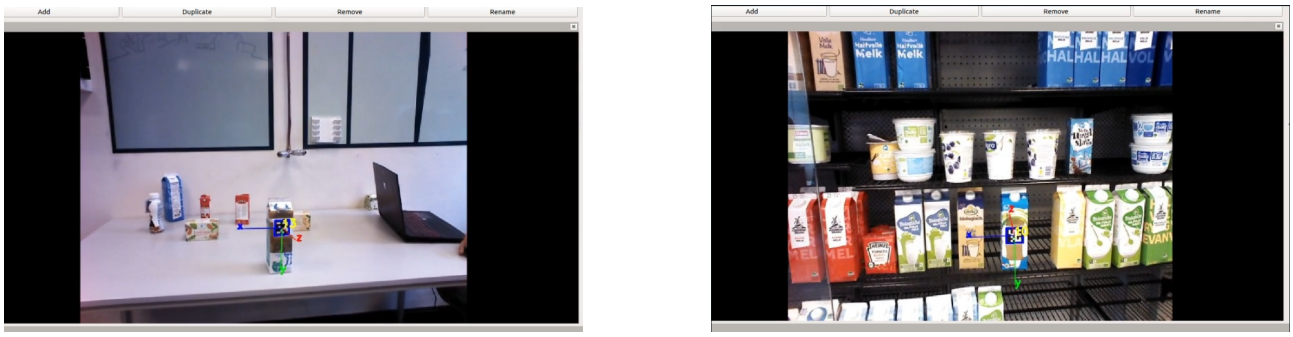
Concerning the nodes created by ourselves, no change was required in the real robot test. The action-based communication established by our solution resulted in no need to adjust almost any of the code. Having said that, we decided to prioritise the recordings of the system and the analysis of the differences in behaviour. For such reason, we have not included in the present report namely rqt_graph of the complete system.
About the components present in the rqt graph, only the nodes related to text-to-speech implementation had to be changed in the real system. The node developed by us, "/tts_client", required the change in a single line of code to bind with the action server for speech in the real robot. As a consequence, the nodes "/tts_to_soundplay", "/sound_play_node", and all the topics and means of communication with and between them were replaced by a single node present in the real robot for reproducing the sounds corresponding to the provided text.
On the other hand, it was necessary to change the waypoints of the system for the correct functioning of the PDDL system. However, no appreciable change was required to the functional graph and so, this issue will be further discussed in the following sections.

4.2 Recorded behaviours

Similarly to that of the previous section, the behaviours of the system can be better seen in the videos available in our GitLab repository (see here).

4.2.1 Perception (detection)

The ArUco detection worked without any big issues in real-world testing. There were however a few small hurdles that had not been taken into consideration as they were not present in the simulation. The delay of the perception was very large when TIAGo's camera was used. It takes an estimated 0.5 seconds for the image to make it to the computer. Making the image compressed improved this slightly, however, it was still very noticeable. Secondly, the frame rate was much lower than expected. The frame rate in the simulation was at times close to 50 Hz. However, in the real world testing the frame rate was much lower. Lastly, the ArUco tag detection never misread the tags in the simulation. Unfortunately, this does seem to happen every so often during real-world operations. A possible cause for this is the vibrations coming from TIAGo when he moves. This causes the camera to shake and therefore artefacts can occur in the image. The detection range of the real TIAGo seems to be better than the simulation, allowing TIAGo to detect tags from greater distances.



(a) ArUco detection at the picking location

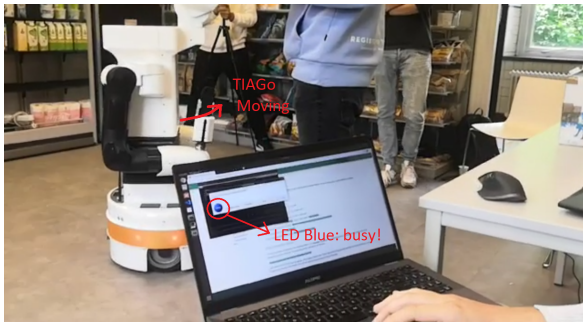
(b) ArUco detection at the placing location

Figure 18: ArUco detection in the real environment

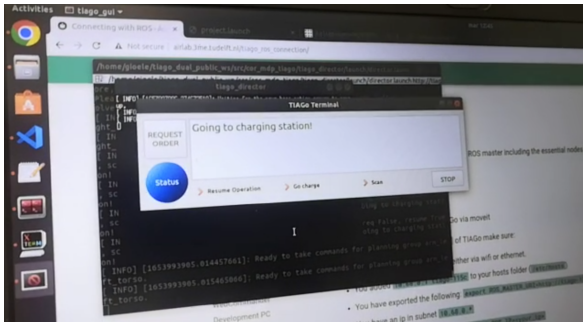
4.2.2 GUI (tiago_gui)

The GUI user interface operates in a similar way to that of the simulation. It enabled to send commands to the robot, which were executed appropriately by the robot. Conversely, the status of the robot was correctly stated at all times. An example of the correct functioning of the robot can be appreciated in Figure 19, in which the robot is initially moving to the charging station, with the GUI looking as depicted in Figure 19b. Once arrived at the charging station, the LED changes its appearance. Upon clicking the request order button, it can be

seen in Figure 19d that the robot starts to move to the checkout and that the appearance of the GUI changes once again.

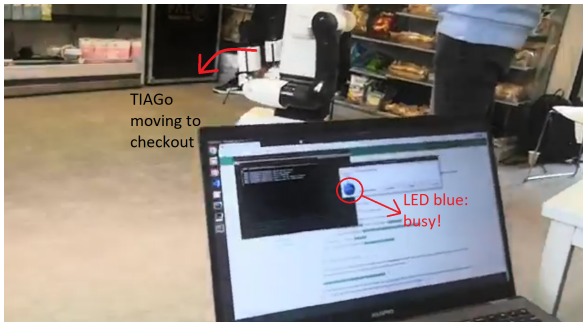


(a) Robot and GUI when moving to charging station



(b) Detail of the GUI when moving to charging station

(c) Robot and GUI once arrived to the charging station



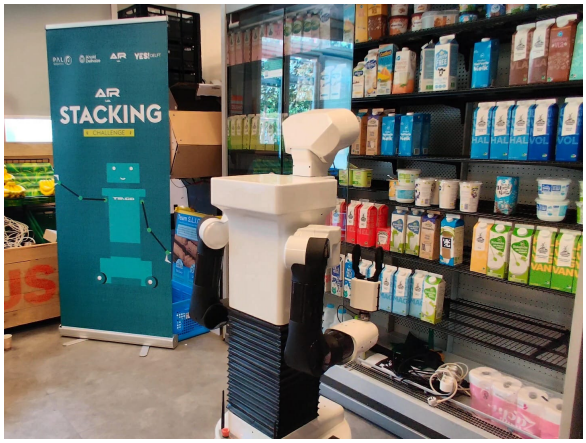
(d) Robot and GUI after sending an "ORDER REQUEST" command

Figure 19: Details of the GUI in different stages.

4.2.3 Vocal interaction (tiago_tts)

We reproduced the situation depicted in subsubsection 3.2.2, in which a human is standing in the way of the robot for a prolonged duration and vocal interaction is carried out to request collaboration from the user. A video of namely situation can be seen in the repository and some captures of it can be seen in Figure 20a. It is important to notice, though, that we only accomplish to perform the interaction once, as the event by which it was triggered was hardly ever reached, as we will discuss in the following section.

(a) Robot interacting with a human using text-to-speech (manual mode)



(b) Robot resumed the operation when the human got out of the way (manual mode)

Figure 20: Using text-to-speech for collaboration with customers that are obstructing the movement of TIAGo

4.2.4 Picking operation (tiago_director + PDDL)

The pick and place action constituted one of the major problems encountered in the session with the physical robot, and we only managed to successfully perform a couple of trials in which the robot did manage to perform

some steps of the picking action. A sequence of the different steps undergone by the system for the picking action can be observed in Figure 21. The sequence is similar to the result of PDDL planning in the simulation, as we have discussed in subsubsection 3.2.3. The robot will first move to the checkout guided by the tiago_director after receiving the pick-up request. And then the detection node will play a role in extracting and sending the new-added product information to the parameter server so that the update_goals node could integrate the information with the goal we want to pick. After that, the robot will execute the pre-grasp and pick motions.

In all cases in the real test, the robot managed to successfully perform the displacement to the "checkout" and perform the movement to the pre-grasp position (see Figure 21c). However, unlike the high success rate of pick action in simulation, most of the time in real test, the movement from the pre-grasp to the picking position (and posterior arm movements) was not accomplished with success, as we will discuss in the next section.
In addition, TIAGo was not able to execute the real grasping action even though the pre-grasp was already aligned. This had probably to do with the fact that the product was seen as an obstacle which it has to avoid, but more on that in subsubsection 4.3.1. In Figure 21d, we have manually moved TIAGo forward and closed the gripper to show that the pre-grasp should have been sufficient to do the actual picking action.

(a) Robot waiting at charging station for request

(b) Robot moving to the checkout

(c) Robot executing the pre-grasp motion

(d) Picking of the product (manual mode)

Figure 21: Different stages of the PDDL's pick and place.

4.3 Debug report

In the test with the real robot, we noticed that two functionalities of the system didn't work as planned. The first of them involved the picking action, and the second one, related to the attempts to avoid a human when it is standing at the picking location, which ultimately leads to the trigger of the vocal interaction.

4.3.1 Picking error

As mentioned in subsubsection 4.2.4, the biggest problem we encountered was related to the picking action. We are not yet completely clear on what might have caused the problem, but we had some hypotheses.

First of all, we considered the possibility of it being a problem related to the RRT algorithm included in the Moveit library. RRT is not exactly an optimal planning algorithm, especially if run for a short amount of time (which is however necessary to keep the robot responsive and as fast as possible). However, the same algorithm is run on the simulation and, even though sometimes there were errors there as well, the algorithm would work most of the time. In the case of the real scenario, instead, the algorithm never worked despite our multiple attempts.
At this point, we thought it could be a problem related to the offset of the coordinates for the product detection. To be more clear, the robot first detects the product and its coordinates, then applies an offset along the three axes to determine where exactly to place the gripper to correctly pick the product. We thought the problem might be related to this offset and in particular with the possibility of the real robot frame being rotated with respect to the one in the simulation. We, therefore, tried modifying the offsets along all the three axes but, although we tried several different settings, it once again did not work.
We had, however, one trial where, probably due to some errors in the self-localization that will be further explained in subsubsection 4.3.2, the robot got the relative coordinates of the product completely wrong. In this case, therefore, the robot thought the perceived object was in a location that was, effectively, empty and only in this case it did perform the whole picking action, obviously missing the product. Even though this action was yet another failure, it showed a different behaviour with respect to all the previous ones and therefore made us think. The first thing that came to our mind is that the robot was probably seeing the product itself also as an obstacle. This might be the reason why it only worked when the product itself was not where the robot wanted to move its gripper. This did not completely make sense since in the pick action server some lines of code should take care of this problem by adding the object as a collision object using the collision_object server. We, therefore, tried modifying these lines to try and solve this issue but even this attempt was unsuccessful.
In the end, we did not have time to try other solutions, so unfortunately we did not manage to perform the full pick action correctly. A possible workaround that we, unfortunately, thought of after our time was up was to find a way to directly remove the voxels belonging to the product from the octomap. In this way, the robot could not detect the product as an obstacle, if that was indeed the problem.

4.3.2 Drifting when consistently trying to avoid human

Although the hand-crafted method to perform the navigation interruption as discussed subsubsection 3.2.2 worked in the simulation, it didn't work in the real world. The robot tried heavily to somehow go to the prescribed location but didn't collide with one of our team members, surprisingly, however, TIAGo didn't start the vocal interaction (see subsubsection 4.2.3). Because it finished moving to the sent location, but the actual location was not the same, the action hasn't been completed. This has probably to do with the similar issue of the localization ambiguity due to drifting (i.e. the real location is misaligned with the sent location). One of the solutions was to re-localize the robot and re-map the whole environment. This is because the laser-range finder has only a 180-degree field of view [6] and the mapping was done just once as TIAGo's navigation system can adapt to changes that occur over time. However, a new map of the same environment should be redone when there are major changes to the layout[7]. In our case, the moving humans could influence the layout and after we retried the testing several times, the localization errors could accumulate. Therefore, it is necessary to re-map the environment after we started testing the robot several times.

References

[1] *Move_ base - ROS Wiki*. URL: http://wiki.ros.org/move_base (visited on 04/05/2022).

[2] Drew McDermott et al. "PDDL-the planning domain definition language". In: 1998.

[3] Maria Fox and Derek Long. "PDDL2.1: An extension to PDDL for expressing temporal planning domains". In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124.

[4] Michael Cashmore et al. "Rosplan: Planning in the robot operating system". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 25. 2015, pp. 333–341.

[5] Aufar Zakiev et al. "Virtual experiments on aruco and apriltag systems comparison for fiducial marker rotation resistance under noisy sensory data". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–6.

[6] *TIAGo for perception - PAL robotics blog*. URL: https://blog.pal-robotics.com/versatility-of-tiago-as-a-research-platform-for-perception/.

[7] *TIAGo for navigation - PAL robotics blog*. URL: https://blog.pal-robotics.com/versatility-tiago-research-platform-navigation/.

Appendix

A Change log

Date	Feedback/Update statement	Description
03/05/2022	Feedback Sprint 0	Cover page; '...'? Nice time introduction. Your problem definition is well formulated in general but needs to be more specific. What are the unstated tasks? In your vision part, you are exploring what problem you are solving, not how! Try to make it more specific. How exactly are you solving the problem you focus on? (so, what specific options are there for the robot to help?) How will the robot indicate its intentions? How does the robot interact with the employees? How will the robot move through the supermarket? What will the robot do when the task is finished? Operational scenario names look good, but please indicate what happens in the respective scenario. The node section is not adding anything. Try to be specific!
10/05/2022	Update statement	As indicated in the feedback of the Sprint 0, we forgot to set the title of the (we left it as three dots), we have now inserted a title suitable to our project goal.
10/05/2022	Update statement	Performed changes in subsection 1.2 to include a more detailed view of how the solution is envisioned. We inserted a list with key points of the final solution and the means to implement them.
10/05/2022	Update statement	Following the feedback provided in Sprint 0, we have included the complete list with descriptions of the operational scenarios in subsection 1.3.
10/05/2022	Update statement	A complete list of the intended nodes of the final solution has been included in the last section of chapter 1.
10/05/2022	Update statement	As required for the deliverable of Sprint 1, we have carried out a full development of section 2, including the hierarchy tree and activity diagram as shown in Appendix C and Appendix D.
20/05/2022	Feedback Sprint 1	2.1: In general, very good operational scenarios. In S11: what "stopped" means, be precise. Very nice separation of the manual stop from the self-stop 2.2: very complete, detailed and precise operational needs and systems requirements, well done! 2.3: Excellent functional hierarchy and descriptions, well done! 2.4: The activity diagram is very detailed, you seem to have thought about it carefully. However, the diagram itself presents multiple notation problems that result in a confusing flow. See notes. There cannot be two connections coming out of a node (it happens with the System start node), except if it is a decision or a fork node In general, I do not see a send signal node for the receive signal nodes that you have (which seem correct, with a corresponding interruptible region. Also, in general, you probably need parallel flows for perception activities that can interrupt the nominal flow. They can have their separate start node with a timer to indicate that they are executed regularly, for example. Very good that you created a subdiagram for one of the functions, but you need to indicate the entry/start point and the exit/termination point. Also, you need to start thinking of allocating the functions in the flows to your building blocks, using swimlanes. Change log: The change log is missing a proper format of a table existing of columns (Date, Feedback / Update statement, Description) with rows of each change/feedback separately. The description could use some more detail in which the actual change is described shortly.
22/05/2022	Update statement	Added a more precise description of how we intended the robot to stop.

24/05/2022	Improved change log	Following the indications provided in the feedback from Sprint 1, we have changed the format of the change log into a table with the required fields for better tracking of the document changes.
25/05/2022	Update statement	Following the feedback provided in Sprint 1, we have introduced the swimlanes in our design; we have added the start and end points of the "macro" diagram for moving to a certain place; we have fixed the problem of multiple arrows starting from the same node without decisions nor forks, and parallel executions with timers have been integrated for activities running on a parallel that interrupt the nominal flow or handle exceptions.
08/06/2022	Update statement	Implemented a complete Chapter 3 with the complete rqt_graph and the recorded behaviours for simulation. Chapter 4 was already started, with the recorded behaviours from the demo with the real robot.
12/06/2022	Update statement	Final implementation of Chapter 4, extended debug report and comparison between the behaviour of the real system with that of the simulation. An additional explanation for the reasons why certain parts of the solution didn't work as expected was also included. Videos of the recorded behaviours in the demo were also uploaded to GitLab and linked in the section.
14/06/2022	Update statement	We didn't explicitly mention earlier in the report how and where to implement the self-made GUI since we considered this convenient, however, during the final presentations other team members did mention this as an important point. It can be seen at subsubsection 3.2.6.
16/06/2022	Feedback Sprint 3	3.1: It's nice that you attached the complete rqt_graph and the relevant one, which makes it much easier to understand the whole system. 3.2: 3.2 Excellent job on including videos of recorded behaviours. Pick and place actions are very smooth. 3.2.4 In Figure 13, there are aruco detections in simulation and the real world. Any difference in performance? 3.2.6 When an obstacle is impeding the product picking, does this obstacle have to be a human being? As in the perception part, you only have human detection in the system. Does the emergency button stop the current and following actions completely once it's pressed? As in the "emergency_stop_speed_up.mp4" on GitLab, it doesn't seem that TIAGo stops moving because the emergency button has been pressed, but because a tag has been detected. And even after the button has been pressed, TIAGo's head still moves down trying to look at the product.
19/06/2022	Update statement	Removed the image of the real world Aurco detection from chapter 3 to chapter 4 to avoid confusion and perform the comparison between simulation and the real world in the latter, as stated in the course guidelines.
20/06/2022	Update statement	In sections 3.2.6 and 3.2.7 we made more explicit the conditions under which the robot triggers an interaction to ask for collaboration, try to find an alternative solution to the problem, or triggers an error message. Additionally, a more extensive description of how the emergency stop is implemented, and how it could be further improved was provided. A final review of the entire document was also carried out before the final deliverable.

B Belbin test

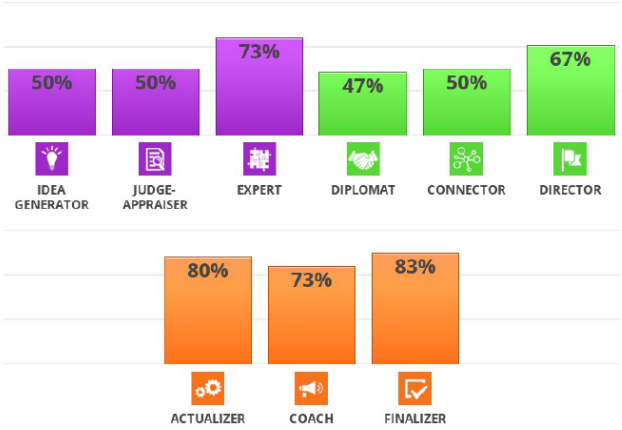


Figure 22: Ruben's Belbin test results

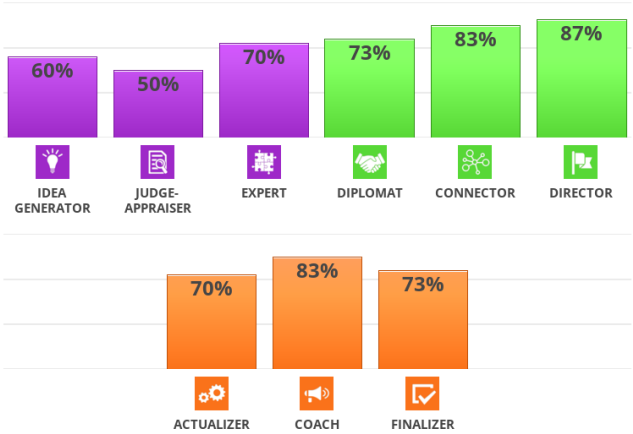


Figure 23: Max's Belbin test results

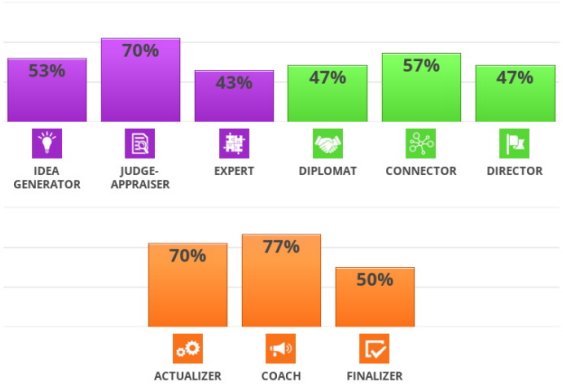


Figure 24: Pim's Belbin test results

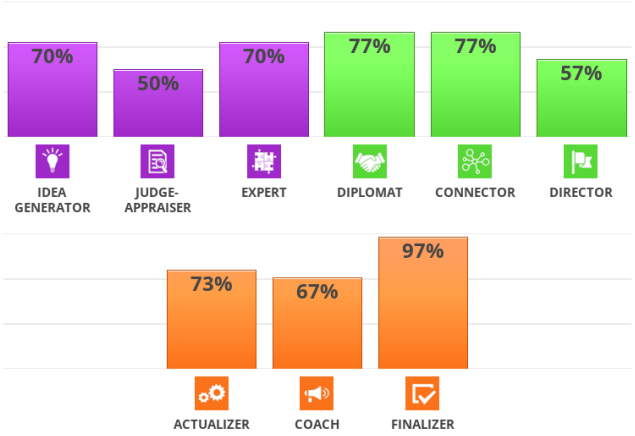


Figure 25: Frankie's Belbin test results

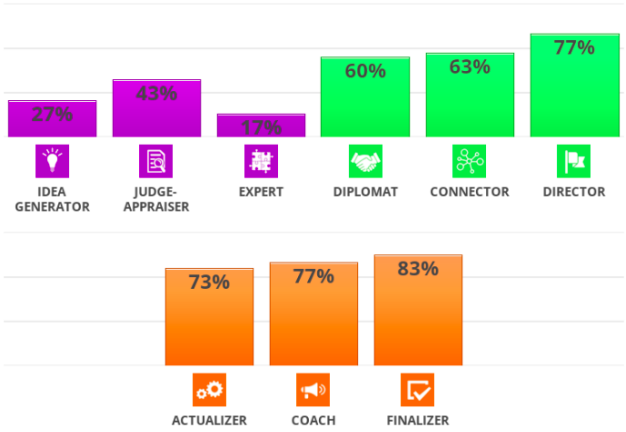


Figure 26: Gioele's Belbin test results

C Functional hierarchy tree

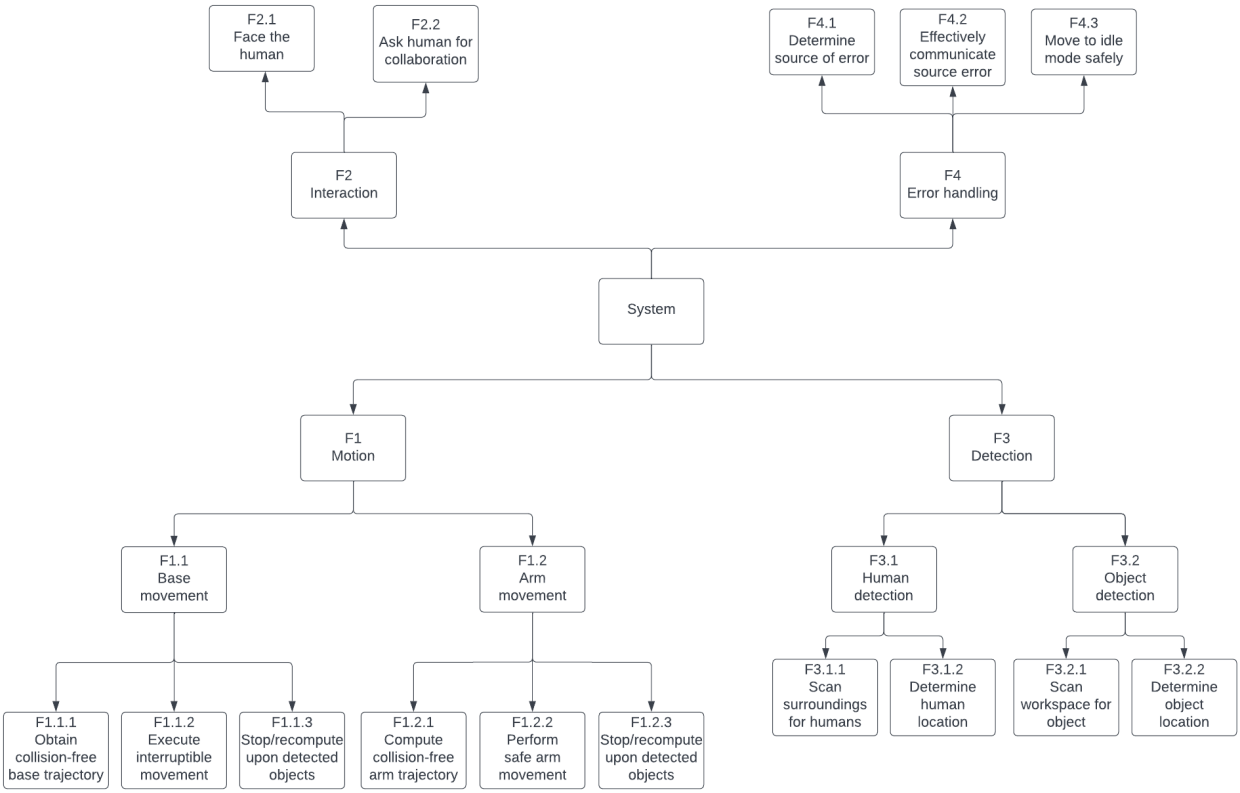


Figure 27: Functional hierarchy

D Activity diagram

D.1 Main activity diagram

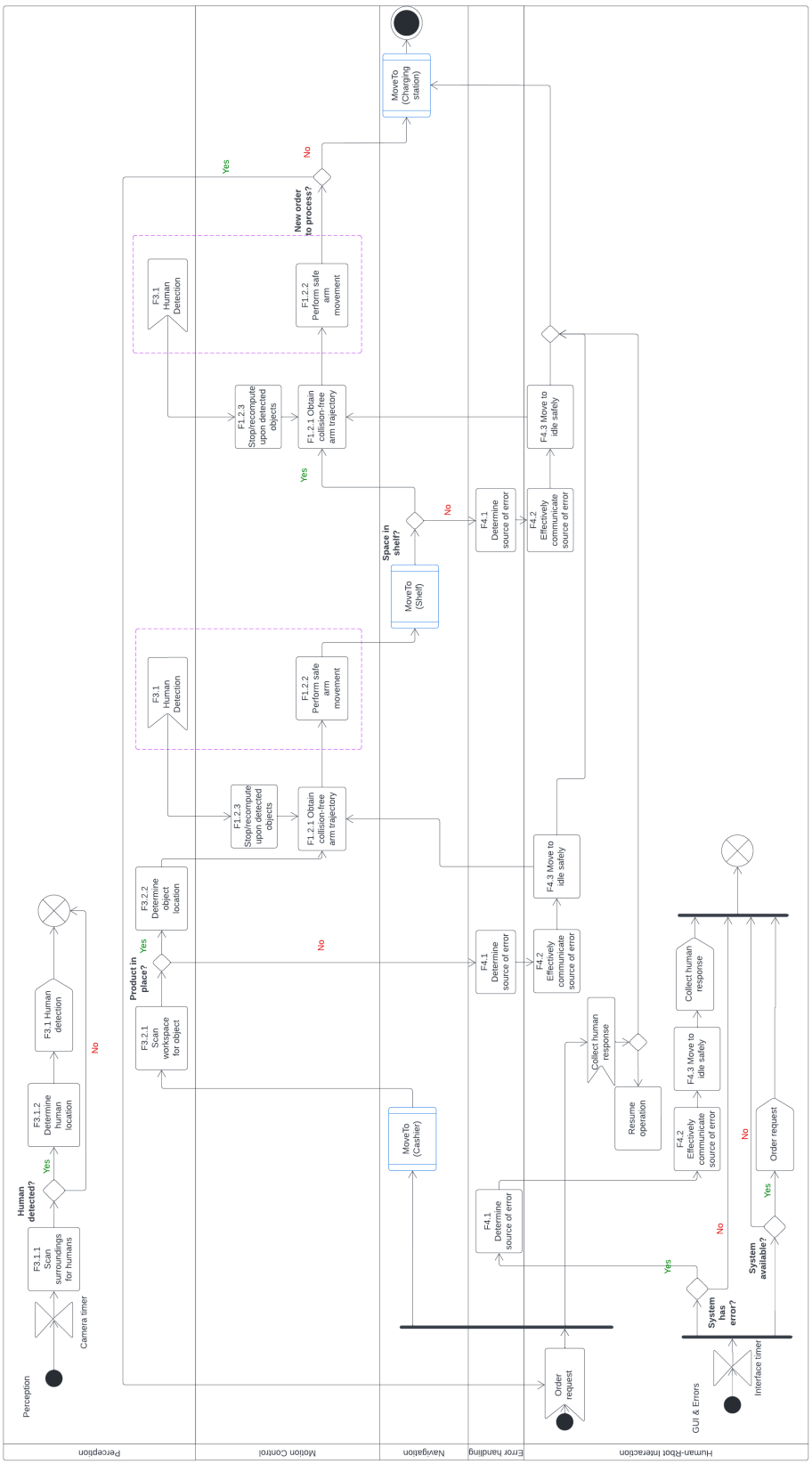


Figure 28: Full activity diagram

D.2 Main activity diagram (compressed view)

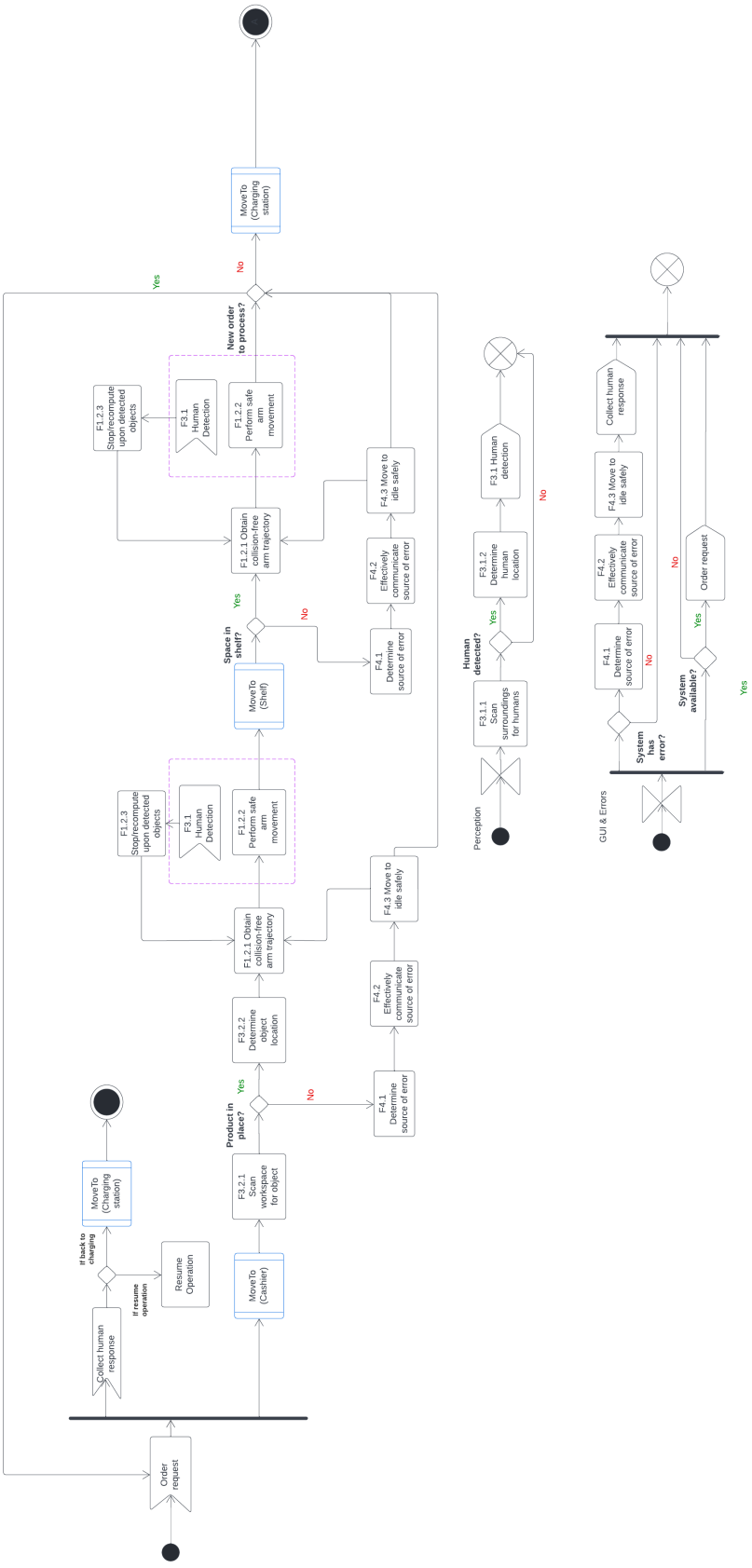


Figure 29: Full activity diagram without swimlanes

D.3 Macro state for movement

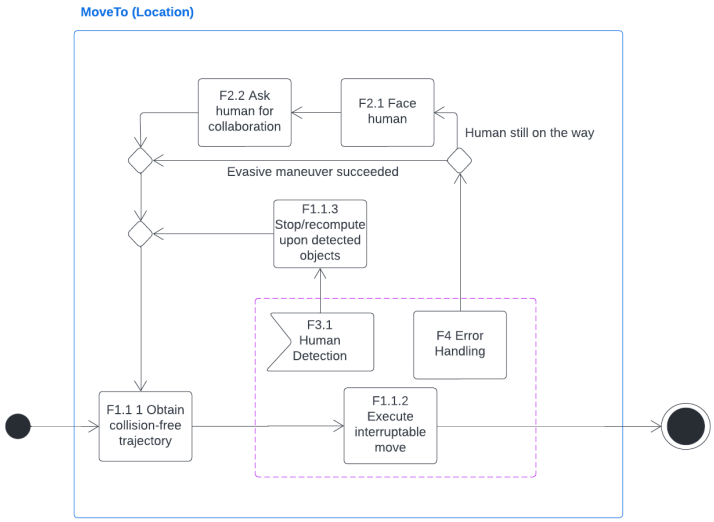


Figure 30: Activity diagram MoveTo macro state

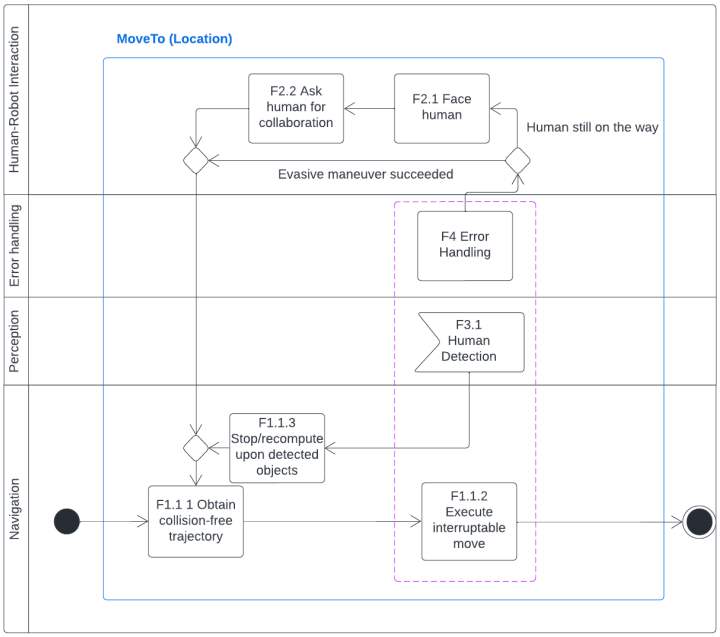
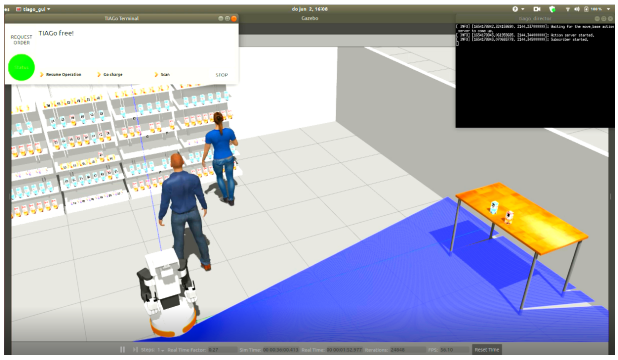


Figure 31: Activity diagram MoveTo macro state with swimlanes

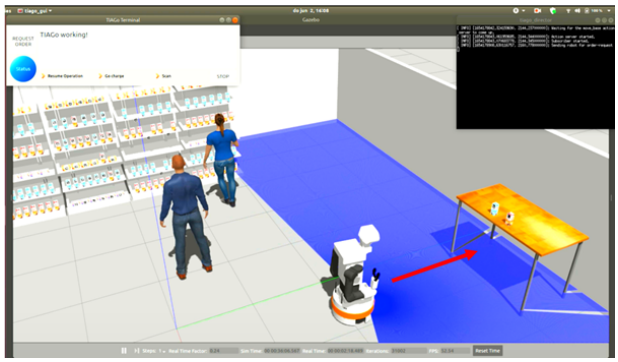
E Captures of robot behaviour in simulation



(a) Robot waiting for order

```
order_req: False
resume: True
scan: False
charge: False
level: 0
product: -1
info_msg: "TIAGo free!"
```

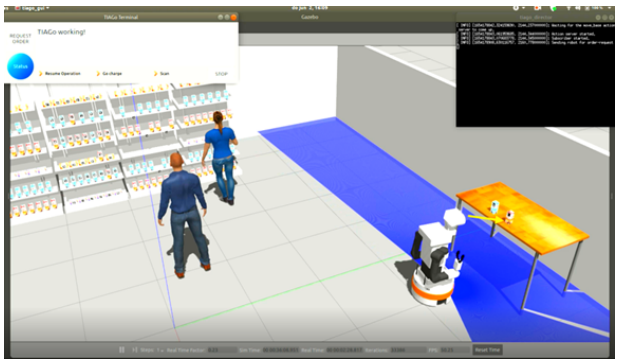
(b) Corresponding parameter description



(c) Robot moving to the pickup location

```
order_req: True
resume: True
scan: False
charge: False
level: 0
product: -1
info_msg: "TIAGo working!"
```

(d) Corresponding parameter description

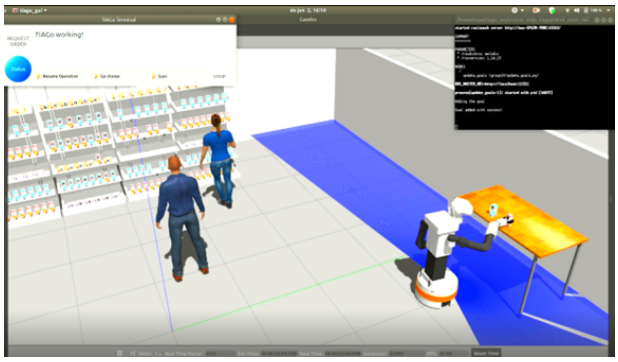


(e) Robot at the pickup location, prior to performing the pick-and-place

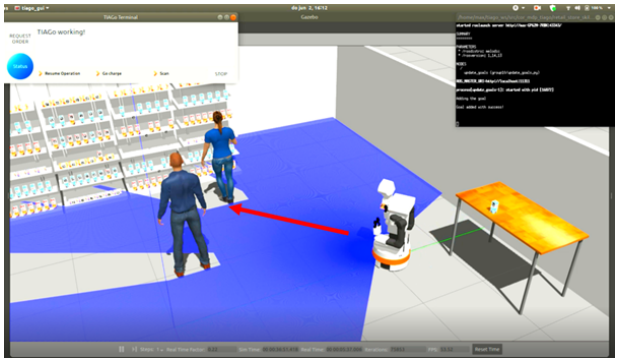
```
order_req: True
resume: True
scan: False
charge: False
level: 0
product: 0
info_msg: "TIAGo working!"
```

(f) Corresponding parameter description

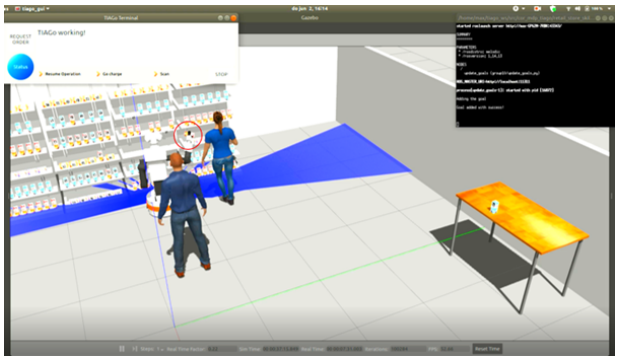
Figure 32: Different stages of the normal operation of the robot for executing an order request.



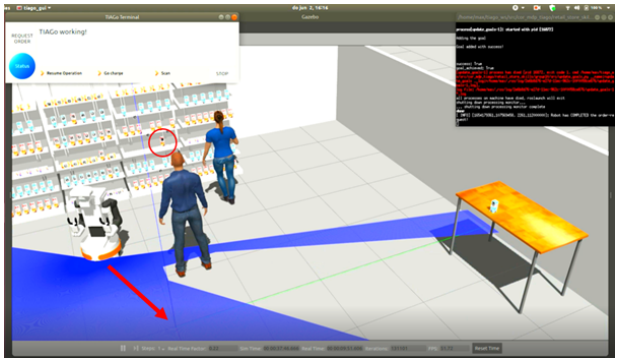
(a) Robot picking up the product



(b) Robot moving to shelf

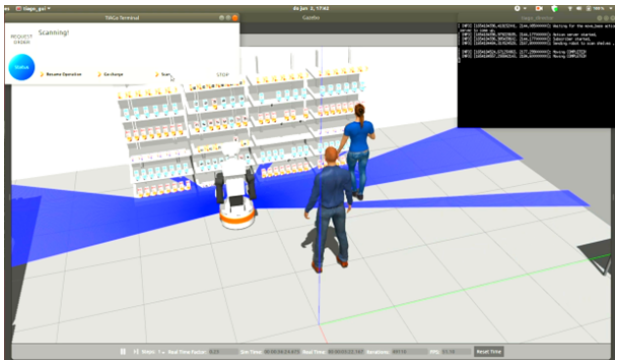


(c) Robot placing the product in the correct shelf



(d) Robot goes back to charging station

Figure 33: Different stages of the PDDL's pick and place.

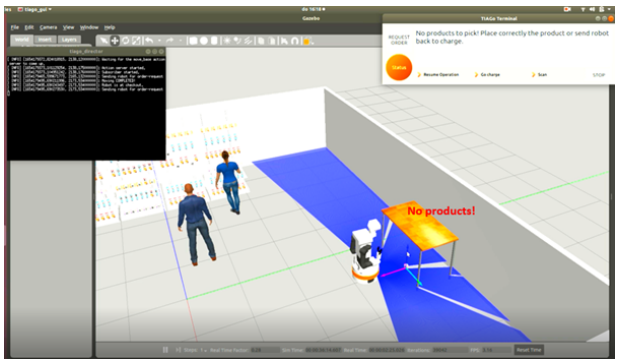


(a) Robot scanning the shelves

```
order_req: False
resume: True
scan: True
charge: False
level: 0
product: 15
info_msg: "Scanning!"
```

(b) Corresponding parameter description

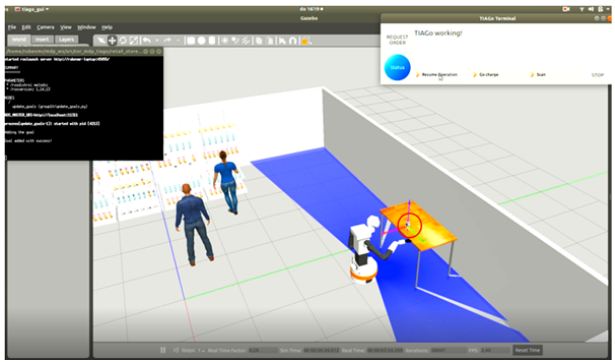
Figure 34: Robot scanning the shelves one by one.



(a) Robot at the checkout when no products are present

```
order_req: True
resume: False
scan: False
charge: False
level: 1
product: -1
info_msg: "No products to pick!
           Place correctly the product or
           send robot back to charge."
```

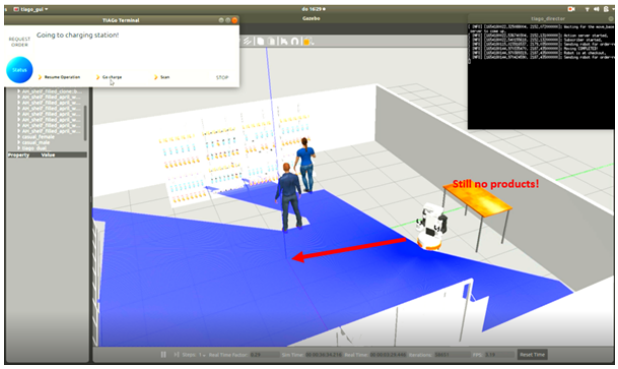
(b) Corresponding parameter description



(c) Robot resuming operation

```
order_req: True
resume: True
scan: False
charge: False
level: 0
product: 0
info_msg: "TIAGo working!"
```

(d) Corresponding parameter description

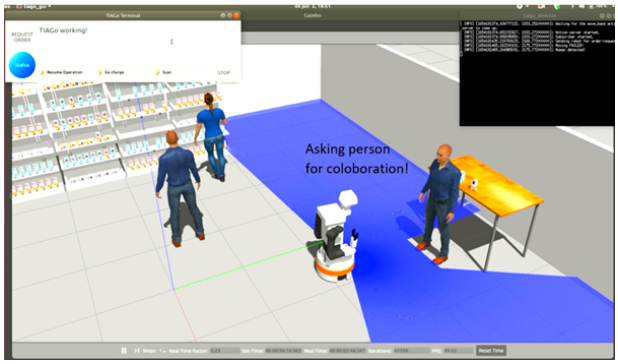


(e) Robot moving back to charging station after encountering missing objects

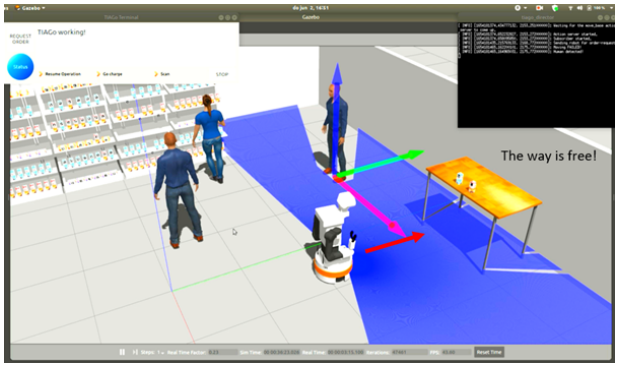
```
order_req: False
resume: True
scan: False
charge: True
level: 0
product: -1
info_msg: "TIAGo working!"
```

(f) Corresponding parameter description

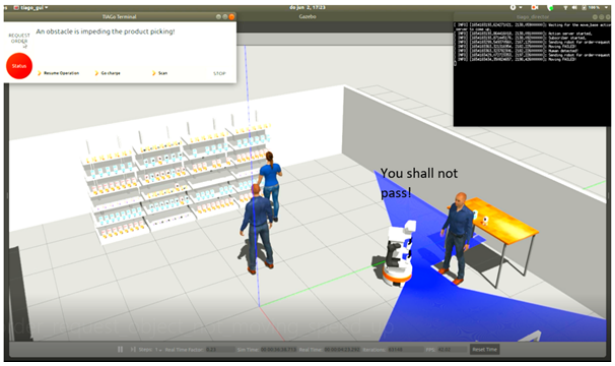
Figure 35: Different fall backs when no products encountered at the checkout.



(a) Robot interacting with a human using text-to-speech



(b) Robot resuming operation

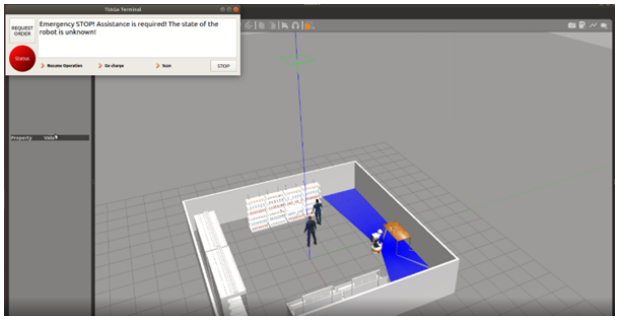


(c) State of the robot upon failed interaction

```
order_req: False
resume: False
scan: False
charge: True
level: 2
product: -1
info_msg: "An obstacle is impeding the product picking!"
```

(d) Corresponding parameter description for Figure 36c

Figure 36: Different fall backs when people are obstructing the movement



(a) Robot after the emergency (STOP) button of the GUI is pressed.

```
order_req: False
resume: False
scan: False
charge: False
level: 2
product: -1
info_msg: "Emergency STOP! Assistance is required
           the state of the robot is unknown!"
```

(b) Corresponding parameter description

Figure 37: State of the robot when the emergency stop button in the terminal is pressed.

F Full rqt graph

The rqt_graph displayed in Figure 5 depicts the locations in the graph corresponding to the nodes created by us or directly linked to them. To inspect them more in detail, Figure 39 is provided in vector format, enabling to zoom in to inspect the names of the topics and nodes.

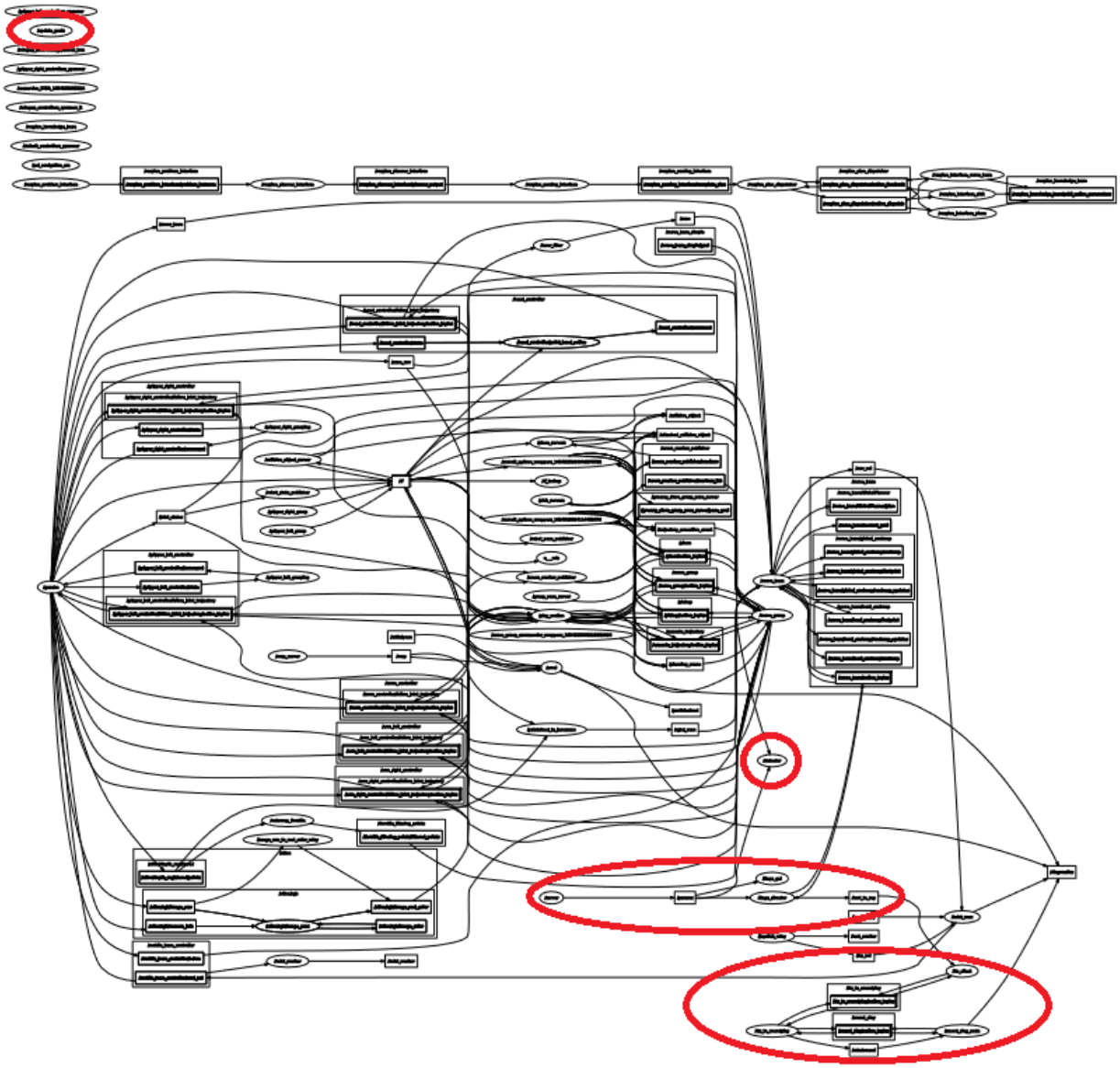


Figure 38: Complete rqt graph with the nodes developed by us indicated in red circles. In order to better inspect the graph, please refer to Figure 39

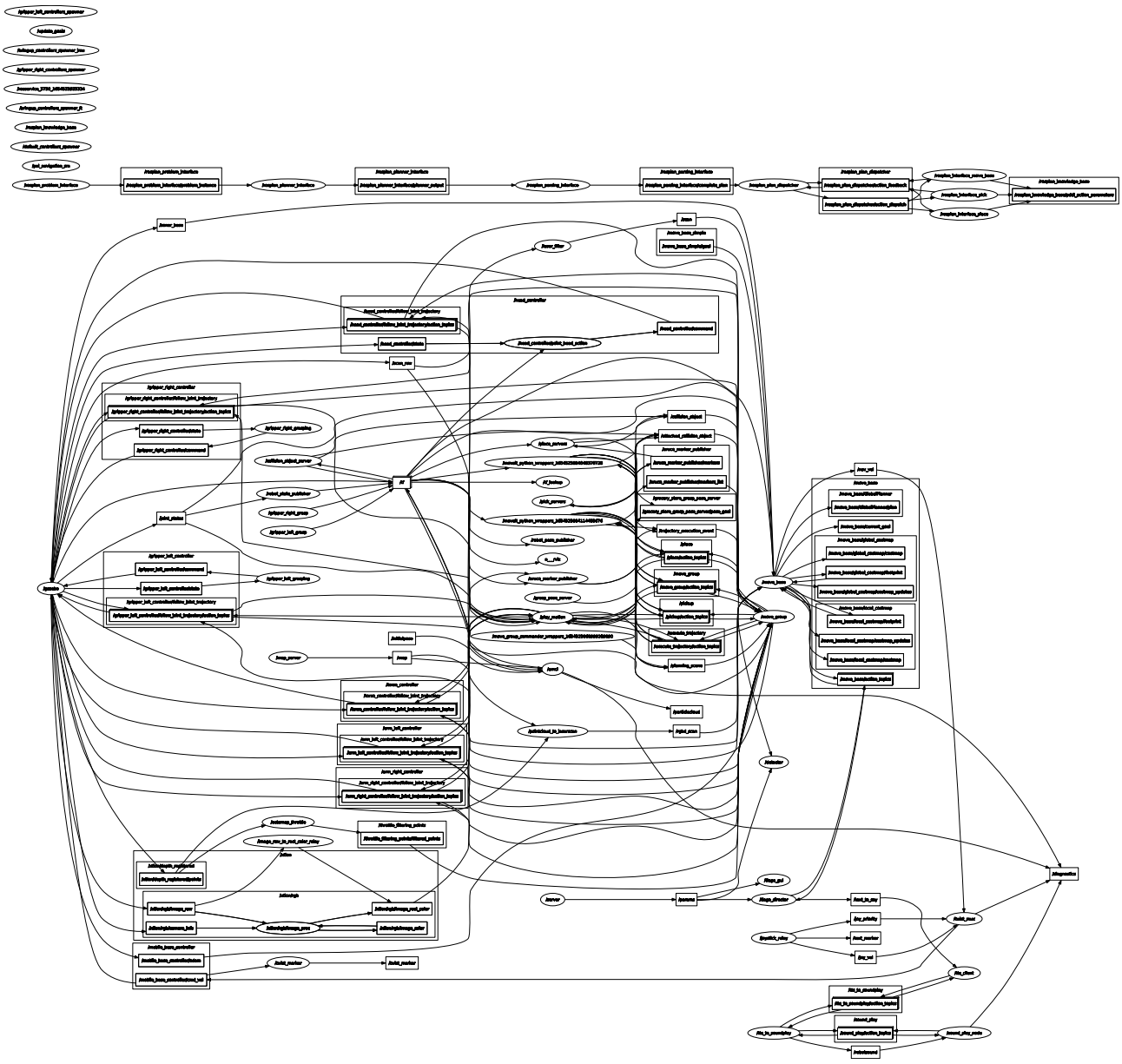


Figure 39: The complete rqt graph of the simulation