UNIVERSITY OF TARTU

Faculty of Science and Technology

Institute of Technology

Robotics and Computer Engineering Curriculum

Fabián Ernesto Parra Gil

IMPLEMENTATION OF ROBOT MANAGER SUBSYSTEM FOR TEMOTO SOFTWARE FRAMEWORK

Master's Thesis (30 ECTS)

Supervisors:

Robert Valner

Karl Kruusamäe

Tartu 2020

IMPLEMENTATION OF ROBOT MANAGER SUBSYSTEM FOR TEMOTO SOFTWARE FRAMEWORK

Abstract:

Robots provide an opportunity to spare humans from tasks that are repetitive, require high precision or involve hazardous environments. Robots are often composed of multiple robotic units, such as mobile manipulators that integrate object manipulation and traversal capabilities. Additionally, a group of robots, i.e., multi robot systems, can be utilized for solving a common goal. However, the more elements are added to the system, the more complicated it is to control it. TeMoto is a ROS package intended for developing human-robot collaboration and multi-robot applications where TeMoto Robot Manager (TRM), a subsystem of TeMoto, is designed to unify the control of main robotic components: manipulators, mobile bases and grippers. However the implementation of TRM was incomplete prior to this work, having no functionality for controlling mobile bases and grippers. This thesis extends the functionality of TeMoto Robot Manager by implementing the aforementioned missing features, thus facilitating the integration of compound robots and multi-robot systems. The outcome of this work is demonstrated in an object transportation scenario incorporating a heterogeneous multi-robot system that consists of two manipulators, two grippers, and a mobile base.

Keywords: Multi-robot system, Compound robot, Object transportation, ROS.

CERCS: T125 Automation, robotics, control engineering

ROBOTITE HALDURI ALAMSÜSTEEMI VÄLJATÖÖTAMINE TARKVARARAAMISTIKULE TEMOTO

Lühikokkuvõte:

Robotid võimaldavad aidata inimesi ülesannetes mis on eluohtlikud, nõuavad suurt täpsust või on üksluised. Üks terviklik robot koosneb tihtipeale mitme eri funktsionaalsusega alamrobotist, millest näiteks mobiilne manipulaator on kombinatsioon mobiilsest platvormist ja objektide manipuleerimise võimekusega robotist. Roboteid saab rakendada ülesannete lahendamisel ka mitme roboti süsteemina, kuid robotite hulga suurenemisel suureneb ka nende haldamise keerukus. TeMoto on ROSi kimp, mis hõlbustab inimene-robot koostöö ja mitme roboti süsteemide arendamist. Robotite haldur on TeMoto alamsüsteem, mis aitab käsitleda mobiilseid platvorme, manipulaatoreid ja haaratseid ühtse tervikliku robotina. Käesolevale tööle eelnevalt puudus Robotite halduril mobiilsete platvormide ja haaratsite haldamise võimekused, mille väljatöötamine oli antud töö peamiseks eesmärgiks. Töö tulemusena valmis TeMoto Robotite halduri terviklik lahendus, mille funktsionaalsust demonstreeriti objekti transportimise ülesande lahendamisel, kaasates kahest manipulaatorist, kahest haaratsist ja mobiilsest platvormist koosnevat heterogeenset mitme roboti süsteemi.

Võtmesõnad: mitme roboti süsteem, liitrobot, objektide transport, ROS.

CERCS: T125 Automatiseerimine, robootika, juhtimistehnika

Contents

List of Figures

List of Tables

1. Introduction

Robots represent enormous potential in several application domains such as robots in medicine and healthcare, where high degree of precision is required resulting in shorter recovery times and more reliable outcomes in surgery procedures [1] [2], Robots are used in production and manufacturing industries to increased productivity, flexibility, versatility and safety [3], or in exploration and rescue missions in emergency areas [4], among other [5].

With a wide structural diversity in terms of shape, size, and design, robots provide different capabilities that range from manipulation through robotic arms and locomotion with mobile bases, to a combination of these capabilities known as compound robots e.g. mobile manipulators, which integrate manipulation and mobility. Furthermore, in order to accomplish complex tasks, robots can be used to cooperate with each other in multi-robot system scenarios.

The fact of having different capabilities, or simply having components that do not necessarily belong to the same manufacturer but still need to operate together, means that the way of controlling them may vary from one to another. That is why it is important to have a unifying platform basis that enables the integration of different drivers and controllers such as ROS, which has become a one of the most practical and popular frameworks for the development of advanced robot systems.

ROS utilizes tools and plugins such as MoveIt! and Navigation to generate collision-free trajectories to control robots. However, ROS tools are still limited, missing a systematic approach for integrating and controlling compound robots and other multi robot systems. For instance, motion planning for manipulators and mobile robots is handled by two separate libraries (MoveIt and Navigation respectively) and thus, there is no straightforward toolset for controlling mobile manipulator robots in ROS.

TeMoto is a software framework that streamlines the development of human-robot collaboration and multi-robot applications. It leverages the tools and data distribution infrastructure of ROS to provide an additional software layer for integrating and controlling resources, such as sensors, actuators, algorithms and robots. TeMoto is provided with a number of managers that are in charge of maintaining knowledge about the resources. In the case of robots, Temoto Robot Manager (TRM) is intended to provide rapid integration of robotic actuator systems, facilitating the implementation of a single or multiple robots; nonetheless, it is missing major key functionalities for controlling mobile robots and grippers. The main task of the thesis is to implement the TRM to support controlling all major robot categories such as mobile platforms, manipulators, and grippers.

This thesis is divided into 7 chapters. The concept of multi-robot systems (MRS) and compound robots is introduced in chapter 2 while an overview of the previous work related to TeMoto and TeMoto Robot Manager is given in chapter 3. Chapter 4 discusses the requirements for this work. The structure of implemented work is discussed in chapter 5 and a multi-robot demo using TRM is presented in chapter 6. The thesis concludes with the remarks on the TRM in chapter 7.

2. Literature overview

With a wide structural diversity in terms of shape, size, and design, robots provide different capabilities that range from manipulation, locomotion to grasping operations. These capabilities are often integrated depending on the task at hand. For example a robot could be combined of a manipulator and a mobile base for solving mobile manipulation tasks such as object retrieval. Moreover, a group of robots can be utilized for scenarios such as area coverage, foraging and object transportation. This section collects the main contributions of the literature related to MRS, compound robots and frameworks that allow developing robotics applications.

2.1. Multi robot systems

A Multi-robot System (MRS) refers to a group of two or more independent robots that work together to achieve a shared goal. These systems tend to be comprised of robots of a simpler design combined together in a modular fashion, in order to provide convenient solutions in terms of cost, performance, efficiency, and reliability [6]. The use of MRS has been described since 1986, such as the project present by Freund et al, to find a path with online collision avoidance, for three stationary robots in an assembly automation application and for scenarios with mobile robots surrounded by moving obstacles. They integrated the dynamics of all robots in a hierarchical structure, and for the collision detection [7].

Farinelli et al. propose a taxonomy classification of works on Multi-Robot Systems according to two groups, *Coordination* as depicted in Figure 1a, and *System Dimensions* in Figure 1b [8]. The first one is a hierarchical structure that represents the interaction between robots, and the second takes into account the system features that influence team development. The cooperation refers to the ability of the system to cooperate with other systems in order to accomplish a specific task [8]. Knowledge characterizes how aware are the robots of each other within the system [9]. The coordination level refers to the set of rules that the robots follow to interact with each other and move cohesively [10]. The last level represents the way the decisions are made within the system rather centralized or distributed[11].

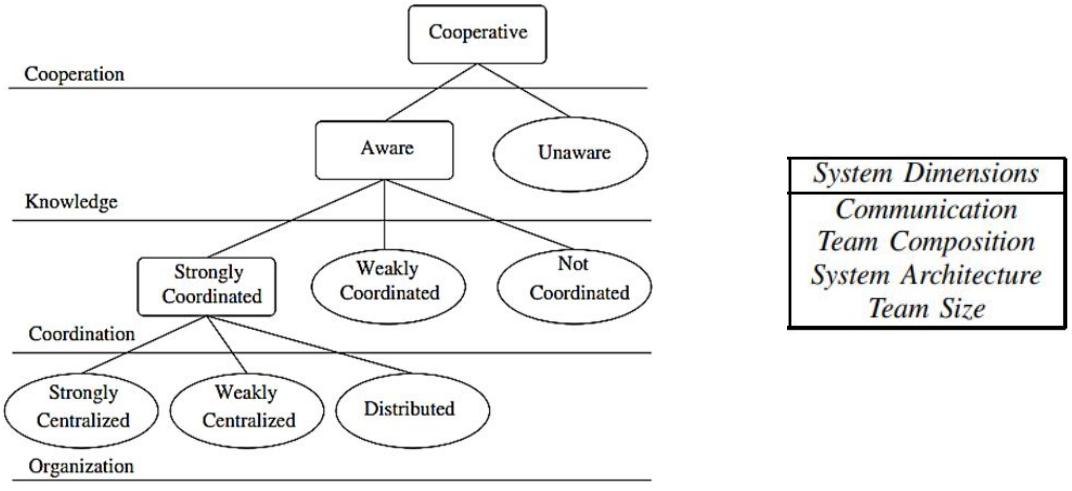


Figure 1. *Classification of Multi Robot System based on (a) Coordination and (b) System dimensions*

Additionally, in the *System Dimension* classification Figure 1b, it is considered that cooperation is obtained through *communication mechanisms* that allow robots to exchange messages directly or indirectly [11]. The *team composition* classifies the MRS according to their heterogeneity, divided in two classes, *heterogeneous* and *homogeneous* teams. In the first one, the robots differ either in their hardware devices or in the software. In the case of homogeneous MRS, all of the members have the same software and hardware layout [8]. The last characteristic refers to the *team size* or number of robots acting in the same environment.

For example, using a large team of homogeneous robots is known as swarms, that are simple agents that perform complex collective behaviors [12]. Swarms are inspired by natural systems in which the collective behavior emerges from the interaction between the robots and with the environment [13]. Its control is normally a distributed and decentralized system.

The main uses of swarm system lie in sensing, information exchange, motion, and achieving dedicated collective actions. In [14], investigate and develop some principles of adaptation and evolution for robots that can dock with each other as in Figure 2. and symbiotically share energy and resources, for a collective interaction with the physical world.

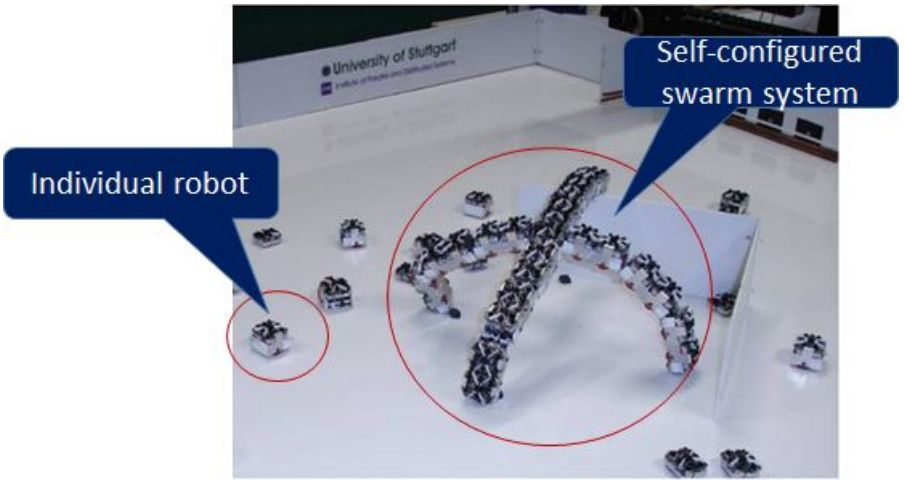


Figure 2. *Adaptation and evolution for symbiotic swarm robot organisms* [14]

2.1.1. Compound Robots

Traditionally MRS are composed of individual hardware platforms with no mechanical interconnection. Yet a single robot could be made out of a number of individual robotic systems such as a manipulator arm integrated on top of a mobile platform. To the best of the author's knowledge, there is no generally accepted terminology to refer to such mechanically coupled systems. However, a number of research projects and companies refer to them as compound robots [15] [16] [17] because they consist of multiple robots or subsystems which are responsible for manipulation, navigation and gripping capabilities (hereinafter referred to as “features”), assembled into one *compound* body.

Commonly appearing compound robots are mobile manipulators, which are composed of a mobile base that provides locomotion, and one or more robotic arms attached to the base for manipulation tasks (e.g., Figure 3). Mobile manipulators are widely applied in the domains of construction, transportation, space exploration, military operation, etc. [18].

In [19], it is proposed a technique to control the feature of mobile manipulators simultaneously where the mobile base is seen as an extra joint for the manipulator and it is included in the kinematics to drive the robot following a desired trajectory from a start point to the end goal. In [20], Ramzy Ali et al. decompose the problem in subsystems, implementing a logic to navigate an environment towards a desired goal avoiding collisions and command the arm after the base has reached the goal, using a separate controller for picking up objects. In [21] Nagatani et al. present a motion planning algorithm to draw large objects on a wall keeping the locomotion controller independent from the manipulator controller but maintaining a cooperative motion by communication between both controllers.

An integration of different features in a single robot is presented in [22] [23], used for haptics-enabled capabilities for teleoperated mobile manipulation robots that support human interaction in explosive disposal missions depicted in Figure 3.

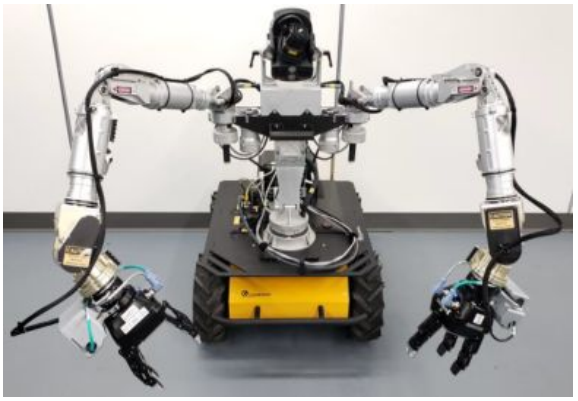


Figure 3. *Example of a compound robot: teleoperated bimanual mobile manipulator* [23] *compound of two robotic arms, a clearpath platform and two grippers*

2.2. Applications of Multi robot systems

MRS's can be utilized on search and rescue to increase communication bandwidth between robots and reduce energy consumption [24]. Using more than one robot, it is possible to achieve a greater coverage area because the agents can be distributed over the space. Lopez-Perez et al. [25] propose a scene partitioning scheme with a distributed algorithm to minimize communication thus energy consumption, and assign weights to the frontiers for the representation of each zone, which are combined with the obtained from other robots to obtain a full map. Wurm et al. consider the exploration with coordinate marsupial robots, that is to say, teams of robots that are able to deploy and retrieve other robots [26]. These heterogeneous robots require actions and advanced planning mechanisms to be implemented in zones of the environment where the bigger robots cannot enter but the smaller robots can. In [27] developed a framework to classify terrain using a heterogeneous team of legged robots and a vibration-based terrain identifier (accelerometers and gyroscopes on the robot) and thus avoid dangerous slippery regions.

MRS have been applied for surveillance tasks as the project presented by Pennisia et al, using a system to identify the presence and position of people in an indoor environment. A set of heterogeneous sensors and robots patrolling the space triggering an alarm when abnormal activity is detected [28]. Nowadays using different types of robots like Unmanned Aerial Vehicles (UAV), surveillance can be extended to outdoor areas like in [29], where a global

architecture was developed to manage services and control systems with common functions, reducing development and operating costs. It was tested on an automated UAV-based surveillance system compound by drones, sensors and other devices. In an interactive user interface they could give waypoint missions to the UAVs and track persons using a container resource for the status, control, detection and streaming data.

MRSs have been successfully applied in disaster response scenarios such as the Fukushima nuclear accident in 2011, where exploration tasks were performed using a mobile robot, and a manipulator was implemented to facilitate the installation of sensors. One to measure radiation levels and a water gauge in the basement of the reactor buildings to determine flood levels [4]. In [30], propose a solution to evaluate the damage of infrastructures of disaster environments and locate dynamic objectives of interest on it. A system that stores tasks and enables automatically these behaviors among a group of robots was tested in a controlled laboratory environment, simulating scenarios where communication is limited or its reliability is low.

A combination of several homogeneous mobile manipulators can be seen in Figure 4, in a system that requires moving objects with different shapes among other solid obstacles. [31]. Antonio Petitti et al. propose two ways of controlling: controlling the twist of the manipulated object, having priori knowledge of the physical parameters of the load, and an approach in which each robot does not have any information about the load [32].

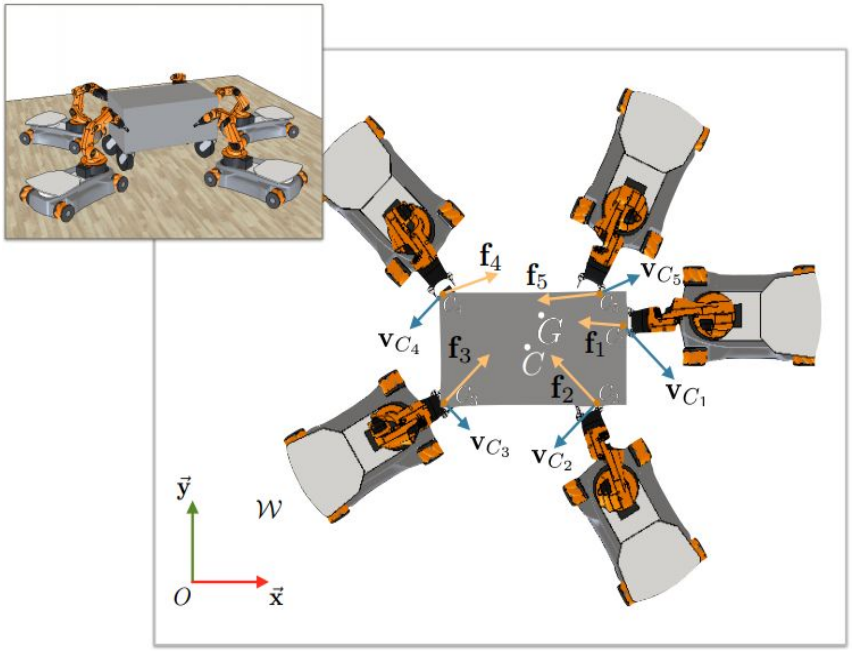


Figure 4. *Homogeneous MRS team of five KUKA youBots cooperatively transporting a load* [32].

As can be seen, MRS range of different application domains is broad, with systems that benefit from the diverse capabilities of its members such as compound robots, or a set of multiple robots interacting with each other.

2.3. Main challenges for Multi-Robot Systems

Multi-robot systems offer several advantages in terms of cost, performance, time, task inherently distributed, binding several resources, parallelism, and redundancy [6]. There is a wide range of domains in which a multi-robot system can be applied [33], and depending on the application, different tasks represent different requirements.

Manuela Veloso [34] exalts as main issues for MRS is the constraints in communication, due the MRS require general mechanism for sharing information. Baele et. al. [35] present a solution for avoiding the congestion of the *communication* network when multiple worker robots transmit data simultaneously and data packets may be lost. Using a heterogeneous configuration with five types of robots, each with specific roles. The smaller robots in charge of collecting data and the bigger one use an aggregator module to reduce duplicated data gotten by smaller robots.

One of the main challenges in mobile manipulators is vehicle-arm *coordination* due the dynamics of the arm and the base not only differs, but also interacts. Besides, the addition of the mobile base introduces kinematics redundancy complicating the integration of techniques for navigation, planning and control to achieve efficient motion [19]. The redundancy results when they exceed the total DOFs that are strictly needed to perform a task, resulting in multiple joint configurations to reach the same end-effector pose.

In [36], Cressel Anderson et al. point out that the challenges with mobile manipulators are the *integration* to achieve real-time performance, and the *control* of their features to produce the desired outcome, because the mobile platform and the manipulator each one have their own control system. There are applications where it may be sucient to move them separately, sending the mobile platform to a desired location, and once reach the goal, move the manipulator, but other tasks require a closer coupling of these control systems.

For the aforementioned challenges and the fact that there are systems that implement more than one robot to complete a task, we therefore need a platform or unified *frameworks* capable of dealing with different elements, with information about how to access their drivers and controller, and an interface to provide interaction with the user.

A framework is a platform or layered structure for developing applications, integrated with a collection of components that collaborate to produce a reusable architecture [37]. It helps to manage hardware devices and interact with the software system. A framework is composed of predefined classes, functions and special tools that can be used through application programming interfaces (API), that provide access to the elements supported by the framework [38].

A real-time visual-based system was implemented at Simon Fraser University [39], to select and command individual robots in a multi-robot environment. The election of the robot to be commanded was based on a non-verbal communication, resulting from a score evaluating a facing engagement algorithm. A single human in front of a group of robots, looking to the desired robot (as a static gesture), enables the face detector and with a dynamic gesture can interact with the desired robot, in their case, the user moving the hands can control the selected mobile base to the right or the left.

Some design principles of general software to reduce robotic applications design and implementation time were presented in [40], where the user can control a set of robotic arms with any set of sensors. The modular architecture was achieved by encapsulating the features into structures and a generic interface that allows to change their attributes. Using a server and client model, they could control an external device whether it is connected directly or remotely.

In [41], Lopez et al. propose a centralized system for controlling mobile single and multi-robot applications, tested on applications like tour guide robots and transportations tasks in hospitals. All the robots and all the users are connected to a central server using wireless communication as shown in Figure 5. The server executes off-board remote control like planning for multiple robots, scheduling and resource sharing among other tasks. Each robot has an on-board local control for reactive behavior, that contains an executive layer which performs a task decomposition in basic tasks and a Robot-Web interface layer that represents a set of processes to interact with the users and connect to the Central Server [41].

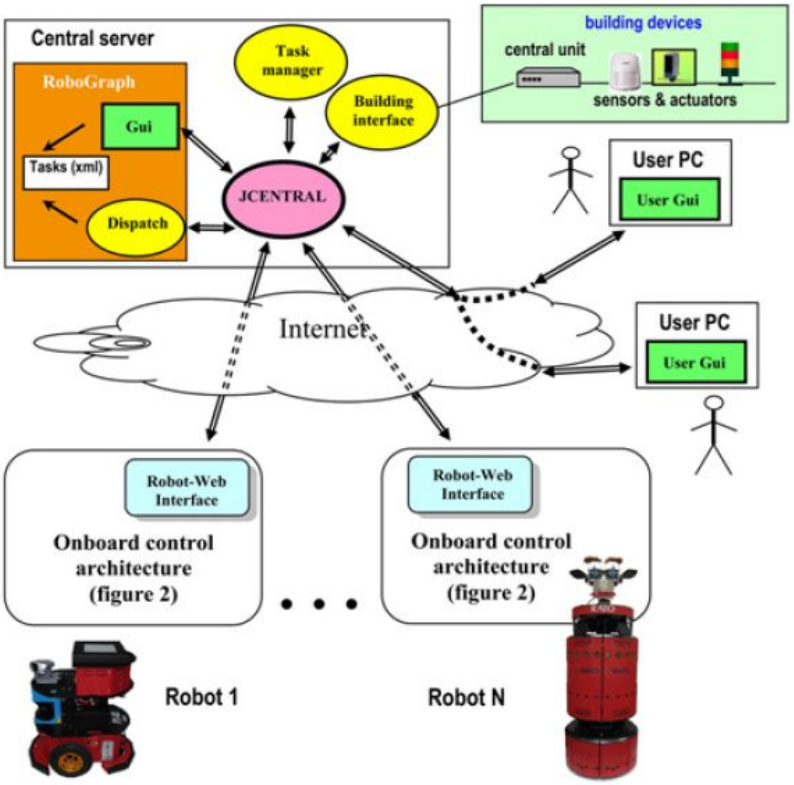


Figure 5. *Example of a multi-robot and multi-user framework architecture* [41]

2.4. Robot Operating System

The Robot Operating System (ROS) [42] is a robotics application development and integration framework which provides a *publisher-subscriber* based data distribution infrastructure that allows software applications, such as sensor drivers and data processing algorithms, to exchange data.

There are three forms of communication in ROS: topics, services, and actions as depicted in Figure 6. Topics are buses over executable programs exchange messages. Topics can have multiple publishers and subscribers [43]. Services have a request and response interactions. A server provides a service and a client uses the service by sending the request and awaiting for the response [44]. Similarly, actions have a request and response interaction, but compared to services, actions are used when the task takes a long time to execute. Actions have messages on which they communicate: A *goal* is the task to accomplish, the *feedback* allows a client to track the progress of a goal and a *result* is sent upon completion of the goal [45].

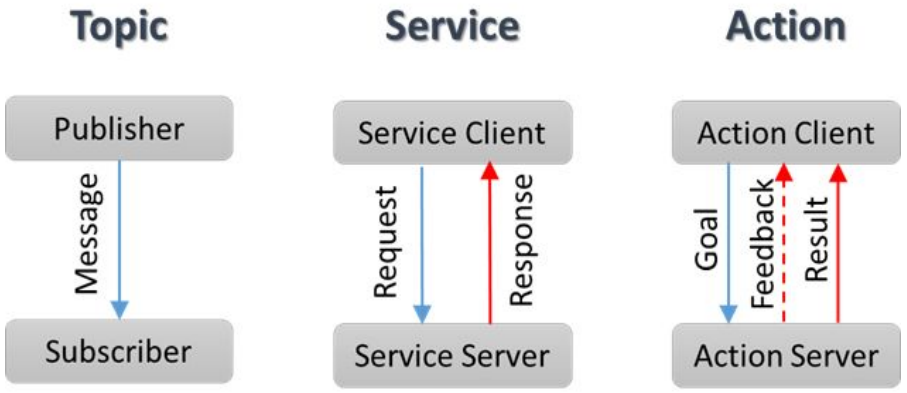


Figure 6. *Forms of communication in ROS*

ROS has been widely adopted by the robotics community and thus a wide variety of ROS based tools and libraries have been created to simplify the creation of complex and robust robot behavior [46]. The primary tools for controlling manipulators and mobile bases are MoveIt! [47] and ROS Navigation [48] respectively. MoveIt provides functionalities for planning collision-free trajectories for serial manipulators while Navigation helps to move a mobile robot with collision avoidance.

2.4.1. MoveIt

Moveit is a ROS package for manipulation purposes such as pick and place or grasping operations [49]. Currently there is a wide variety of robots that support MoveIt [47]. Creating a MoveIt! support for custom robots is simplified via MoveIt! Setup Assistant [50].

MoveIt provides a standard framework for integrating motion planners and inverse kinematics solvers and exposes their functionality via ROS based communication interface [51]. A motion planning request can be sent to indicate the arm to move to a different position, or to command the end effector to a new pose. To execute the planned trajectories on the robot, MoveIt instantiates a client to communicate with the hardware controller [52].

The motion planners allow to calculate an optimum path, or a sequence of valid configurations that the joints must perform to move the end effector gradually from the current pose to a goal pose [52]. The planner takes into account velocity and acceleration constraints, and it also has collision checking to avoid self-collision or a crash with some other objects in the world. Using a *FollowJointTrajectoryAction*, which is an action server running on the robot, MoveIt can communicate with the robot to execute the trajectories.

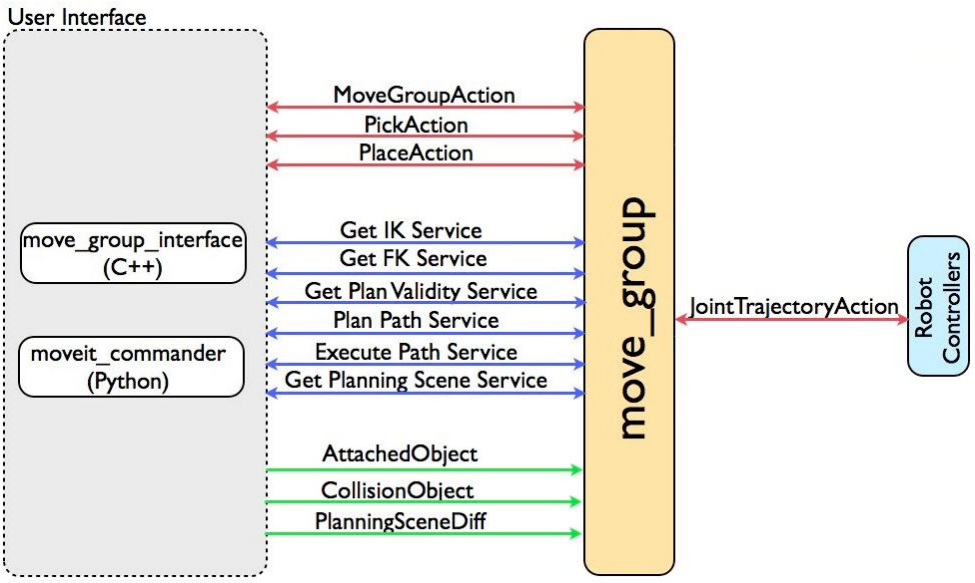


Figure 7 *Primary node of Moveit: Move_group architecture* [51]

The Figure 7 shows the architecture of MoveIt. The primary node that allows to connect all of those functionalities as plugins is called move_group. MoveIt provides ROS actions (red lines in Figure 7) and services (blue lines in Figure 7) for the user to use according to their needs. It is possible to use either C++ or Python based interfaces to command the move_group node [51]. MoveIt knows the kinematic capabilities of a robot via the unified robot description (URDF).

Additionally, MoveIt uses a concept called planning group, which is a set of joints or links in a robotic arm that plans together in order to achieve a goal position of a link or the end effector [53]. It is possible to define more than one planning group for robots with multiple arms as a compound robot e.g. a dual-arm setup or to control smaller sections of a manipulator.

2.4.2. Navigation

ROS Navigation is a framework that exposes functionalities of trajectory planners through a ROS based interface, using a collection of packages that enable mobile robots to move in the environment avoiding obstacles encountered along the way from its current position to a goal position [52]. It is designed to be as general-purpose as possible primarily meant for differential drive (two separately driven wheels placed on either side of the robot body) and holonomic wheeled robots (robot can move in any direction) [48].

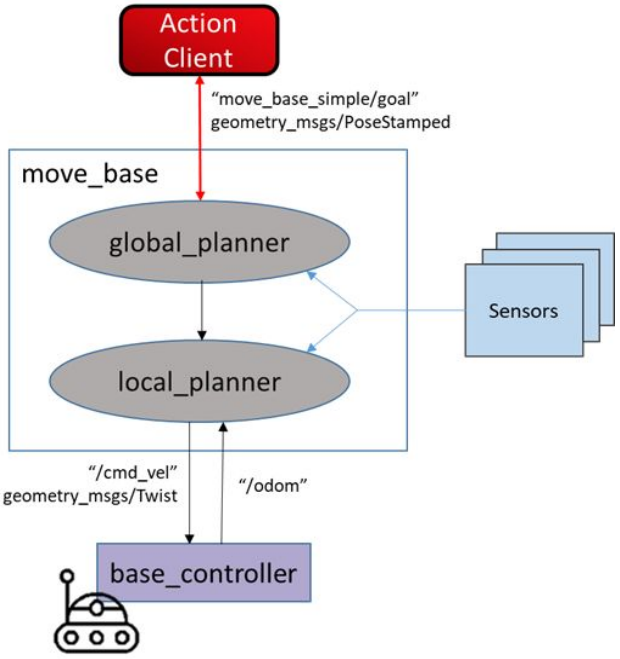


Figure 8. *Schematic diagram of ROS Navigation.* [53]

The move_base block shown in Figure 8 represents the core of Navigation because its function is to take goal poses, generate a trajectory and move the mobile robot from its current position to the goal position.

Inside of the move_base node, it links the global and local planners to adjust the behavior of the robot during the path planning [54]. The move_base takes input from odometry, that uses a motion sensor on the robot, e.g. encoders, to provide the estimated robot position publishing the data in an /*odom* topic. The sensor source is needed, due it allows to gather environment information and avoid obstacles in the world [54].

The output of the navigation is given in the ROS topic */cmd_vel* in the form of linear velocity and angular velocity using *geometry_msgs/Twist* message type, and the robot base controller converts those values into the equivalent motor speed to follow the trajectory [52].

In order to receive goal requests, the ROS Navigation uses ActionServer for communication with an ActionClient. Move_base is an implementation of a *SimpleActionServer*, which is an action server with a single goal policy, subscribed to a topic called *move_base_simple/goal* with a *geometry_msgs/PoseStamped* message type. A *SimpleActionClient* can send goal poses (information where the robot should move) to this server, and the move_base generates a trajectory making use of the global and local planners [45].

3. Previous Work - TeMoto

As described in chapter 2, there are several software frameworks for controlling robots where ROS is the most widely used robotics development platform. While ROS provides a data distribution infrastructure, it does not provide a methodology for organizing and managing the individual nodes, such as sensors and data processing algorithms, that comprise a robot [55]. This management is necessary in a number of cases, where energy conservation or system reliability is desired. For example a robot that is equipped with a variety of sensors, such as 3D LIDARS and cameras, might not need to access all of the sensors at all times. Thus having a programmatic way of managing (enabling, disabling, configuring) the given robotic resources allows designing robots that use energy and computation resources efficiently and adapt without manually reconfiguring the robot, e.g., maintaining visual feedback by switching from broken sensor to a working one. [56]

TeMoto[1] is a software framework that provides the aforementioned resource management capabilities. TeMoto (Figure 9) is developed upon ROS with a purpose of streamlining the development of human-robot collaboration and multi-robot systems. It leverages the tools and data distribution infrastructure of ROS to provide additional software layers for managing resources.

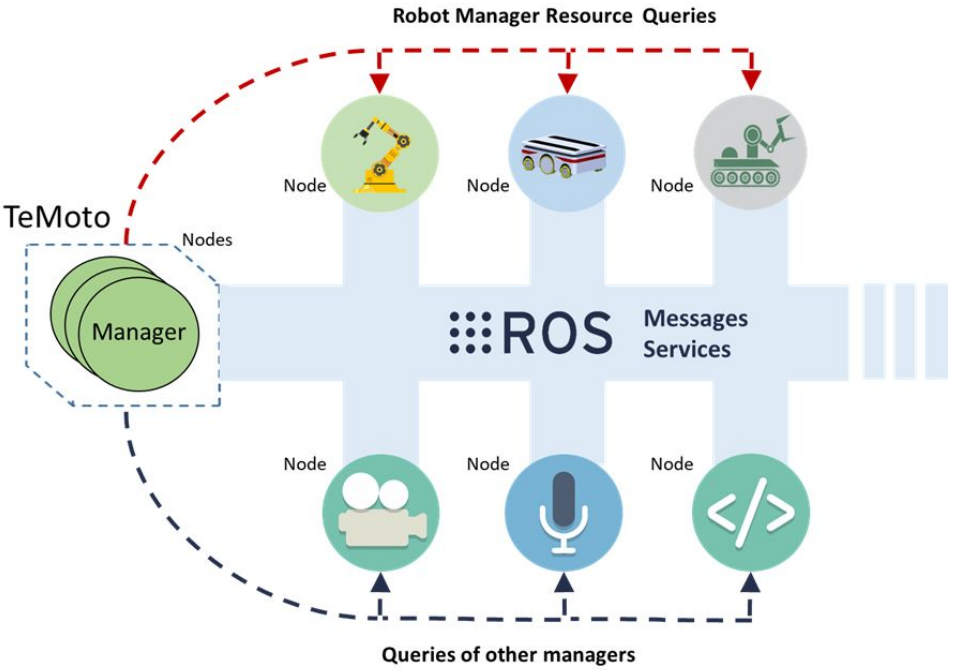


Figure 9. *Overview of the TeMoto framework, which manages robotic resources by utilizing ROS as a data distribution infrastructure. TeMoto comprises a set of manager nodes, each responsible for managing a specific set of resources where TeMoto Robot Manager manages robotic actuators.*

The TeMoto framework comprises a set of programs called *managers*, which manage *resources*, such as sensors, actuators, algorithms and robots [57] as depicted in Figure 9. The managers maintain knowledge about resources, including the status and availability of a given resource and they handle resource requests [55]. Managers expose a ROS based control

[1] https://github.com/temoto-telerobotics

interface which allows accessing the resources by user-defined programs or other managers [57]. Each manager is designed to manage a specific type of resource, e.g., Component Manager manages sensor and algorithm resources, whereas the Robot Manager manages robotic devices such as manipulators, grippers and mobile bases.

3.1. TeMoto Robot Manager

This section provides an overview of TeMoto Robot Manager[2] (TRM) in the state it was prior to this work. TRM is a manager which is designed to integrate different types of robots and provide the application developer a unified structure for controlling a robot as a whole unit, rather than controlling each feature separately. This helps to reduce time and complexity of controlling MRSs. This version of TeMoto has the capability of controlling robots with manipulation features depicted in Figure 10 as it was the only supported feature prior to this thesis. It can plan and execute a trajectory for the end effector of a specific robot.

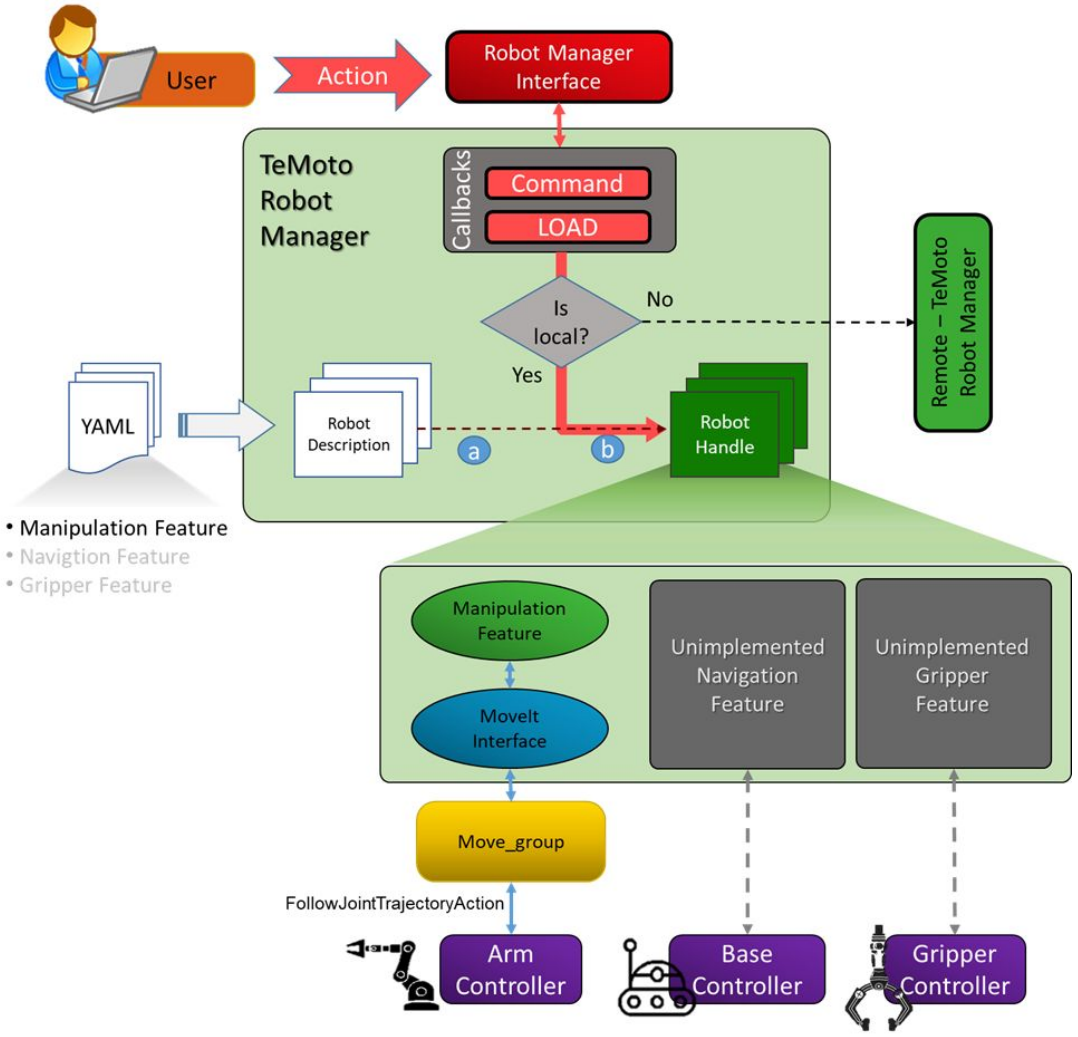


Figure 10. *Architecture of TeMoto Robot Manager with a representation about how a robot is (a) initialized or (b) commanded.*

The architecture or TRM depicted in Figure 10. Robot Manager Interface that allows other applications to access the functionalities of TRM, the Robot Manager able to read YAML

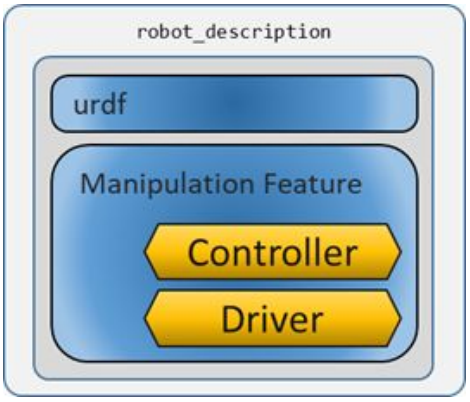
[2] https://github.com/temoto-telerobotics/temoto_robot_manager

description files and a robot handle which contains the implementation of the manipulation feature to command robotic arms. The commands pass from RMI to TRM, to the robot handler, that identify the manipulation feature, and make use of the moveit interface to command the robot.

TeMoto can control resources that are situated on separate physical machines, TRM can redirect requests to remote TRM. This implementation allows users to command different robots, simply by changing the name of the robot in the argument of the methods to command a different unit either locally or remotely.

All of the information related to the robots, including the definition of manipulation feature and the URDF, are outlined in a robot description YAML file that, at the user's disposal, can be stored together with the respective ROS package of the robot (one file for each robot), or into a single file with the description of various robots. The robot description describes all elements of the robot (Figure 11a) and provides TRM the necessary information to know how to access each robot. An automatic scan is performed on the different subfolders searching for robot description definitions, and TRM parses the information found to create them as a resource.

To initialize a robot (step “a” in Figure 10), TRM takes the values regarding the driver and controller as depicted in Figure 11b. The driver is an executable in charge of enabling direct control of real hardware, letting TeMoto and the robotic device communicate with each other. TRM makes use of the MoveIt! package for controlling manipulators, hence the controller in the robot description YAML outlines the settings for the move_group node (discussed in section 2.4.1), represented by the yellow block in the architecture in Figure 10. When the robot is loaded, a control interface is instantiated. The MoveIt interface (blue block in Figure 10) is in charge of sending the commands to the move_group, which in turn initiates an action client that communicates to an action server running on the robot (purple block in Figure 10).




```
Robots:
- robot_name: "name"
  description: "description"
  reliability: 0.8
  urdf:
    package_name: " package_name "
    executable: "executable”

  manipulation:
    controller:
      package_name: "package_moveit_config"
      executable: "controller_robot.launch"
      args: "arg1:=false arg2:=true"
      planning_groups: ["planning_group_name"]
    driver:
      package_name: "package_moveit_config"
      executable: "driver_robot.launch"
      args: "robot_ip:=xxx.xxx.x.xxx"
```

(a) (b)

Figure 11. *General structure of robot_description YAML file (a) in a schematic form (b) in natural language*

The Robot Manager Interface (RMI), depicted as a red block in Figure 10, is the bridge to users or other applications to access the functionalities of TRM. Its main objective is to simplify the use of TRM via providing a C++ based API. Using ROS services for passing commands, the RMI sends the instructions to the robot manager, for example to initialize a robot (step "a" in Figure 10, or to command it (step "b" in Figure 10). Whenever an instruction is given via RMI, an according callback function is invoked within TRM which executes the requested routine.

Method	**Arguments**	**Description**
loadRobot	Robot_name	Initialize the robot as resource
planManipulation	Robot_name Planning group Target Pose	Compute a trajectory to a desired pose with a defined planning group
execute	Robot name	Execute a plan

Table 1. *Main set of functions provided on the Robot Manager Interface API*

The API of RMI, which allows accessing the functionality of TRM is listed in Table 1. When a command is sent, the robot manager identifies which robot a certain instruction is addressed to using the *robot_name* argument. The Robot Manager searches for the configuration of the robot to direct the instruction to the respective hardware.

4. Objective

The main objective of this thesis is to improve TeMoto Robot Manager to the level that it can be used for Multi-Robot System. Consequently, the navigation and gripper functionalities must be implemented besides MoveIt. Additionally, illustrate the new capabilities of TRM in a demo using the three features working together to accomplish a common task.

4.1. Functional Requirements

This section describes the requirements that the new functionalities of TRM should fulfill.

1) Expand the capabilities of TRM to fully support compound robots.
2) Driver and controller for manipulation, navigation and gripper features must be definable via Robot Description YAML file.
3) TRM creates a control interface per each feature stored in the Robot Description YAML file.
4) The manipulation feature is able to plan and execute a trajectory for manipulator robots using the move_group interface though moveit platform.
5) The navigation feature is implemented utilizing ROS Navigation.
6) A gripper feature allows opening and closing grippers.
7) The TRM interface API provides access to all the features of the TRM.

4.2. Non-functional Requirements

1) All the development must operate on ROS Kinetic under Ubuntu 16.04 operating system

5. Design and Implementation

The architecture of TRM proposed in this thesis is shown in Figure 12. Making use of the previous architecture of TeMoto Robot Manager discussed in chapter 3.1, the elements that provide the navigation and gripper functionalities were added (Region highlighted with blue dashed line).

Since navigation is one of the major fields in robotics, and grippers allow us to grasp and handle objects, this section describes the methods included in the Robot Manager Interface, the adjustments made in the robot description file and the elements that allow communication with the real hardware instantiated in the Robot handle, so that the developer can send commands to manipulators, mobile bases and grippers through TRM.

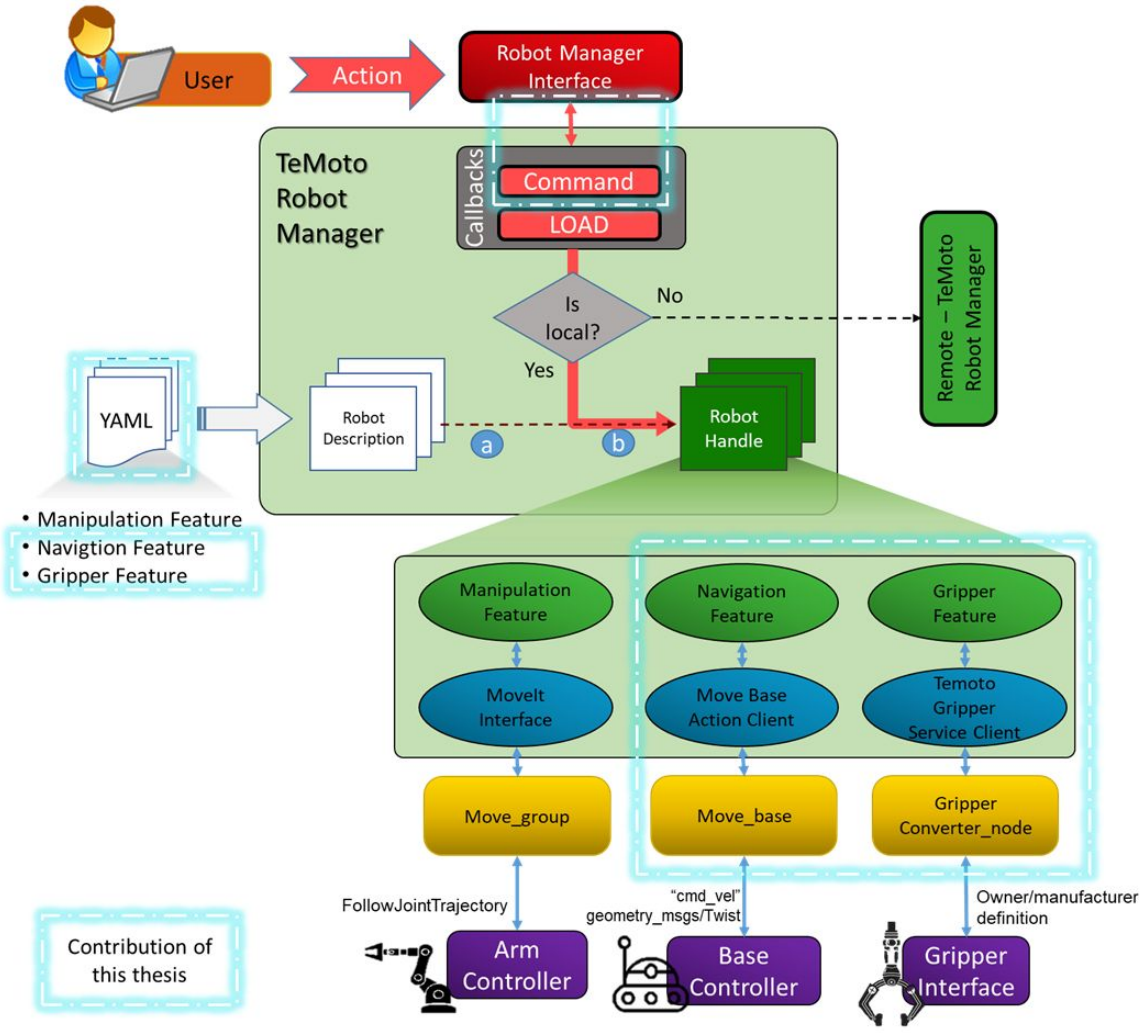


Figure 12. *Architecture of TeMoto Robot Manager with extended features for navigation and gripper.*

Similarly to the manipulation feature, the fields for the navigation and gripper features were included in the YAML file as depicted in Figure 13. In order to let TRM know the capabilities of a robot, they need to be listed in the robot description file, which starts with a robot_name key, to identify each robot within the system. As YAML is a format that relies on indentation for the structure, because it denotes nesting, the new features are set at the same level as the manipulation feature indicating that the feature listed belongs to that particular robot.

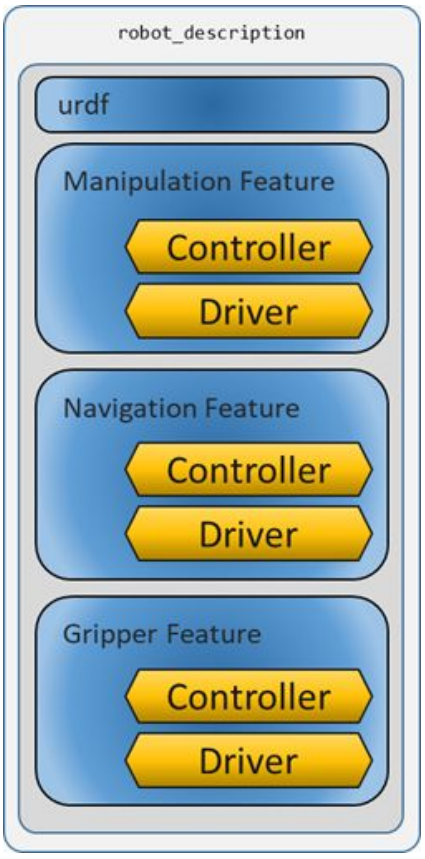


Figure 13. *Structure of a robot description YAML File*

Each feature includes subfields for the controller and driver (yellow block in Figure 13), that takes string values that represent the name of the packages, launch files for the executables and optional arguments to specify values that are passed to the specific launch file. That is the core information that provides TRM the ability to manage those resources.

If one or more features are defined in the robot description file, TRM is able to identify which feature it is referring to, and instantiate a corresponding control interface. For instance, if the robot to parse corresponds just to a mobile base, the navigation feature is the only description that needs to be defined. In the case of a compound robot for example, all of the features must be listed under the same robot as in Figure 13. However, in case of a multi-robot system, it is required to include a robot definition with different robot_name for each robot, even if the hardware is the same (in the case of homogeneous MRS).

5.1. Navigation feature

In order to include navigation functionalities to TRM, the existing platform ROS Navigation (discussed in Chapter is included 2.4.2) is implemented. This allows developers to control mobile bases through the RMI by defining a desired pose of the robot with respect to the map frame and sending it as goals.

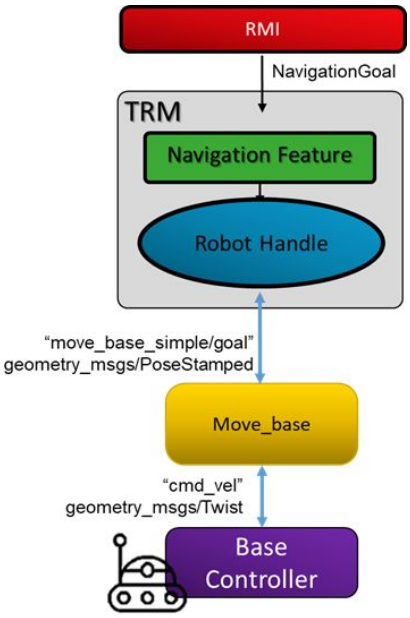


Figure 14. *Architecture of Navigation feature*

The navigation feature contains the driver and the controller definition in the robot description file as shown in Figure 15. They contain the executables and parameters that are in charge of setting up the move_base node (discussed in section 2.4.2). The driver allows the connection with the real hardware and provides the */odom* topic. Equally, the controller makes the link between the move_base node and the base controller through the topic */cmd_vel*, taking the goals and converting them into motor speed, depending on the configuration of the robot.

```
Robots:
- robot_name: "name"
  description: "description"
  reliability: 0.8
  urdf:
    package_name: " package_name "
    executable: "executable"
  navigation:
    controller:
      package_name: "package_name_navigation"
      executable: "controller_robot.launch"
      global_planner: "global_planner"
      local_planner: "local_planner "
    driver:
      package_name: "package_name"
      executable: "driver_robot.launch"
```

Figure 15. *Example of Navigation feature in a Robot Description YAML file*

When a load command is addressed to a robot with navigation feature, TRM executes those launch files and instantiates a control interface. As the move_base node makes use of ROS action protocol that accepts goals from action clients, the robot handler is defined as an action client to be able to send goal requests to the move_group server through a *SimpleActionClient* (server with a single goal policy).

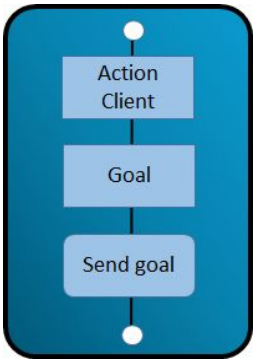


Figure 16. *Flow diagram to send a goal using an action client for the navigation implementation*

The Figure 16 illustrates the flow of the process in the robot handler to seng a goal to the movo_base node. Firstly, it constructs an action client that takes the server name to connect to (*robot_ns/move_base*). Then, a *goal* object is created with a *geometry_msgs/PoseStamped* message definition with a desired pose, meaning a representation in the space of position and orientation about where the robot should move to in the world. Then the robot handle sends the goal request to the move_base action server, and waits for its execution. A goal result is sent back upon completion of the goal.

In the RMI, a method was included that allows developers to send navigation goals to mobile bases. That function requires the robot_name, frame_id and goal pose. This frame_id specifies the reference frame for the location of the goal, that is to say, if it is defined as "*map*", the coordinates will be considered in the global reference frame or absolute position, but in case of being defined it as base_link, the coordinate would be with respect to robot base frame attached to the robot, so it is a relative position.

5.2. Gripper feature

The ISO 14539 standard about manipulating industrial robots defines a gripper as an end effector designed for seizing and holding objects, and it mentions different types of finger control, among which there are two-value control (i.e. open and close), position control, velocity control, force control or hybrid [58]. However, its protocol and message definition are left at the discretion of the manufacturers, so that each brand has its services and types of messages, and there is currently no unified platform or structure to control different grippers. This makes its direct integration into the TeMoto framework inconvenient, as it is impractical to add dependencies to an endless number of packages for each possible gripper.

However, it is necessary to include a general structure capable of handling different grippers. For this reason, it is proposed to include an intermediate node between TRM and the gripper hardware, which acts as a converter node or translator between different types of service definitions.

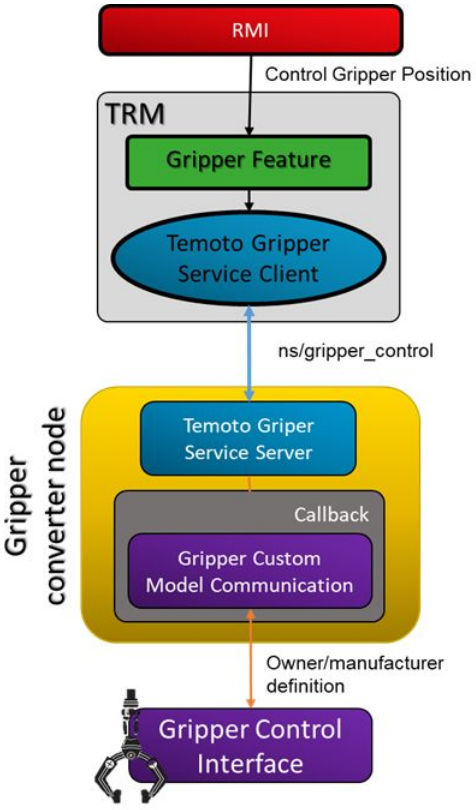


Figure 17 *Architecture of gripper converter node*

Just like the other features (manipulation and navigation), the driver and controller are defined in the robot description file as in Figure 18. The driver launches a file to bring up the real robot with the connection type and parameters to control the gripper. It initializes the services required by the robot for the finger operation. The proposed converter node is launched as a controller of the gripper feature. Firstly, this node is in charge of creating a temoto_gripper_control Server (demarcated in blue in Figure 17) to receive commands, secondly it must adjust the values receive from the TeMoto definition to the custom value for the gripper, and finally send the adjusted command to the gripper control interface.

```
Robots:
- robot_name: "name"
  description: "description"
  reliability: 0.8
  urdf:
    package_name: " package_name "
    executable: "executable"
  gripper:
    controller:
      package_name  "package_name_gripper"
      executable: "converter_node.launch"
    driver:
      package_name: "package_name"
      executable: "driver_gripper.launch"
```

Figure 18. *Example of Gripper feature in a Robot Description YAML file.*

In the RMI, a method was included that allows to send gripper position commands. This function requires the name of the robot to which this feature belongs, and a float value as value of control.

TRM acts as a Service Client (line 1 Figure 19), which calls the server running in the converter node. The request message is compound by the name of the robot to which the gripper belongs or is attached (line 3) and a *float32* as a control parameter (line 4).

```
1 - client=nh_.serviceClient<temoto_robot_manager::GripperControl>(ns/gripper_control);
2 - temoto_robot_manager::GripperControl gripper_srvc;
3 - gripper_srvc.request.gripper_name = robot_name;
4 - gripper_srvc.request.position = position;
5 - client_gripper_control_.call(gripper_srvc)
```

Figure 19 *TeMoto service client to pass gripper commands to the server running on a converter node.*

When the converter node receives a gripper control request, a callback function is executed in which the message can be adjusted to the data type, protocol, or specification to the certain gripper. For example, if the gripper needs a boolean value for only open and close commands, a control value of 0.0 that stands for fully closed and 100.0 for open can be used; or in case of position control, a transformation (e.g. scale) can be implemented and assigned to the gripper.

In addition, this controller node must communicate with the real hardware, which acts as Gripper Service Client (marked purple in Figure 17), relaying the previous request received with the adjusted values to the respective server or interface provided by the gripper.

5.3. Robot Manager Interface

As discussed in chapter 3.1, TRM uses a server-client architecture to pass commands. For the platform to be widely used and following the Goldilocks principle, provide enough functionality to be useful but not too much that the package is heavyweight and difficult to use from other software, TeMoto gives developers the functionalities and a variety of services they need.

For the manipulation feature, the robot can plan and execute trajectories to a desired point in the space. A getEndEffector function was included to ask for the current pose of the end effector. Additionally, some overload functions for planning a trajectory were included, to be able to command the robot either to a pose or a predefined named target pose, that contains all joint values under a name as a group state, which can be useful to adjust initial states or to ensure the pose of each link in the manipulator. In the case of the navigation feature, the user can send a pose goal establishing a coordinate and the orientation of the mobile base with the corresponding reference frame. For the gripper feature, a user can send a control gripper position making use of a float value as an argument.

The main functions that provide access to TRM are listed in the Table 2.

Feature	Method	Arguments	Description
All	loadRobot	Robot_name	Initialize the robot as a resource
Manipulation	planManipulation	Robot name Reference frame Pose	Compute a trajectory to a desired pose with a defined planning group
	executePlan	Robot name	Execute a planned trajectory for a defined robot
	getEndEffectPose	Robot name	Ask for the current pose of the end effector
Navigation	navigationGoal	Robot name Reference frame Pose	Allows to send a goal pose to the mobile base
Gripper	controlGripperPosition	Robot name Position	Send a value for opening and closing gripper command

Table 2. *Main set of functions provided on the Robot Manager Interface API for the manipulation, navigation and gripper feature.*

6. Discussion and Demonstration

In order to demonstrate the functionalities of the TRM, a following heterogeneous multi-robot scenario was implemented: The task is to deliver an object located on the table to the manipulator who is furthest from it. Since the mobile robot does not have the ability to take the object by itself, the first robotic arm must take the object and place it on top of the mobile robot, which navigates the environment towards the second robot arm and thus be able to take it. For its execution, there is a mobile robot, and two manipulators from different manufacturers each one with a gripper, who work together to move an object from one point to another. [3]

6.1. Control Architecture

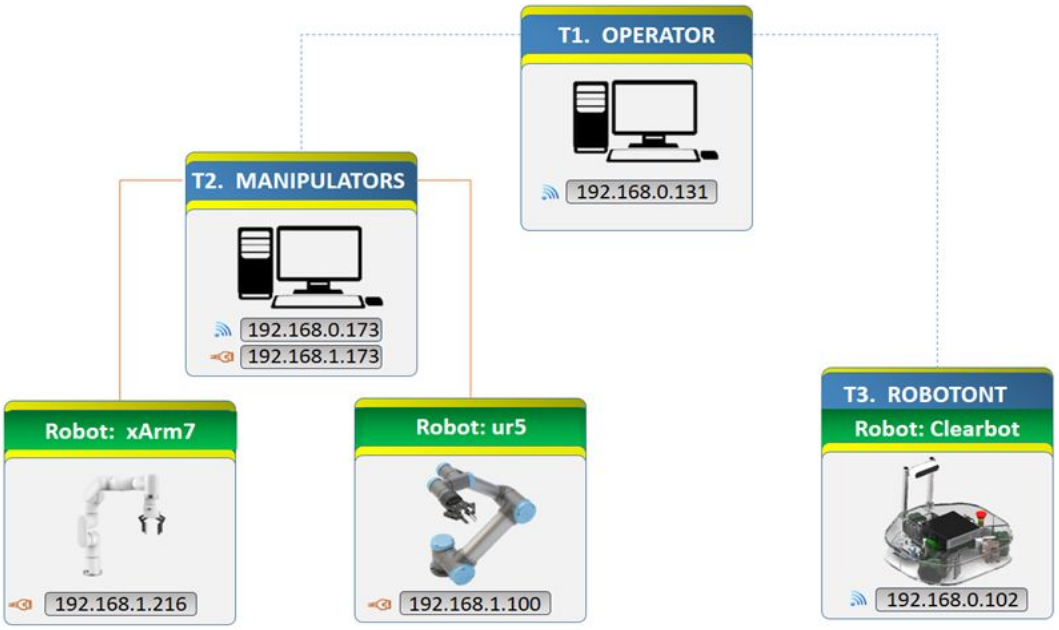


Figure 20. *TeMoto Architecture of the demo setup*

The Figure 20 shows a schematic of the architecture for the demo, to exemplify how it is distributed and how the instructions flow through the system. It contains three different instances of TeMoto running on separate stations. The first one T1-Operator is the instance arranged for the user. It does not contain any hardware or robot attached to it. T2-Manipulators contain the robot description of the xArm7 [59] and UR5 [59] robots, with its respective features (manipulation and gripper), and T3-Robotont is a TeMoto running on a Clearbot robot [61] (navigation). The TeMoto workstations communicate with each other through a wireless network, whereas the manipulators belong to a different network with a wired connection.

Since the intention of this demo is to show the ability of TeMoto to control a multi-robot system and understand multiple features in a single robot as a whole unit, rather than

[3] https://github.com/temoto-telerobotics-demos/robot_manager_mrs_demo

gathering information from the environment or set up dynamically coordinates in the space, all of the poses and goals implemented are hard coded. However, the trajectories and motion are planning on the go.

6.2. Hardware

The platforms used for the demo can be seen on Figure 21.

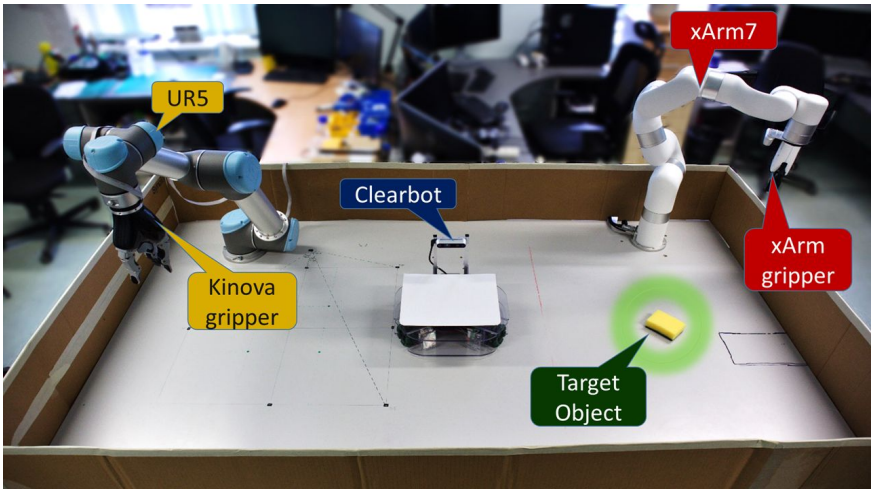


Figure 21. *Heterogeneous Multi-robot system to transport an object task in a demo scenario*

6.2.1. UFACTORY xArm7 manipulator

xArm7 is a robotic arm with a 7 Degree of freedom manufactured by Ufactory [59], made of aluminium and carbon fiber with a 3.5 kg payload, ±0.1mm repeatability and 700 mm of reaching area. Its communication protocol is through Ethernet TCP-IP. The xArm7 has a flexible collaborative gripper with customized fingertips suitable for a wide range of uses, designed specifically for the xArm robot. It is controlled through the IO port in the tool head.

```
Robots:
- robot_name: " xarm7_robot "
  description: " xarm7_robot "
  reliability: 0.8
  urdf:
    package_name: " xarm7_description "
    executable: " urdf/xarm7_with_gripper.xacro "

  manipulation:
    controller:
      package_name: " xarm7_wgrip_moveit "
      executable: " controller_temoto.launch "
      args: " fake_execution:=false show_rviz:=true debug:=false "
      planning_groups: ["xarm7"]
    driver:
      package_name: " xarm7_wgrip_moveit "
      executable: " driver_temoto.launch "
      args: " robot_ip:=192.168.1.216"

  gripper:
    controller:
      package_name  " temoto_gripper_converter "
      executable: " xarm_gripper_temoto_controller.launch "
    driver:
      package_name: " temoto_gripper_converter "
      executable: " xarm_gripper_temoto_driver.launch "
```

Figure 22. *xArm7 Robot description*

The robot description file implemented for the xArm robot is represented in Figure 22. It contains the definition for a robot named “xarm7_robot” and includes the manipulation and gripper features.

Making use of the ROS packages for xArm series from UFACTORY [62] a new package was created, including the table to represent the real environment where the robot is going to operate, and to prevent any kind of collision when MoveIt plans a trajectory. For the gripper functionality, two nodes were created. One for the driver that utilizes services to enable the gripper and configure the grasp speed, and one node for the controller that adjust the control command (0 - 100) to the proper range of the open distance between 0 to 850.

6.2.2. Universal Robots UR5

The UR5 is a lightweight, highly flexible, 6-DOF industrial robot arm produced by Danish company Universal Robots [60, p. 5], that allows to automate repetitive tasks with payloads of up to 5 kg in a 850 mm reaching area, ideal to optimize low-weight collaborative processes, such as picking, placing, and testing.

```
Robots:
- robot_name: " ur5_robot "
  description: " ur5_robot "
  reliability: 0.8
  urdf:
    package_name: " ur_description_robots "
    executable: " urdf/ur5_with_base.xacro "

  manipulation:
    controller:
      package_name: " ur5_v2_moveit_config "
      executable: " ur5_moveit_planning_execution.launch "
      planning_groups: ["manipulator"]
    driver:
      package_name: " ur_robot_driver "
      executable: " ur5_modified_bringup.launch "
      args: " robot_ip:=192.168.1.100 "

  gripper:
    controller:
      package_name  " temoto_gripper_converter "
      executable: " kinova_temoto_controller.launch "
    driver:
      package_name: " kinova_bringup "
      executable: " kinova_robotType:=m1n6s300 "
```

Figure 23. *UR5 and KG-3 compound robot description*

The UR5 uses a software version 3.12 with a control box CB3.1. Unlike the UFACTORY xArm7, the UR5 does not have native gripper, so a Kinova KG-3 gripper was mounted as the end-effector. As with xArm robot, a new ROS package was created using the MoveIt Setup Assistant, that includes the table and the gripper that was conditioned for this demo.

6.2.3. Kinova Gripper

A Kinova KG-3 gripper was attached to the end effector coupling of the UR5, to provide grasping and gripping functionalities that conform to objects of varying shapes and sizes. This gripper is used with a standalone controller box to power and control the actuators through a USB 2.0 port interface, attached to the T2-Manipulators TeMoto workstation.

Notice that the definition of the kinova gripper is under the robot "ur5_robot" in the robot description file, Figure 23, due it is mechanically coupled to this robotic arm. That is why TeMoto sees both the manipulator and the gripper as one robot / compound robot: "ur5_robot". In order to command the gripper, the *robot_name* argument required in the method on the RMI corresponds to "ur5_robot".

6.2.4. Clearbot Mobile Robot

It is an omnidirectional mobile robot with ROS support packages developed by the Institute of Technology at University of Tartu [61]. This platform has an intel mini pc and nucleo development board for processing data, able to run an instance of TeMoto. It is equipped with RealSense D435 camera that uses stereo vision to calculate depth that allows gatter information of the environment as well as contributes to the localization of the robot; each gearmotor has integrated a quadrature encoder that provides the odometry.

```
Robots:
- robot_name: " clearbot "
  description: " clearbot "
  reliability: 0.8
  urdf:
    package_name: " robotont_description "
    executable: " xacro/main.urdf.xacro "
  navigation:
    controller:
      package_name: " navigation_goal "
      executable: " controller_temoto.launch "
      global_planner: " navfn/NavfnROS "
      local_planner: " base_local_planner/TrajectoryPlannerROS "
    driver:
      package_name: " robotont_driver "
      executable: " driver_temoto.launch "
```

Figure 24. *Clearbot robot description*

The robot description file for Clearbot platform contains just the navigation feature as it is the only capability for this robot as can be seen on Figure 24. The driver and controller are in charge of communicating with the hardware, subscribes to the command velocity topicand publishes the odometry.

Since the Clearbot robot has to navigate in the space designated for the test, a map was provided using the map_server. This contains the information of the table borders and the footprint of the manipulators as they are fixed objects as shown in Figure 25. The reference

frame is located at the middle of the map. It is important to have reliable data from the camera and the odometry, otherwise the localization might fail.

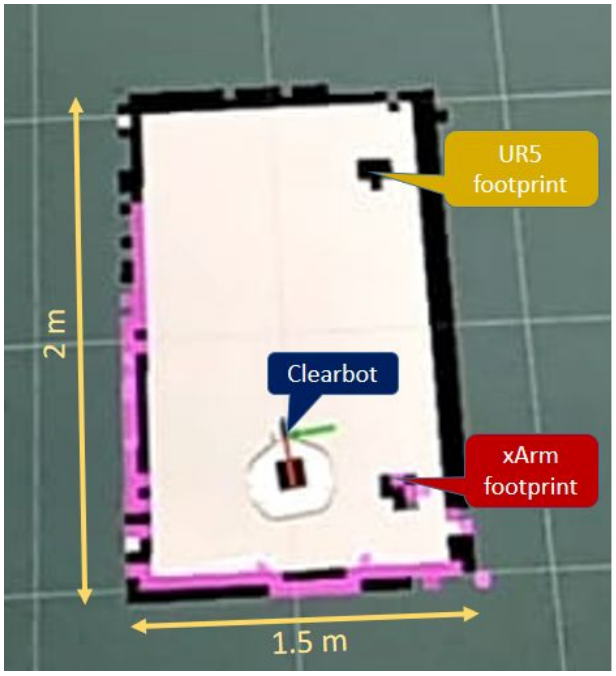


Figure 25. *Clearbot navigating on the map to the goal_1. Black edges correspond to global costmap, and pink represent the local costmap. Image taken from the rviz environment.*

6.3. Execution

This demo aims to exemplify the use of TRM and the new capabilities, in a heterogeneous multi robot system, to pick up an item located on the table, using a robot manipulator and a griper. The navigation feature is tested with a mobile robot in charge of moving into the environment towards the robot arm, which places the object on top of it, and then the mobile robot transports it to the second manipulator, which takes the object with its gripper. The entire routine is settled up in an action, thus the operator just needs to send a command to trigger the entire task.

There are three instances of TeMoto labeled as T1,T2 and T3 running in separate workstations as shown in Figure 26. Once the instances are started, the system is in charge of finding the robot description files (shown in pink in the schematic Figure 26) and remaining the valid configurations. Since there are several TeMoto platforms running, they share information to notify that a specific resource is available.

We must give a command to TeMoto in order to have the robots do a complex task. An initial instruction is required even if they do their tasks autonomously. In this case the TeMoto action is addressed to T1. From the T1-operator workstation, the TeMoto action gives all of the set of instruction for the pick and place operation, starting by initializing each robot, and although they do not belong locally in this station, the system knows these resources are hosted remotely and can be used, so the request is sent to the corresponding instance, which is in charge of launching the driver and controller. To initialize a robot, TRM uses the name defined as *robot_name* in the robot description YAML file,

Once the robots are initialized (Figure 27.1), the xArm is commanded to face an object, and using the gripper, lift it off the table (Figure 27.2). Using a navigation goal, the Clearbot robot is commanded to move near the xArm (Figure 27.3) which puts the object on top of Clearbot (Figure 27.4), and then a second navigation goal is sent to the Clearbot, this time to carry the object and navigate towards the UR5 (Figure 27.5). Once the mobile base robot reaches this pose goal, the UR5 plans a trajectory to get closer to the object (Figure 27.6) and using the kinova gripper takes it off the Clearbot robot (Figure 27.7), which concludes the action (Figure 27.8).

In the case of xArm gripper, the opening and closing position is normally controlled with values that range between 0 and 850 (fully close - fully open respectively). For Kinova m1n6s300 gripper the maximum value is 6400 for each finger.

Making use of the Robot Manager Interface, a value from 0 to 100 is used representing opening percentage, meaning that the user does not have to worry about the exact value for each gripper, and now is able to control all grippers the same way. When this value reaches the converter node, in the callback function it is adjusted according to the maximum value that this hardware accepts.

Every time a command is addressed to the xarm7_robot or ur5_robot, the requests are directed to the T2-manipulators RM, which communicates with the external hardware, but if the instruction given correspond to a goal pose for the mobile base, the instruction is redirected to the T3-Robotont RM.

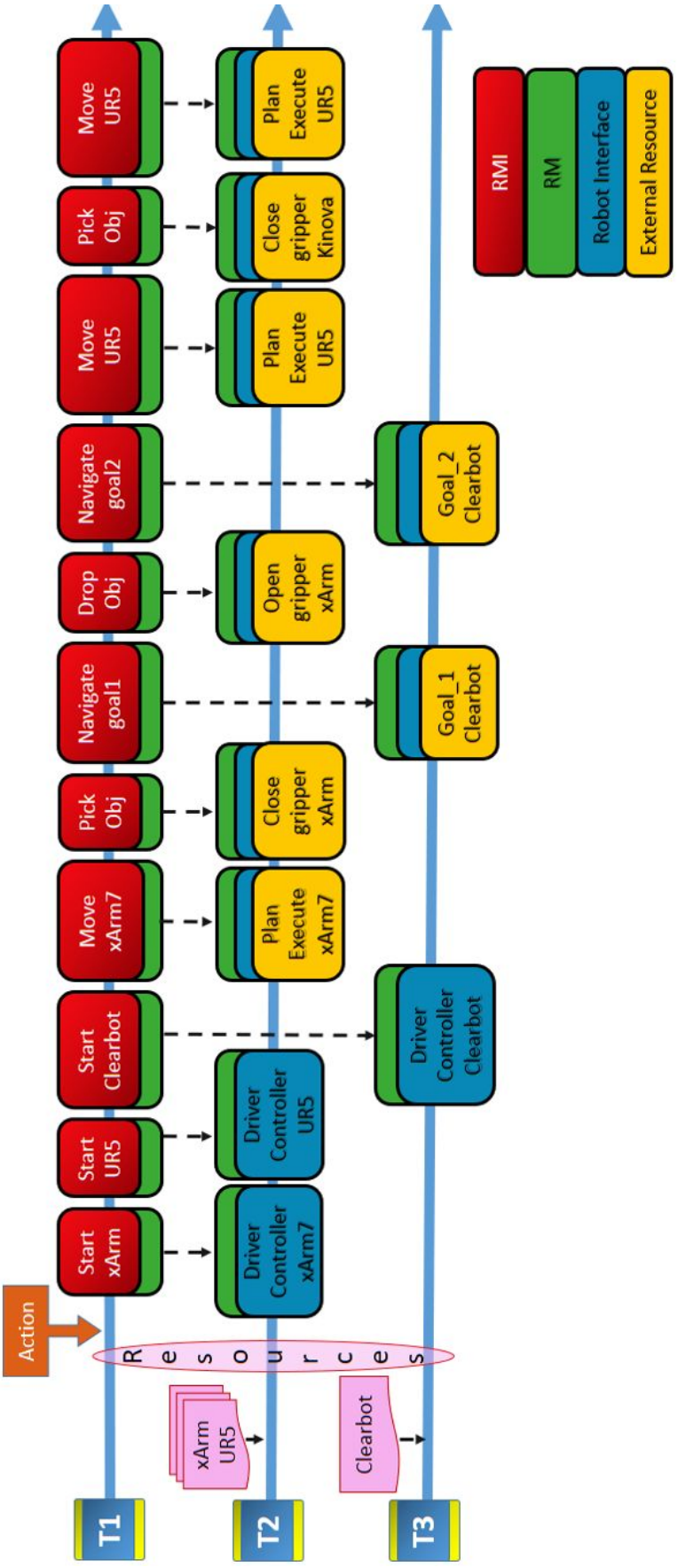


Figure 26. *Block diagram representing the sequence instructions flow and interaction between different TeMoto workstations.*

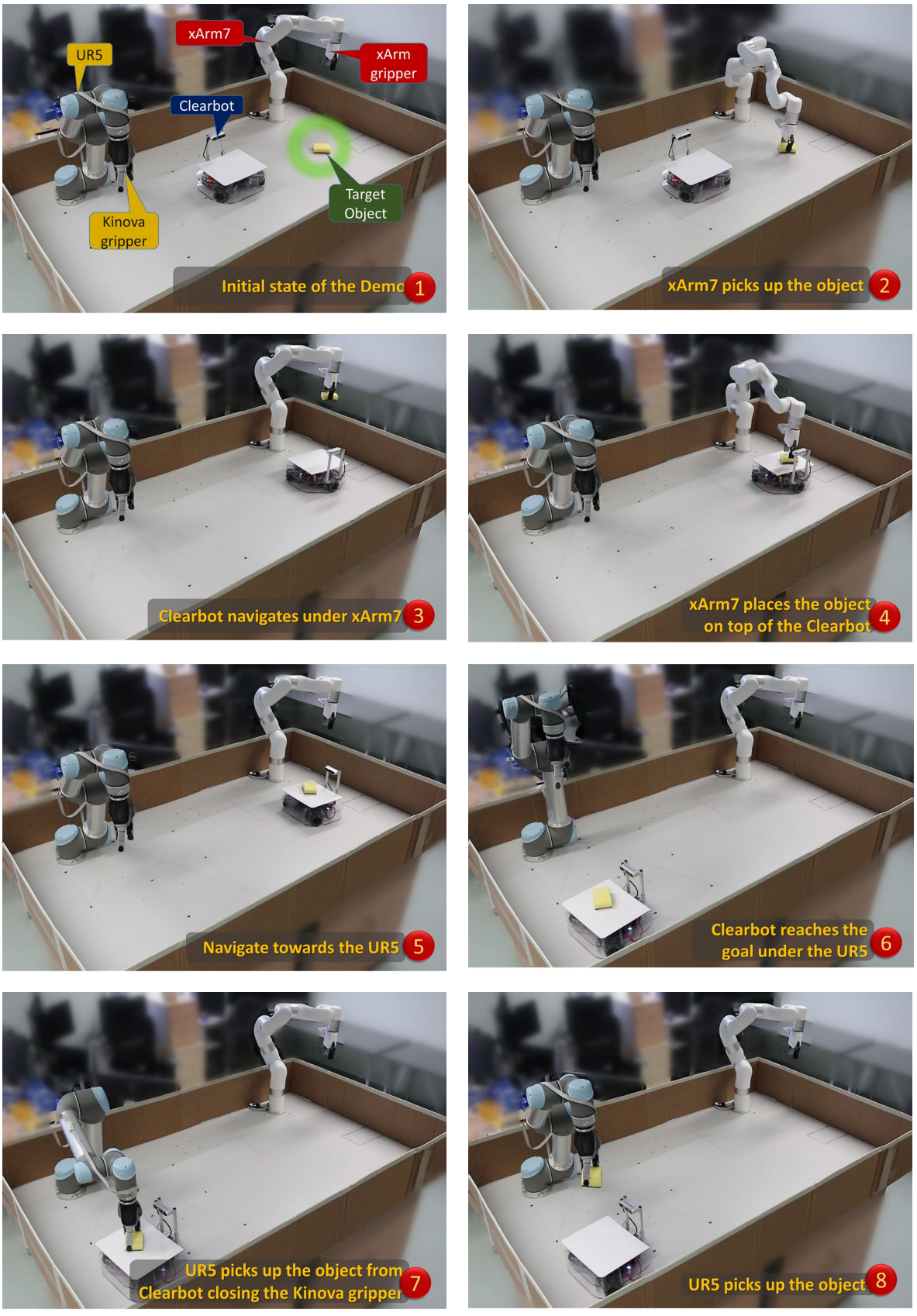


Figure 27. *Implementation of TRM to control a Heterogeneous MRS in a collaborative transportation task, using manipulation, navigation and gripper features.*

6.4. Demo results

Although the action contains all hardcoded values, it was possible to demonstrate that TRM is capable of handling different hardware interfaces within the same action, controlling manipulation, navigation and gripper features in a heterogeneous multi-robot system.

Using the manipulation feature, it was possible to plan and execute trajectories to desired poses in the space for different robotic arms. By adding the arrangement of objects surrounding the manipulator to the moveit_config, such as the case of the floor and walls, helps to avoid possible collision. The Clearbot robot was able to navigate to the desired points, with goals sent from the Robot Manager Interface defined with respect to the map frame, that is, absolute positions.

The converter node worked for both kinova and xArm gripper, converting from TeMoto gripper services to the respective message type. It was a good strategy to send values in percentage instead of the exact value required by the hardware, since it is easier to imagine its operation in a range from 0 to 100% than in an arbitrary number, leaving the converter node to adjust it to the corresponding value.

It could have had a more interactive system if this implementation were combined with other managers such as context and component manager (TeMoto managers), to make use of sensors to obtain the exact position of the object to be transported, and use that information as input to plan a path to that point in the space. However, for the development of this thesis, the objective was to demonstrate that TeMoto has the ability to command different types of robots, from a central station for the operator, rather than the interaction between managers.

6.5. Limitations and Future Work

Each robot is required to have its own robot description file, even if they are completely the same. In case of having a swarm for example, the same configuration must be replicated N times with a different robot_name to be able to identify them within the system. It would be convenient in cases like this, to have a team_size or number of homogeneous platform parameter fields so that the system automatically instantiates each platform.

Because the architecture used to pass commands is based on a service-client or action-client structure that use callback functions, each instruction waits for the task to be executed and the system responds once it has finished. This implies that within a TeMoto action, only a particular feature can be addressed at a time. In case of needing to operate robots with same feature simultaneously, for example a dual-arm setup, one option would be to use separate TeMoto actions for each robot, or it might be necessary to include a logic in the Robot Manager that generates threads to execute the instruction in parallel.

In case of multi-manipulator robot setups, the manipulators are not aware of each other unless they are described within the same moveit_config. This is a problem if we want to ,e.g., mount another xArm7 to the table and start controlling it via moveit without modifying the moveit_config of the pre-existing arm, these robots will be two individual robots without inter-robot collision avoidance. If the arm is included in the previous configuration, they should be commanded via different planning groups.

7. Conclusions

The main core of TeMoto architecture is to facilitate software developments by allowing programmers spending their time on adjusting the parameters of its application to fit in the TeMoto structure, rather than dealing with low-level details. As a result of this thesis, some functionalities for navigation and gripper control purposes were included on the TeMoto framework and meets the described prerequisites.

Incorporating the ROS navigation and gripper functionalities to TeMoto, the range of robots that can be operated with this framework is greatly expanded, and being able to define their multiple characteristics in a single robot description, as in the case of compound robots, allows them to be operated as a unit rather than separate parts.

The implementation was demonstrated to be operational by testing it on a heterogeneous multi-robot system, with a mobile base and two manipulators (xArm7 - UR5), each one with a gripper (xArm gripper - Kinova 3 fingers), in a collaborative task to transport an object from one place to another which involves planning trajectories, navigate in an environment and grasping operations.

The converter node proposed for the integration of the gripper acting as a translator between different message definitions, helps to communicate TeMoto with the hardware. It worked fine for both xArm and Kinova grippers, even though they manage their own services.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Robert Valner and Karl Kruusamäe, for their patient guidance, enthusiastic encouragement, and useful critiques of this research work. I truly appreciate that they allow me to be part of this project, and they were always there to help me at times when I had questions.

I would like to extend my gratitude to the Faculty of Science and Technology and its members for allowing me to cultivate my passion for robotics and technology.

I want to thank the support and the great love of my family, that from a distance, they always kept me going and this work would not have been possible without their input.

Finally, I wish to thank my friends and everyone else who contributed to this process.

References

[1] G. Stollnberger *et al.*, “Robotic systems in health care,” in *2014 7th International Conference on Human System Interactions (HSI)*, Jun. 2014, pp. 276–281, doi: 10.1109/HSI.2014.6860489.

[2] A. Okamura, M. Matarić, and H. Christensen, “Medical and Health-Care Robotics,” *Robot. Autom. Mag. IEEE*, vol. 17, pp. 26–37, Oct. 2010, doi: 10.1109/MRA.2010.937861.

[3] F. Sherwani, M. M. Asad, and B. S. K. K. Ibrahim, “Collaborative Robots and Industrial Revolution 4.0 (IR 4.0),” in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, Mar. 2020, pp. 1–5, doi: 10.1109/ICETST49965.2020.9080724.

[4] K. Nagatani *et al.*, “Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots,” *J. Field Robot.*, vol. 30, no. 1, pp. 44–63, 2013, doi: 10.1002/rob.21439.

[5] M. Ben-Ari and F. Mondada, “Robots and Their Applications,” in *Elements of Robotics*, M. Ben-Ari and F. Mondada, Eds. Cham: Springer International Publishing, 2018, pp. 1–20.

[6] A. Gautam and S. Mohan, “A review of research in multi-robot systems,” in *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, Aug. 2012, pp. 1–5, doi: 10.1109/ICIInfS.2012.6304778.

[7] E. Freund and H. Hoyer, “Pathfinding in multi-robot systems: Solution and applications,” in *1986 IEEE International Conference on Robotics and Automation Proceedings*, Apr. 1986, vol. 3, pp. 103–111, doi: 10.1109/ROBOT.1986.1087653.

[8] A. Farinelli, L. Iocchi, and D. Nardi, “Multirobot systems: a classification focused on coordination,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 34, no. 5, pp. 2015–2028, Oct. 2004, doi: 10.1109/TSMCB.2004.832155.

[9] J. J. Roldán, E. Peña-Tapia, A. Martín-Barrio, M. A. Olivares-Méndez, J. Del Cerro, and A. Barrientos, “Multi-Robot Interfaces and Operator Situational Awareness: Study of the Impact of Immersion and Prediction,” *Sensors*, vol. 17, no. 8, Jul. 2017, doi: 10.3390/s17081720.

[10] F. J. Mendiburu, M. R. A. Morais, and A. M. N. Lima, “Behavior coordination in multi-robot systems,” in *2016 IEEE International Conference on Automatica (ICA-ACCA)*, Oct. 2016, pp. 1–7, doi: 10.1109/ICA-ACCA.2016.7778506.

[11] Z. Yan, N. Jouandeau, and A. A. Cherif, “A Survey and Analysis of Multi-Robot Coordination,” *Int. J. Adv. Robot. Syst.*, vol. 10, no. 12, p. 399, Dec. 2013, doi: 10.5772/57313.

[12] C. A. Czarnecki, “Multi-Robot Systems: A Perspective,” *IFAC Proc. Vol.*, vol. 27, no. 4, pp. 201–205, Jun. 1994, doi: 10.1016/S1474-6670(17)46023-X.

[13] Y. Tan and Z. Zheng, “Research Advance in Swarm Robotics,” *Def. Technol.*, vol. 9, no. 1, pp. 18–39, Mar. 2013, doi: 10.1016/j.dt.2013.03.001.

[14] G. Baele *et al.*, “Open-ended on-board Evolutionary Robotics for robot swarms,” in *2009 IEEE Congress on Evolutionary Computation*, Trondheim, Norway, May 2009, pp. 1123–1130, doi: 10.1109/CEC.2009.4983072.

[15] K. Chen *et al.*, “Compound manipulation mode for improving task-ability of multi-arm multi-flipper crawler robot,” in *2017 IEEE/SICE International Symposium on System Integration (SII)*, Dec. 2017, pp. 463–468, doi: 10.1109/SII.2017.8279256.

[16] M. Selvaggio, S. Grazioso, G. Notomista, and F. Chen, “Towards a self-collision

aware teleoperation framework for compound robots," in *2017 IEEE World Haptics Conference (WHC)*, Jun. 2017, pp. 460–465, doi: 10.1109/WHC.2017.7989945.
[17] "Compound Robot," *Heavy Duty AGV Manufacturer*. https://heavydutyagv.com/case-study/compound-robot (accessed Apr. 24, 2020).
[18] O. Khatib, "Mobile Manipulators: Expanding the Frontiers of Robot Applications," in *Field and Service Robotics*, London, 1998, pp. 6–11, doi: 10.1007/978-1-4471-1273-0_2.
[19] C. Lopez-Franco, J. Hernández-Barragán, A. Alanis, N. Arana-Daniel, and M. López-Franco, "Inverse kinematics of mobile manipulators based on differential evolution," *Int. J. Adv. Robot. Syst.*, vol. 15, p. 172988141775273, Jan. 2018, doi: 10.1177/1729881417752738.
[20] R. Ali, D. A. Aldair, and A. Almousawi, "Implementation and Design of Fuzzy Supervisory Controller for Mobile Robot Manipulator," *Basrah J. Eng. Sci.*, vol. 16, Jan. 2017.
[21] K. Nagatani, T. Hirayama, A. Gofuku, and Y. Tanaka, "Motion planning for mobile manipulator with keeping manipulability," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2002, vol. 2, pp. 1663–1668 vol.2, doi: 10.1109/IRDS.2002.1043994.
[22] "The benefits of a dual-arm robotic system," *Wevolver*, 1578971761248. https://www.wevolver.com/article/the.benefits.of.a.dualarm.robotic.system/ (accessed Apr. 07, 2020).
[23] uclabiomechatronics, "New teleoperated bimanual mobile manipulator has arrived," *UCLA Biomechatronics Lab*, Nov. 13, 2018. https://uclabiomechatronics.wordpress.com/2018/11/13/new-bimanual-mobile-manip/ (accessed Apr. 07, 2020).
[24] C.A.CZARNECKI, "Multi-Robot Systems: A Perspective | Elsevier Enhanced Reader." https://reader.elsevier.com/reader/sd/pii/S147466701746023X?token=B7E54DB7A96296426E79A5FDD49C61A01C4CB6A76005B34F4921BFBC96BB0530E79FA8D5C7B2DDA460F1F1D47F4445D5 (accessed Mar. 16, 2020).
[25] J. Lopez-Perez, U. Hernandez-Belmonte, J.-P. Ramirez-Paredes, M. Contreras-Cruz, and V. Ayala, "Distributed Multirobot Exploration Based on Scene Partitioning and Frontier Selection," *Math. Probl. Eng.*, vol. 2018, pp. 1–17, Jun. 2018, doi: 10.1155/2018/2373642.
[26] K. Wurm, C. Dornhege, P. Eyerich, C. Stachniss, B. Nebel, and W. Burgard, "Coordinated exploration with marsupial teams of robots using temporal symbolic planning," presented at the IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings, Nov. 2010, pp. 5014–5019, doi: 10.1109/IROS.2010.5649820.
[27] D. W. Haldane, P. Fankhauser, R. Siegwart, and R. S. Fearing, "Detection of slippery terrain with a heterogeneous team of legged robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4576–4581, doi: 10.1109/ICRA.2014.6907527.
[28] A. Pennisi, F. Previtali, F. Ficarola, D. D. Bloisi, L. Iocchi, and A. Vitaletti, "Distributed Sensor Network for Multi-robot Surveillance," *Procedia Comput. Sci.*, vol. 32, pp. 1095–1100, Jan. 2014, doi: 10.1016/j.procs.2014.05.538.
[29] J.-H. Park, S.-C. Choi, I.-Y. Ahn, and J. Kim, "Multiple UAVs-based Surveillance and Reconnaissance System Utilizing IoT Platform," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan. 2019, pp. 1–3, doi:

10.23919/ELINFOCOM.2019.8706406.
[30] J. Gregory *et al.*, “Application of Multi-Robot Systems to Disaster-Relief Scenarios with Limited Communication,” in *FSR*, 2015, doi: 10.1007/978-3-319-27702-8_42.
[31] J. Alonso-Mora, S. Baker, and D. Rus, “Multi-robot formation control and object transport in dynamic environments via constrained optimization:,” *Int. J. Robot. Res.*, Aug. 2017, doi: 10.1177/0278364917719333.
[32] A. Petitti, A. Franchi, D. Di Paola, and A. Rizzo, “Decentralized Motion Control for Cooperative Manipulation with a Team of Networked Mobile Manipulators,” 2016. .
[33] R. Darmanin and M. Bugeja, “A Review on Multi-Robot Systems categorised by Application Domain,” Jul. 2017, doi: 10.1109/MED.2017.7984200.
[34] M. M. Veloso and D. Nardi, “Special Issue on Multirobot Systems,” *Proc. IEEE*, vol. 94, no. 7, pp. 1253–1253, Jul. 2006, doi: 10.1109/JPROC.2006.877080.
[35] H. Kim, B. H. Kim, C.-N. Chu, J. H. Lee, K. H. Ha, and S. Hong, “Development of multi-scale cooperative robot,” *2013 Int. Conf. ICT Converg. ICTC*, 2013, doi: 10.1109/ICTC.2013.6675479.
[36] C. Anderson *et al.*, “Mobile manipulation: A challenge in integration,” Apr. 2008, doi: 10.1117/12.777329.
[37] E. Mwendi, “Software Frameworks, Architectural and Design Patterns,” *J. Softw. Eng. Appl.*, vol. 07, pp. 670–678, Jan. 2014, doi: 10.4236/jsea.2014.78061.
[38] Yuan-hsin Kuo and B. A. MacDonald, “A Distributed Real-time Software Framework for Robotic Applications,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 1964–1969, doi: 10.1109/ROBOT.2005.1570401.
[39] A. Couture-Beil, R. T. Vaughan, and G. Mori, “Selecting and Commanding Individual Robots in a Multi-Robot System,” in *2010 Canadian Conference on Computer and Robot Vision*, May 2010, pp. 159–166, doi: 10.1109/CRV.2010.28.
[40] Sergi Hernandez Juan, Josep M. Mirats Tur, “Generic modular framework for robotic arm applications - Ecet,” *International Conference on Computer Systems and Technologies*. https://www.yumpu.com/en/document/read/50600554/generic-modular-framework-for-robotic-arm-applications-ecet (accessed Mar. 15, 2020).
[41] J. López, D. Pérez, and E. Zalama, “A framework for building mobile single and multi-robot applications,” *Robot. Auton. Syst.*, vol. 59, no. 3, pp. 151–162, Mar. 2011, doi: 10.1016/j.robot.2011.01.004.
[42] “ROS.org | Powering the world’s robots.” https://www.ros.org/ (accessed Apr. 22, 2020).
[43] “Topics - ROS Wiki.” http://wiki.ros.org/Topics (accessed May 17, 2020).
[44] “Services - ROS Wiki.” http://wiki.ros.org/Services (accessed May 17, 2020).
[45] “actionlib.” http://library.isr.ist.utl.pt/docs/roswiki/actionlib.html (accessed Apr. 27, 2020).
[46] “ROS.org | About ROS.” https://www.ros.org/about-ros/ (accessed Apr. 15, 2020).
[47] “Robots | MoveIt.” https://moveit.ros.org/robots/ (accessed Apr. 22, 2020).
[48] “navigation - ROS Wiki.” http://wiki.ros.org/navigation (accessed May 09, 2020).
[49] “What is moveit_ros? All about MoveIt! ROS,” *The Construct*, Jan. 25, 2018. https://www.theconstructsim.com/ros-moveit/ (accessed Apr. 22, 2020).
[50] “MoveIt! Setup Assistant — moveit_tutorials Kinetic documentation.” http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html (accessed Apr. 22, 2020).

[51] “Concepts | MoveIt.” https://moveit.ros.org/documentation/concepts/ (accessed Mar. 31, 2020).
[52] L. Joseph, *Mastering ROS for Robotics Programming*. Packt Publishing Ltd, 2015.
[53] “navigation/Tutorials/RobotSetup - ROS Wiki.” http://wiki.ros.org/navigation/Tutorials/RobotSetup (accessed Apr. 23, 2020).
[54] “(PDF) ROS Navigation: Concepts and Tutorial,” *ResearchGate*. https://www.researchgate.net/publication/302986850_ROS_Navigation_Concepts_and_Tutorial (accessed Apr. 19, 2020).
[55] R. Valner, *INTUITIVE “HUMAN-ON-THE-LOOP” INTERFACE FOR TELE-OPERATING REMOTE MOBILE MANIPULATOR ROBOTS*. Zenodo, 2018.
[56] R. Valner, K. Kruusamäe, M. W. Pryor, and A. Zelenak, “TeMoto 2.0: Source Agnostic Command-to-Task Architecture Enabling Increased Autonomy in Remote Systems.” Mar. 18, 2018.
[57] “temoto-telerobotics.github.io by temoto-telerobotics.” https://temoto-telerobotics.github.io/temoto-telerobotics.github.io/site/ (accessed Mar. 17, 2020).
[58] “ISO 14539:2000(en), Manipulating industrial robots — Object handling with grasp-type grippers — Vocabulary and presentation of characteristics.” https://www.iso.org/obp/ui/#iso:std:iso:14539:ed-1:v1:en (accessed Apr. 28, 2020).
[59] “xArm,” *store.ufactory.cc*. https://store.ufactory.cc/pages/xarm (accessed Apr. 29, 2020).
[60] “UR5 collaborative robot arm | Flexible and lightweight robot arm.” https://www.universal-robots.com/products/ur5-robot/ (accessed Apr. 29, 2020).
[61] “robotont.” http://robotont.ut.ee/main?lang=en (accessed May 04, 2020).
[62] *xArm-Developer/xarm_ros*. UFACTORY, 2020.

Non-exclusive licence to reproduce thesis and make thesis public

I, Fabian Ernesto Parra Gil

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

"IMPLEMENTATION OF ROBOT MANAGER SUBSYSTEM FOR TEMOTO SOFTWARE FRAMEWORK"

supervised by Karl Kruusamäe and Robert Valner.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other person's intellectual property rights or rights arising from the personal data protection legislation.

Fabián Ernesto Parra Gil
20.05.2020