Università degli Studi di Verona

DIPARTIMENTO DI DI INFORMATICA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Tesi di Laurea Magistrale

Multi-Robot Task Allocation for logistic applications

Candidato:
Davide Zorzi
Matricola vr414572

Relatore:
Alessandro Farinelli

Anno Accademico 2017-2018

Contents

Abstract

Robotics technology has recently matured sufficiently to deploy autonomous robotic systems for daily use in several applications: from disaster response to environmental monitoring and logistics. In this project we present and evaluate the main difference of central allocator task coordinator. In this application we address off-line coordination, by casting the Multi-Robot logistics problem as a task assignment problem and proposing three solution techniques: Single robot Single task (SR:ST), which is a baseline greedy approach, Greedy Set Partition Strategy - Single robot Multiple task (GSP1:N) and Set Partition Strategy - Single robot Multiple task (SPS1:N), which are based on composing task to minimize the task completion time.
We evaluate the performance of our system in a realistic simulation enviroment (build with ROS and stage). In particular, in the simulated enviroment we compare our task assignment approaches with the baseline greedy approch. Results show that fully exploiting the capacities of robots is key to optimze system performance and that the GSP1:N achieves similar performance to SPS1:N while being able to scale up to many tasks.

Keywords: Multi-Robot Task Allocation, logistic applications, Multi-Robot systems, coordination, task assignment

Chapter 1

Introduction

The use of MRS is a key aspects to obtain effective solutions in several real world applications such as for example [1].
More particularly, this thesis addresses the cooperation of a team of mobile robots in logistic missions. The main aspects studied herein are strategies for effective logistic performance, agent's coordination, scalability and applicability in real-life situations. In our logistic application the robot team execute specific tasks modeled as exogenous events that are characterized by a pick-up location and a delivery location each.
This introductory chapter presents the context of the research in order to clarify the motivation and significance of the problem. In addition, some guidelines about MRS in general and, more specifically, agents in logistic missions are herein introduced to lay the groundwork to approach the problem in hands. Finally, we provide an overview of the thesis content.

1.1 Context and Motivation

In recent years, robotics has been one of the scientific fields with the most substantial advances. Among the different areas of robotics, mobile robotics gained in the last decades significant attention roboticists (i.e., researchers on robotics) around the world. In particular, issues such as autonomous nav-

[1] See for example the Kiva system used by Amazon in their warehouse https://www.wired.com/2009/01/retailrobots/.

igation, path planning, self-localization, coordination of robots, cooperative dynamics, mapping, exploration and coverage have become popular topics and have exploited the progress of artificial intelligence, control theory, real-time systems, sensors' development, electronics, communication systems and systems integration [1].
Nowadays, we see different types of robots operating in different environments as on land, underwater, in the air, suspended on wires, climbing and so on. This evident growth is extremely motivating for the development and contribution of new developments by the community.
Security applications are a fundamental task with unquestionable impact on society. Combining this fact with the technological evolution observed in the last decades, it becomes clear that robot assistance can be a valuable resource by taking advantage of robots' ability to cary out tedious or dangerous tasks for a very long time.

The principal scope of the MRS in logistic application is reduce the costs of the production phase and increase the productivity that the robots guarantee in these applications.

In particular, Multi-Robot allocator task for logistic applications has high utility and is considered as a key area where the use of robots can have a dramatic impact on productivity in last decade, especially in terms of strategies for coordinating teams of robots. A key point in for effective application of MRS in logistic application is to coordiante actions of the robot platforms [2]. However, many of the studies in the literature present unrealistic simplifications, strong limitations or questionable applicability as illustrated 1.3. Therefore, there is an eminent potential to explore in this context.
Task allocation for logistic applications problem is very challenging in the context of MRS , because agents must navigate autonomously, coordinate their actions and acquire information about the surrounding enviroment, possibly with communication constraints and should be able to achieve good performance on the number of robots in the team and the enviroment's dimension. Clearly, cooperation among robots is one of the most decisive issues in this context, since robots most efficiently work together in order to improve the performance of the system as a whole. All of these features lead to an excellent case study in mobile robotics and conclusions drawn from such

studies may support the development of future approaches not only in the logistic domain but also in MRS in general.

1.2 Multi-Robot Systems

During the last two decades, researchers in the field of mobile robotics have begun to investigate problems that involve multiple robot rather than using single robot, and research in MRS has witnessed notorious progress an never before. In many applications, an autonomous mobile robot equipped with different sensors may adequately complete a given assignment. However, in several situations. it proves to be more expensive, less efficient and less robust than using a MRS . In some cases, due to the need of combining different tasks and the dynamics of the environment, it is only viable to achieve the mission with a multiple distributed autonomous robotic system. Some characteristics of MRS include distributed control, autonomy, communicative agents and greater fault-tolerance. A single robot may be vulnerable to hostile environments or attackers, for example, in military actions. In such scenarios, agents would greatly benefit from the assistance of nearby agents during emergencies, failures or malfunctions.

Logistic applications can significantly benefit from the use of several robots, however (as we discussed above) the effective use of MRS in logistic applications requires a significant effort to delevop an effective and efficient coordinated solution.

Most missions are solved much quicker it robots operate in parallel. Increasing robustness and reliability of the solution is also feasible in MRS by introducing redundancies in the capabilities across robot team member and graceful performance degradation, remaining functional if some of the agents fail.

One of the main difficulties when approaching these systems is to coordinate many robots to perform a complex, global task in an efficient manner, maximizing group performance under a wide range of conditions, with the flexibility to take advantage of the resources available, embrace the requirements and constraints imposed and resolve issues like action selection, coherence, conflict resolution and communication. This cannot be done by just increasing the number of robots assigned to a task. A coordination mechanism must exist to establish relationships between agents so that they can accomplish the mission effectively.

1.3 Multi-Robot Systems for logistic applications

Logistic apllication an infrastructure with multiple robots is no different than other Multi-Robot assignments, in the sense that it incorporates all the previously mentioned characteristics of MRS . To understand this problem, it is important to first introduce the definition of logistic application.

Definition 1. *The industrial logistics is the process of planning, organization and control of all the activities of handling and storage of goods, which, starting from the suppliers and reaching up to the end user, guarantee an adequate level of service to the customer consistent with the costs to it associated* [2]*.*

A traditional warehouse employee typically spends most of his or her time walking around the warehouse to gather all of the items for an order. In manual warehouse, a picker might walk between seven and fifteen miles per shift [3]. To save labor by reducing the time spent walking, a lot company adopt MRS to improving the productivity and have an efficient service for customers.

Many real-world applications of MRS require agents to operate in known common environments. Our logistic scenario is a map of laboratory for industry 4.0 (ICE lab) with a modern production line, extended with equipment for augmented reality and digital production, and connected with the university computational platform plus new hubs.

Below is reported the Figure 1.1 of the Industrial Computer Engineering Laboratory used in our logistc scenario.

[2]See Basic concepts of Industrial Logistics and Logistics System https://elearning.unito.it/sme/pluginfile.php/36284/mod_folder/content/0/LOGISTICA

[3]http://www.businessinsider.com/working-conditions-at-an-amazon-warehouse-2013-2?IR=T

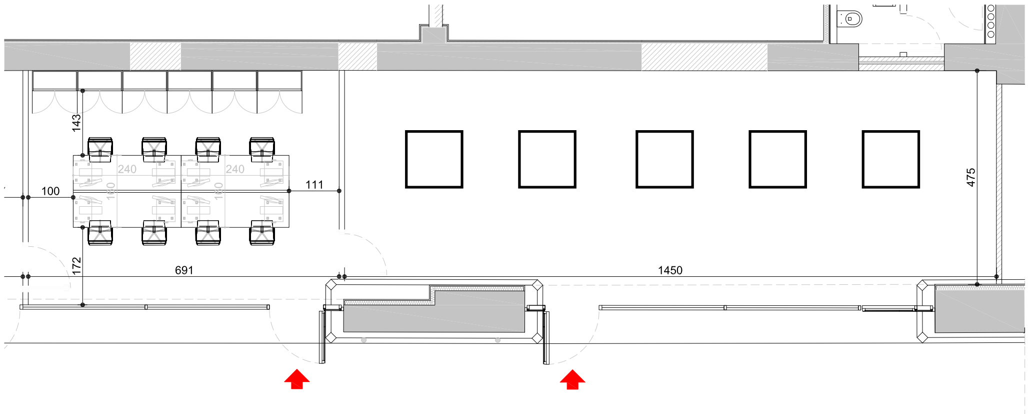


Figure 1.1: This is our logistic scenario of Industial Computer Engineering Laboratory

The map used in the simulator is scaled to have measurements in real enviroment. In this map we have build the logistic system to perform pick-up and delivery tasks.

An pick-up and delivery task is characterized by a pickup location and a delovery location. An agent has to move from its current location via the pickup location of the task to delivery location of the task. When the agent reaches the pickup location, it starts to execute the task and the task is removed from the task set.

Wheeled mobile robots mostly operate over a differential drive mechanism. It consists of two motors attached wirh wheels on a similar axis. Both motor drives are independent of each other's movement. The common access of both motors is known as their center of curvature. The agent cannot move in the direction along axis. Since both motors can move independently, their velocities are also different and we can easily vary the trajectory of movement by varying velocities. The velocities of both drivers must be synchronized in order to move the agent in the desired path. The robot constructs a two-dimensional geometric representation of its environment using the laser scanner. It utilizes a combination of this geometric data and odometry information supplied through the wheel encoders to determine its current location. For more information about navigation and localization see section 4.2. We focus our study to task-allocation problems where one wants to minimize the team cost subject to the constraint that each task must be executed by a given number of cooperative agents simultaneously. For MRS coordination in logistic application there are a lot technique, in automated warehouse, where agents are constantly engaged with new task, one agent has to be assigned to each delivery task avoiding collisions with other agents. In [3] and [10] paper has recently received a lot of attention. They present two decoupled MAPD algorithms, Token Passing (TP) and Token Passing with Task Swaps (TPTS). Where agents operate in known common enviroments modeled such as grids, for each task in a given 2-dimensional 4-neighbor grid with blocked and unblocked cells. The multi-agent pathfinding problem is to compute collision-free paths for multiple agents, in a post-processing step to adapt its paths to continuous forward movements with given traslational velocities and point turns with given rotation velocities. They take kinematic constraints for instance the robot can turn around only 90 degree at time and can perform only one task at time. In our techniques not consider kinematic constraints and our robots can perform more than one task at time, if the capacity of robots are enough. Then we can allocate more than one task and execute its in one shot. The agents are constantly engaged with new tasks and have to navigate between locations where the tasks need to be executed. In particular, the MRS for logistic applications, the set of robots must complete a stream attend to stream of incoming pickup-and-delivery tasks.

1.4 Thesis contribution

The contribution of this thesis:

- extension of package ROS, implementing an external coordinator for generating tasks and algorithms for composing tasks.
- proposing three technique:
  1. baseline greedy approach SR:ST .
  2. set partition algorithm SPS1:N consider all possible subtasks of the task set.
  3. greedy set partition algorithm GSP1:N proposing an approximation solution with less complexity but efficient subset of task.
- all experiments are executed in a real scenario, precisely on ICE Lab [4].

We can see in section 5 that the baseline approach is the worst algorithm takes more time and takes a more onerous path. Instead the other two algorithms are quite similar. The set partition method is better based on the time and distance traveled but the greedy method, despite having a lower complexity, obtains excellent results.

The limitation of the set partition approach (SPS1:N) lies in the size of the task set having an exponential complexity would take too long to compute all possible subsets of task. For this limitation in the experiments we will use a maximum of 9 tasks for this strategy.

In our approaches we focus on task assignment in base the capacity of the robots for minimize the time traveling and the path distance for every composed task.

[4]See on site the Computer Engineering for Industry 4.0 http://www.di.univr.it/

1.5 Thesis outline

Having answered the questions "What problem is addressed in this thesis?" and "Why are we studying this problem?", it is fundamental to answer an additional question, which is: "How is the problem going to be solved?". This question requires a more in-depth answer, that is detailed throughout the rest of thesis.

Initially, an analysis of relevant literature concerning related work to the MRS is conducted in chapter 2. This allows to formulate the problem.

In chapter 3 we detail our reference scenario for MRS coordination and well as the formalization of the problem we addressed and the solutions we propose. Where we introduce the baseline approach for our experiments to then focus on two techniques which are based on composing task to minimize the task completion time.

Later on, in chapter 4 a prelimenary framework to solve the MRS in logistic application.

Finally, the last chapter sums up the work and provides final conclusions and future directions of research.

Chapter 2

Background and Related Works

In this section, we detail the main issues for MRS coordination in industrial domains, then we provide a detailed discussion on coordination approaches, highlighting challenges and main solution techniques.

2.1 Multi-Robot System for Logistics Applications

In this thesis we focus on industrial scenarios where robots have a high degree of autonomy and operate in a dynamic environment.

In this article [14] they presented the Kiva warehouse-management system create a paradigm for pick-pack-and-ship warehouse that improves worker productivity [1] . The Kiva system uses movable storage shelves that can be lifted by small, autonomous robots. By bringing the product to the worker, productivity is increased by a factor of two or more, while simultaneously improving accountability and flexibility. The key innovation in the Kiva system is the application of inexpensive robots capable of lifting and carrying three-foot-square shelving units, called *inventory pods*. The robots, called *drive units*, transport the inventory pods from storage locations to stations where workers can pick items off the shelves and put them into shipping cartons. Throughout the day, the picker stays in her station while a continuous

[1] https://www.amazonrobotics.com/#/

stream of robots presents pick-faces. By moving the inventory to the worker, rather than the other way around, we typically see worker productivity at least double. The Kiva drive units operate in a controlled, known enviroment, greatly simplifying the design problem and making the solution pratical. Another distinguishing attribute of MRS is the extent to which agents are cooperative (in the sense that they must coordinate activities to achieve a system goal) or are self-interested and have independent, often conflicting, objectives. Although the overall system is cooperative, the Kiva robots are essentially independent. Warehouses and distribution centers play a critical role in the flow of goods from manufacturers to consumers. They serve as giant routing centers in which pallets of products from different manufacturers are split, and the items are redirected into outgoing containers. The drive units are small enough to fit under the inventory pod and are outfitted with a mechanical lifting mechanism that allows them to lift pods off the ground. The pods consist of a stack of trays, each of which is subdivided into bins. A variety of tray sizes and bin sizes create the mixture of storage locations for the profile of products the warehouse stores. A Kiva installation is arranged on a grid with storage zones in the middle and inventory stations spread around the perimeter. The drive units are used to move the inventory pods with the correct bins from their storage locations to the inventory stations where a pick worker removes the desired products from the desired bin. Note that the pod has four faces, and the drive unit may need to rotate the pod in order to present the correct face. When a picker is done with a pod, the drive unit stores it in an empty storage location. For compute the path planning the environment is defined such as a grind costitutes a two-dimensional graph of paths that may be given weights at design time. The drive units use a standard implementation of A^* to plan paths to storage locations and inventory stations. The drive unit agents also maintain a list of highlevel goals and are responsible for prioritizing the goals and accomplishing them as efficiently as possible. Then the drive unit agent decides which station to visit first and in what sequence to show the faces to minimize travel time.

The most recent papers in logistic scenario [10] and [3], are definited the Multi-Agent Pickup and Delivery problem (MAPD) where a large number of agents attend to stream of incoming pickup-and-delivery tasks. They using a Token Passing (TP) approach for implement MAPD algorithm that is efficient and effective. The MAPD algorithm takes kinematic constraints of real robot into account direclty during planning, computes continuous agent movements with given velocities that work on non-holonomic robot rather than discrete agent movements with uniform velocity. TP assumes, like many Multi-Agent pathfinding algorithms, discrete agent movements in the main compass directions with uniform velocity but can use a post-processing step to adapt its paths to continuous forward movements with given translational velocities and point turns with given rotational velocities. Unfortunately, the resulting paths might then not be effective since planning is oblivious to this transformation. TP needs to repeatdly plan time-minimal paths for agents that avoid collisions with the paths of the other agents. The important implication for this paper is that agents repeatdly plan paths for themselves, considering the other agents as dynamic obstacles that follow their paths and with which collisions need to be avoided. The agents use space-time A^* for this single-agent path planning. A set of endpoints is any subset of cells that contains at least all start cells of agents and all pickup and delivery cells of tasks. The pickup and delivery cells are called task endpoints. The other endpoints are called non-task endpoints. TP operates as follows for a given set of endpoints: It uses a token (a synchronized block of shared memory) that stores the task set and the current paths, one for each agent. The system repeatedly updates the task set in the token to contain all unassigned tasks in the system and then sends the token to some agent that is currently not following a path. The agent with the token considers all tasks in the task set whose pickup and delivery cells are different from the end cells of all paths in the token.

Overview of TP works:

1. If such tasks exist, then the agent assigns itself that task among these tasks whose pickup cell it can arrive at the earliest, removes the task from the task set, computes two time-minimal paths in the token, one that moves the agent from its current cell to the pickup cell of the task and then one that moves the agent from the pickup cell to the delivery cell of the task, concatenates the two paths into one path, and stores the resulting path.

2. If no such tasks exist and the agent is not in the delivery cell of any task in the task set, then it stores the empty path in the token (to wait at its current cell).

3. Otherwise, the agent computes and stores a time-minimal path in the token that moves the agent from its current cell to some endpoint that is different from both the delivery cells of all tasks in the task set and from the end cells of all paths in the token.

Each path the agent computes has two properties: (1) It avoids collisions with all other paths in the token; (2) No other paths in the token use its end cell after its end time. Finally, the agent releases the token, follows its path, and waits at the end cell of the path. Tey demonstrate the benefit of their approach for automated warehouse. Otherwise, they take kinematic constraints for instance the robot can turn around only 90 degree at time and can perform only one task at time [2]. This limitation in our system does not preside.

In article [2] they focus on approaches that are based on algorithms widely used to solve graphical models and constraint optimization problems, such as the max-sum algorithm. They analyse the coordination problem faced by a set of robots operating in a warehouse logistic application. In this context robots must trasport items from loading to unloading bays so to complete packages to be delivered to customers. Robots must cooperate to maximizes the number of packages completed in the unit of time. To this end crucial component is to avoid interferences when moving in the enviroment. They show how such problem can be formalised as a Distributed Constrained Optimization problem (DCOP)and they provide a solution based on the binary max-sum algorithm. In more detail, in this paper [2], they provide DCOP model for the task assignment problem faced by robots involved in logistics operations in a warehouse. Among the various solution approaches for DCOPs they advocate the use of heuristic algorithms, and specifically the max-sum, an iterative message passing approach that has been shown to provide solutions of high quality for systems operating in real-time and with

[2]Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery https://www.youtube.com/watch?v=RTJvJYJVxJk&t=30s

limited computation and communication resources. When the decicions of one robot affect only a small subset of team, because the message update step, a key operation of max-sum, has a computation completely that is exponential in the number of robots that can perform the same task. This exponential element can be a significant limitation for large scale, real-time systems. To combat this, they show that for specifc types of constraints and using binary variables, such exponential element can be reduced to a polynomial. Hence, they can use the max-sum approach for large-scale systems that must operate with real-time constraints.

In this work, we consider a similar setting where a set of robots are involved in trasportations tasks for logistics. However, we focus on the specific problem of task assignment.

2.2 Coordination in Multi-Robot Systems

Coordination for MRS has been investigated from several diverse perspectives and nowadays, there is a wide range of techniques that can be used to orchestrate the actions and movements of robots operationg in the same enviroment. Specifically, the ability to effectively coordinate the actions of a MRS is a key requirement in several applications domains that range from disaster response to environmental monitoring, militaty operations, manufacturing and logistics. In all such domains, coordination has been addressed using various frameworks and techniques and there are several survery papers dedicated to categorize such different approaches and identifying most prominent issues when developing MRS .

In this paper [4] present and evaluate new ROS package for coordinated multi-robot exploration. The packages allow completely distributed control and do not rely on (but allow) central controllers. Their integration including application layer protocols allows out of the box installation and execution. The communication package enables reliable ad hoc communication allowing to exchange local maps between robots which are merged to a global map (for more detail see section 4.2.4). Exploration uses the global map to spatially spread robots and decrease exploration time. The intention of the implementation is to offer basic functionality for coordinated multi-robot systems and to enable other research groups to experimentally work on multi-robot systems. They use the terms "loacl" and "global" to distinguish contexts of a single robot and the complete multi-robot system. A local map is the map created by each individual robot, while the global map includes local maps of all robots. Communication enables the exchange of data. Lastly, coordinated exploration utilizes communication and the global map to organize the MRS by assigning frontiers to robots.

They contribution in to present ROS package that enable Multi-Robot exploration implementing the aforementioned required components:

- ad hoc communication between robots.
- construction of global maps from local maps.
- exploration of unknown environments.

In this work [12] they focus on MRS coordination, presented a survery of recent work in the area by specifically examining the forms of cooperation and coordination realized in the MRS . Robotics systems may range from simple sensors, acquiring and processing data, to complex human-like machines, able to interact with the enviroment in fairly complex way. Moreover, it is not easy to give a definition of the level of autonomy that is required for a robot in order to be considered an entity acting in the enviroment, as opposed to a simple machine that provides services to the operator. From an engineering standpoint, the MRS can improve the effectiveness of a robotic system either from the viewpoint of the performance in accomplishing certain tasks, or in the robustness and reliability of the system, which can be increased by modularization. In fact, MRS are useful not only when the robots can accomplishing different functions, but also when they have the same capabilities. Even when a single robot can achieve the given task, the possibility of deploying a team of robots can improve the performance of the overall system. Another significant development of MRS is technological improvements both in the hardware and in the associated software are two of the key reasons beyond the growing interest in MRS . The increased availability of complex sensor devices and robotic platforms in the research laboratories favored their development and customization, resulting in robots equipped with reliable and effective hardware that improves their basic capabilities. In addiction, the software techniques developed for the robotic applications take advantage of the hardware improvements and provide complex and reliable solutions for the basic tasks that a robot should be able to perform, while acting in real world environments: localization, path planning, object trasportation, object recognition and tracking, etc. In addiction, the work in this area can be classified from several points of view. Their main motivation is the study and evaluation of the ability to take advantage of coordination to improve system performance. Therefore, the classification we propose is focused on the coordination aspects and thus inspired by the relationships with the field of multi-agent systems. They proposed the taxonomy for classifying the works on MRS in characterized by two groups of dimensions: *coordinatin dimensions* and *system dimensions*. For a suitable classification of the works it is important to clearly define the dimensions that are used:

- *cooperation level:* is the ability of the system to cooperate in order to accomplish a specific task. A cooperative system is composed of "robots that operate together to perform some global task".

- *knowledge level:* is concerned with the knowledge that each robot in the team has about its team mates.

- *coordinatin level:* is the mechanisms used for cooperation in which the actions performed by each robotic agents "in such a way that the whole ends up being a coherent and high-performance operaiton". The underlying feature is the *coordiantion protcol*, that is defined as a set of rules that the robots must follow in order to interact with each other in the enviroment.

- *organization level:* is the way the decicion system is realized within the MRS . Introduces a distinction in the forms of coordination, distinguishing centralized approaches from distributed ones. In particular, a centralized system has a sgent (*leader*) that is in charge of organizing the work of other agents;the leader is involved in the decision process for the whole team, while the other members can act only according to the directions of the leader. Instead, a distributed system is composed ao agents which are completely autonomous in the decision process with respect to each other; in this case of systems a leader does not exist.

In this paper they have addressed the recent developments in the field of MRS , focusing on those approaches that are targeted to specific applications and motivated by engineering considerations. Specifically, they have presented a taxonomy with the aim of highlighting the coordination aspects of the recent proposals in the literature: we have defined a set of coordination dimensions for the classification of the approaches to team coordination, together with a set of system dimensions that account for the design choices that are more relevant to the team organization.

In out system using a central coordinator that give the actions to execute at all robots, which know the static environment and moving in their given task independently.

Another important article [9] focus they work on coordinatin for complex tasks. Complex task are tasks that can be solved in many possible way. Their work is currently limited to complex tasks that can be decomposed into multiple subtasks related by Boolen logic operators. They generalizing task descriptions into task trees, which allows tasks to be traded in a market setting at variable level of abstraction. The task allocation problem addresses the issue of finding task-to-robot assignments the optimize global cost objectives. They address the general problem of allocating complex tasks to a team of autonomous robots. Complex tasks are tasks that require high-level decision-making or planning, and may have many potential solution strategies. Complex tasks are usually identified with problems involving multiple interacting components. These interactions can come from relationships between subtasks such as Boolean logic operations or precedence constraints. Additionally, if there are multiple robots, complex tasks may require reasoning about interactions between the robots executing them. Specifically, they look at tasks hierarchically related by *AND* and *OR* logical operators. The main contribution of this paper are to identify the complex task allocation problem can be more efficiently solved by not decoupling the solution into separate allocation and decomposition phases, and to propose a solution concept that unifies these two stages. The approach is not optimal, but produces highly efficient solutions in unknown and dynamic domains using distributed local knowledge and decentralized planning to continually improve upon global costs.

In contrast to this article in our system we have simple tasks to compose in more complex tasks. The interconnection between the various tasks is given by a heuristic that normalizes the cost of the route based on the number of objects transported in a composed task.

Given our focus on logistic scenarios, here we restrict our attention to coordination approaches based on optimization and specifically on task assignment as this the most common framework for our reference application domain.

Chapter 3

Multi-Robot task allocation for pick-up and delivery

In this section we detail our reference scenario for MRS coordination and as well as the formalization of the problem we addressed and the solutions we propose.

3.1 Problem description

Our reference scenario is based on a warehouse that stores items of various types. Such items must be composed together to satisfy orders that arrive based on customers' demand. The items of various types are stored in particular sections of the building (*loading bay*) and must be trasported to a set of *unloading bays* where such items are then packed together by human operators. The set of items to be trasported and where they should go depends on the orders.

In our domain a set of robots is responsible for transporting items from the loading bay to the unloading bays and the system goal is to maximize the throughput of the orders, i.e., to maximize the number of orders completed in the unit of time. Now, robots involved in transportation tasks move around the warehouse and are likely to interfere when they move in close proximity, and this can become a major source of inefficiency (e.g., robots must slow down and they might even collide causing serious delays in the system).

Hence, a crucial aspect to maintain highly efficient and safe operations is to minimize the possible spatial interferences between robots.

The main goal of our system is composing simple tasks to create a subset of complex tasks to improve the productivity and minimize the robot's time travel and the paths distance.

To manage the merging of the task we can use a range of different techniques. In the setion 3.3 there are the strategies used to create subsets of tasks. After the composing step, we have varieties of tasks subsets. At the end to validate which is the best subset we propose an heuristic. This heuristic function normalize the overall path of the subset of tasks based on the number of items that each robot can carry. Furthermore the purpose is to maximize the number of items in the same journey.

3.2 Problem fomalization

In this section we formalize the MRS coordination problem described above as a task allocation problem where the robots must be allocated to transportation tasks.

In our formalization we have a finite set of tasks. A robot can execute a task if it is available else the robot will go at start position. For how the system is built, a robot to request a task must arrive at the previous vertex of the loading bay.

In more detail, our model considers a set of items of different types $E = \{e_1, ..., e_N\}$, stored in a specific loading bay (L). The warehouse must serve a set of orders $O = \{o_1, ..., o_M\}$. Orders are processed in one or more than one of the unloading bays (U_i). Each order is defined by a vector of demand for each item type (the number of required items to close the order). Hence, $o_j =< d_{1,j}, ..., d_{N,j} >$, where $d_{i,j}$ is the demand for order j of items of type i. When an order is finished a new one arrives, until the set of task is finished. The orders induce a set of $N \times M$ trasportation tasks $T = t_{i,j}$, with $t_{i,j} =< d_{i,j}, P_{i,j} >$, where $t_{i,j}$ defines the task of transporting $d_{i,j}$ items of type i for order o_j (hence to unloading bay U_j). Each task has a destination bay for centralized coordination the $t_{i,j}$ has a set of edges $P_{i,j}$ which respects the strategy used. We have a set of robot $R = \{r_1, ..., r_K\}$ that can execute transportation tasks, where each robot has a defined load capacity for each item type $C_k =< c_{1,k}, ..., c_{N,k} >$, hence $c_{i,k}$ is the load capacity of robot k for items of type i.

We consider in our logistic scenario, homogenous robots, which have the same radius and the same capacity. Because often in the logistic environments robots are all equal.

Since the main of this thesis is composing subsets of tasks we formalize what is a subset of tasks and what we mean for merging paths of a subset of tasks and which is the heuristic funcion used for differentiate the subests of tasks or simple tasks.

Specifically given a set of tasks $\mathcal{T}$ it define intrinsically a set of orders O. One order perform a subset of $\mathcal{T}$. A subest of tasks S, denoted by $S \subseteq \mathcal{T}$.

Where $S = \{T_1, \cdots, T_k\}$ for each element of the subset we combine their paths P to form a single path π. For π define a path that perform the travel order such as a vector of vertices $\pi = \{v_1, \cdots, v_i\}$.

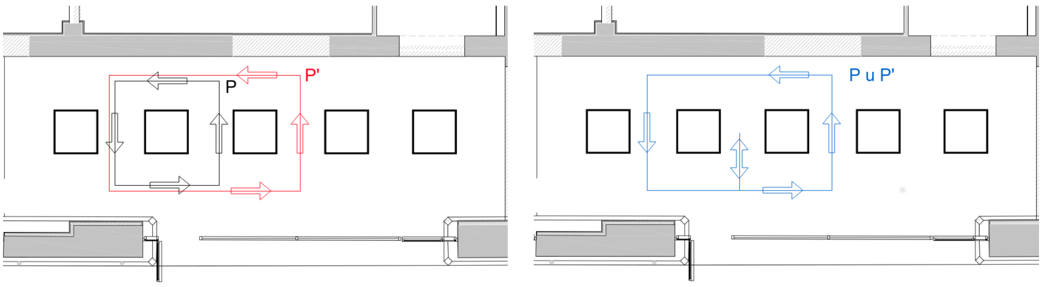


Figure 3.1: Comparison of the union of two paths P and P' to form a single path $\pi = P \cup P'$.

The function $p(\cdot)$ is used to join the paths, this funcion return the vector of vertices π.

For calculate the cost of the paths we used a function $f(\cdot)$ that return the path distance.

Because we maximize the total demand (d_S) we calculate the sum the single demand for each element in the subset. More formally:

$$d_S = demand(T_1) + \cdots + demand(T_k)$$

where $demand(\cdot)$ return the single demand of the specific task.

Another function used in our system is $dst(\cdot)$ that return the euclidean distances between loading and unloading bays.

Finally we define our heuristic function $v(\cdot)$ which can be defined for any task T or subsets S:

$$v(T) = \frac{f(P)}{demand(T)}$$

This heuristic function is used for normalize the cost of path (path distance) on the demand (cost of the item/items) of the order.

For compute the best partition of tasks the heuristic is based on the concept of loss L, which can be defined for any pair of subset S_i, S_j as:

$$L(S_i, S_j) = v(\{S_i \cup S_j\}) - v(S_i) - v(S_j)$$

where $v(S)$ is value of the characteristic function $v(\cdot)$ for subset S. In other worlds, this loss funcion captures the value of synergies between subsets and is computed in constant time.

The function v for a subset S is defined as:

$$v(S) = \frac{f(\pi)}{d_S}$$

where $f(\pi)$ is the path distance between the loading bay L and the unloading bays U_j and the distance to return to the initial vertex and the d_s is the total demand of the tasks.

We want minimize the cost of the loss, that can be defined as:

$$L(S_i, S_j) < 0$$

if the loss L is less than 0 then we allocate the pair and delete the element which form the subset.

3.3 Proposed solution

In this section we propose the approaches with centralized coordination. In a warehouse the connection between robots is efficient and has no big limitations.

1. The first strategy, mentioned above, is the SR:ST which consider only one task allocated for one robot.
2. The second strategy SPS1:N using an optimal approach to compose tasks. Precisely resolve the set partition problem of set tasks $\mathcal{T}$.
3. The last strategy (GSP1:N) extends the first algorithm, the main concept of this strategy is composing the tasks in a single travel with greedy coalition formation approach.

In section 6 as a future development we want to explore a distributed method because in this thesis we only focused in centralized method.

3.3.1 Single robot : Single task (SR:ST)

This method is a baseline for our logistic scenario. The important constraint of this approach is to consider only one task allocated for one robot at time. The set of tasks is ordered by the function mentioned below.

Algorithm 1 Pop minimum element

```
procedure PME(T_i, T_j ∈ 𝒯)
    if (demand(T_i) < demand(T_j)) ∧ (dst(T_i) < dst(T_j)) then
        return true
    else
        if dst(T_i) = dst(T_j) then
            return true
        end if
        return false
    end if
end procedure
```

Such function sorts the tasks based on the distance of the unloading bays and the demand of a specific task. For distance we consider the euclidean distance between loading bay L and unloading bays U_j. Instead for demand is the weight of the item.

After the choose of the task to allocate by using the function $f(P)$ return the path distance, precisely the cost of the trail.

For this method we do not use the function $p(\cdot)$ because we have only one task for robot then the task already has the path. As mentioned above, we do not combine routes.

For completeness in this method we do not consider the heuristic function $v(\cdot)$ because the demand for all task is always 1. Because we allocate only one task for one robot at time.

This algorithm take linear time or $O(n)$ time, its time complexity is $O(n)$. This means that the running time increases at most linearly with the size of the input. More precisely, this means that there is a constant c such that the running time is at most cn for every input of size n.

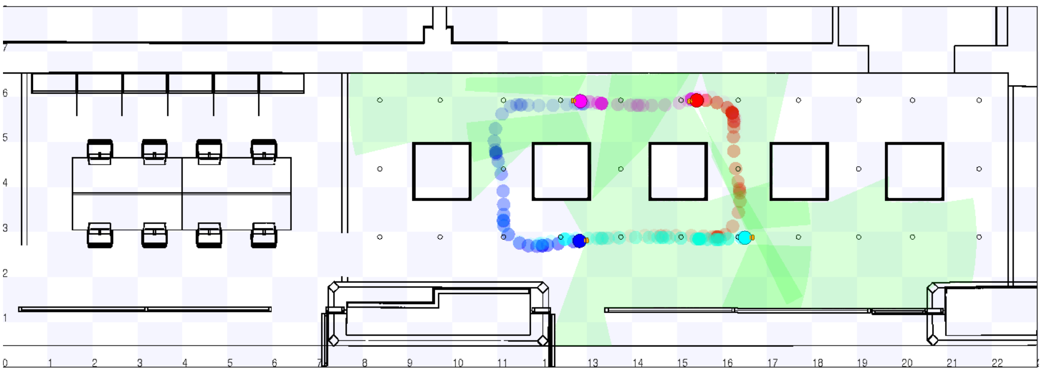


Figure 3.2: Example of execution SR:ST with 4 agents with capacity 4 and 9 tasks.

3.3.2 Set Partition Strategy - Single robot : Multiple task (SPS1:N)

This method consists to compute all possible patitions of the task set using Set Partition algorithm [7] and use only the best partition which is based on the previously mentioned heuristic $v(\cdot)$.

After initialzation phase which all agents send their capacity and identifier, start the partition algorithm [7] that return all possible patitions of subsets combination of tasks P^N.

Then foreach partitions the heuristic $v(\cdot)$ has been used to calculate the loss. For calculate the loss L of a partition we have to calculate the $v(\cdot)$ for all elements in the subsets partition and finally sum all values for all subsets. Furthermore the combination with the lowest loss has been choose. So a subset of combination as been sort in a increasing order. Then the first subset of combination has the lowest value of heuristic function is the first task assigned at the first request from a robot.

Algorithm 2 Set Partition Strategy

```
procedure SPS(P^N)                                  ▷ P^N= partition(𝒯)
                              ▷ define C the maximum capacity of the robots
    for P_i ∈ P^N do
        if demand(P_i) > C then P^N \ P_i
        end if
        for S_j ∈ P_i do v(S_j)
        end for
    end for                                         ▷ sort all P_i for lowest v(·)
    return the first P_i
end procedure
```

The function `partition(·)` take exponential time.

More formally, this algorithm is exponential time because $T(n)$ is bounded by $O(2^{n^k})$ for some constant k.

An example of solution with set $|\mathcal{T}| = 4$:

iteration	**partition size**	**partition**
1	1	{{a, b, c, d}}
2	2	{{a, b, c}, {d}}
3	2	{{a, b, d}, {c}}
4	2	{{a, b}, {c, d}}
5	3	{{a, b}, {c}, {d}}
6	2	{{a, c, d}, {b}}
7	2	{{a, c}, {b, d}}
8	3	{{a, c}, {b},{d}}
9	2	{{a, d}, {b, c}}
10	2	{{a}, {b, c, d}}
11	3	{{a}, {b, c}, {d}}
12	3	{{a, d}, {b}, {c}}
13	3	{{a}, {b, d}, {c}}
14	3	{{a}, {b}, {c, d}}
15	4	{{a}, {b}, {c},{d}}

For $|\mathcal{T}| = 4$ takes 0.000043 seconds. If we increase the set, time increases exponentially. An example of $|\mathcal{T}| = 12$ takes 2.8 seconds and $|\mathcal{T}| = 13$ takes 21.2 seconds.

For this reason in our experiment we limited the size of the task set at most 9 tasks.

3.3.3 Greedy Set Partition Strategy - Single robot : Multiple task (GSP1:N)

The main concept of this approach is composing tasks using Greedy Coalition Formation based on [6] and [5]. Where one wants to minimize the team cost subject to the constraint that each task must be executed by a given number of cooperative agents simultaneosly. Each task requires a number of different demands, and each coalition for the task needs to provide the requered capabilities.

Later initialzation phase which all agents send their capacity and identifier, start the Coalition Formation algorithm [6]. When compute new one

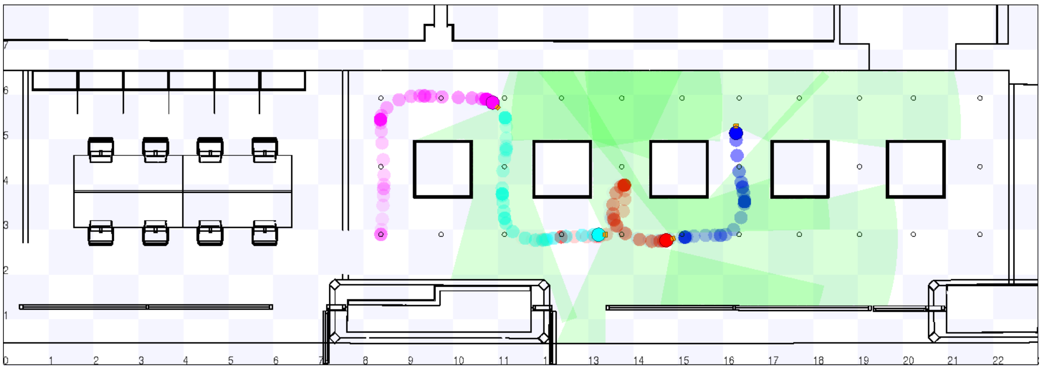


Figure 3.3: Example of execution SPS1:N with 4 agents with capacity 4 and 9 tasks. See example 3.3.3

possible coalition the heuristic $v(\cdot)$ has been used to calculate the loss $L(T_{i,j})$ and if is negative that coalition is insered into the formation and the T_i , T_j are deleted from the task set $\mathcal{T}$.

Algorithm 3 Greedy Coalition Formation

procedure GCF($\mathcal{T}$) $\triangleright$ $\mathcal{T}$ = set of tasks
  $\triangleright$ define C the maximum capacity of the robots
  for $T_i \in \mathcal{T}$ **do**
    for $T_j \in \mathcal{T}$ **do** $\triangleright$ define $T_{i,j} = T_i \cup T_j$
      if $(T_i \neq T_j) \wedge (demand(T_{i,j}) \leq C)$ **then**
        if $v(T_{i,j}) - v(T_i) - v(T_j) < 0$ **then**
          $\mathcal{T} \setminus \{T_i\} \setminus \{T_j\}\ \mathcal{T} \cup \{T_{i,j}\}$
        end if
      end if
    end for
  end for
  return $\mathcal{T}$
end procedure

This algorithm take polynomial time, its running time is upper bounded by a polynomial expression in the size of the input for the algorithm.

More precisely, this algorithm take $O(n^k)$ for some positive constant k.

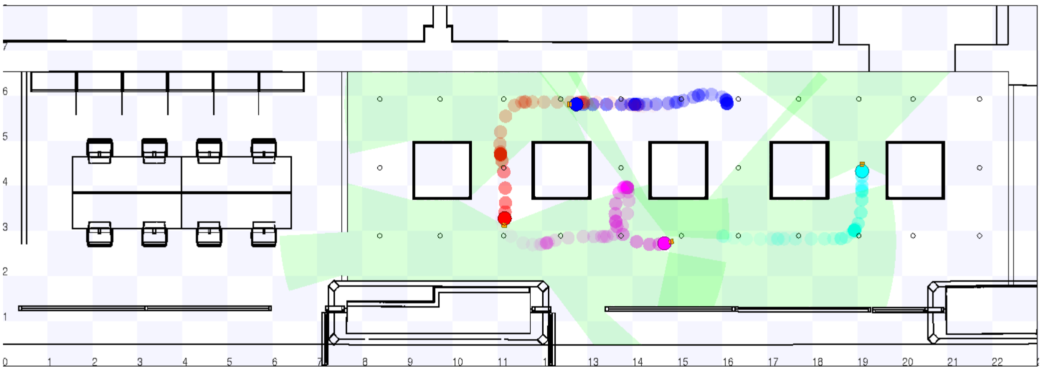


Figure 3.4: Example of execution GSP1:N with 4 agents with capacity 4 and 9 tasks. See example 3.3.3

Example with 9 tasks and 4 robots

Given a set of tasks $\mathcal{T} = \{\{T_0\}, \{T_1\}, \cdots, \{T_8\}\}$ defined like:

$T_i = (item, demand, unloading\_bay)$.

The agents have the same capacity $C_{0,1,2,3} = 4$.

task	item	demand	unloading bay
0	A	1	0
1	B	2	1
2	C	3	2
3	A	1	0
4	B	2	1
5	C	3	2
6	A	1	0
7	B	2	1
8	C	3	2

Table 3.1: The task set $\mathcal{T}$ for 9 tasks

Given a finite task set $\mathcal{T}$ we can see how our strategies works. For this example we have created a video of the simulations execution, in ROS and stage, which are available on YouTube [1].

[1] YouTube site https://youtu.be/XbdBklu98HE

The SPS1:N created 5 orders to perform all elements in the tasks set.
In the table 3.2 we can see the partition composed by subsets of tasks.

task	item	demand	unloading bay
{4,7}	B	4	{1}
{0,1,3}	{A,B}	4	{0,1}
{2,6}	{C,A}	4	{0,2}
5	C	3	2
8	C	3	2

Table 3.2: The result of the Set Partition Strategy (SPS1:N)

The GSP1:N creted 6 orders, one more than SPS1:N , to perform all elements in the tasks set.
In table 3.3 we can see the partition composed by subsets of tasks.

task	item	demand	unloading bay
{3,2}	{A,C}	4	{0,2}
{0,1}	{A,B}	3	{0,1}
{6,4}	{A,B}	3	{0,1}
5	C	3	2
8	C	3	2
7	B	2	1

Table 3.3: The result of the Greedy Set Partition (GSP1:N)

In this Figure 3.5 we can see because the Greedy approach is less complex than SPS1:N . On the execution before to create a coalition is checked if its can be allocated. This check is the value of the loss of the coalition, if is less than zero the colaition is insered in the solution else break the cycle for and pass another coalition.

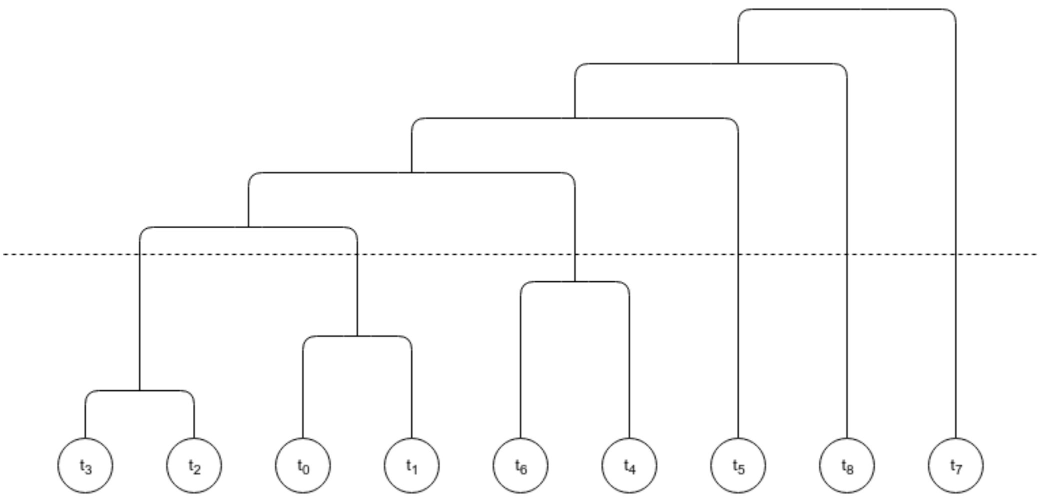


Figure 3.5: The horizontal line represents a cut on executin defines the coalition structure.

Chapter 4

Empirical Setting

Writing software for robots is difficult, particularly as the scale and scope of robotics continues to grow. Different types of robots can have wildly varying hardware, making code reuse non trivial. On top of this, the magnitude of the required code can be daunting, as it must contain a deep stack starting from driver-level software and continuing up through perception, abstract reasoning, and beyond.

4.1 Robot Operating System (ROS)

Our choice fell on ROS (Robot Operating System) which is a widespread open-source, meta-operating system for a robot. It provides several services that are commonly offered by an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It is worth noting that the full source code of ROS is publicly available, ROS is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects.

4.1.1 Nomencalture and Architecture

In this section we simply outline the terminology adopted in the ROS community to allow an easy comprehension of the following discussion.
The fundamental concepts of the ROS implementation are *nodes*, *messages*,

topics, and *services*. In ROS a system is typically comprised of many nodes. In this context, the term *"node"* is interchangeable with *"software module"*. The use of term *"node"* arises from visualization of ROS-based systems at runtime: when many nodes are running, it is convenient to render the peer-to-peer communications as a graph, called the *computation graph*, with process as graph nodes and the peer-to-peer links as arcs.
Nodes communicate with each other by passing *messages*. A message is a a strictly typed data structure. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep. Messages descriptions are usually stored in `my_package/msg/MyMessageType.msg` and define the data structures for messages sent in ROS, called custom message.
Here is a simple example of a `*.msg` file that uses a header, some integer primitive, arrays of integer and array of other `*.msg` files. The message is specified in a language neutral interface definition language (IDL) which uses very short text files to describe its fields and allow an easy composition of complex messages:

```
Header header
bool take
bool go_home
uint32 ID_ROBOT
uint32 item
uint32 order
uint32 demand
uint32 dst
uint32 path_distance
uint32[] route
```

The custom message above rappresent a `Task.msg` which contains the basic information to define a task in the system. Instead, the custom message below, rappresent a `Mission.msg` which is composed of task messages addressed to a specific robot.

```
Header header
uint32 ID_ROBOT
uint32 capacity
Task[] Mission
```

These simple high-level message definitions is then parsed and processed by a code generator module, one for each support language (currentrly `C++`), which generates native implementations that “feel” like native objects, and

are automatically serialized and deserialized by ROS as messages are sent and received.
A node sends a message by publishing it to a given *topic*, which is simply a string such as `/topic` or `/pkg/topic`. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other existence (decoupling). It is important to point out that because nodes connect to each other at runtime, the graph can be *dynamically* modified.
Although the topic-based publish-subscribe model is a flexible communications paradigm, its "broadcast" routing scheme is not appropriate forsynchronous transactions, which can simplify the design of some nodes. For this purpose ROS includes the concept of *services*, defined by a string name and a pair of strictly typed messages: one for the request and one for the response. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.
As for the topic-based paradigm a high-level description of a service is then parsed and processed by a code generator module which generates the corresponding native implementation in a supported target language. Usually `C++` messages are generated in `my_package/msg_gen/cpp/include/my_package`, while `C++` services are generated in `my_package/srv_gen/cpp/include/my_package`.
To support collaborative development, the ROS software system is organized into *packages*. A ROS package is simply a directory which contains an XML file describing the package and stating any dependencies. A collection of ROS packages is a directory tree with ROS packages at the leaves: a ROS package repository may thus contain an arbitrarily complex scheme of subdirectories. This structure is primarily meant to partition the building of ROS-based software into small, manageable chunks of functionality.
In ROS, a *stack* of software is a cluster of nodes that does something coherent as a whole, as is illustrated in the simple *navigation* example reported in Figure. To allow for "packaged" functionality such as a navigation system, ROS provides a tool called `roslaunch`, which reads an XML-like description of a graph and instantiates the graph on the cluster, optionally on specific

hosts. Thus ROS is able to instantiate a set of nodes with a single command, once the nodes are described in a `launch` file, the simple usage is:

```
    roslaunch [package] [filename.launch]
```

4.1.2 The Stage 2D Simulator

For visualization purposes we adopted the Stage 2D robot simulator which provides a virtual world populated by mobile robots and enriched with sensors, actuators and both approximate and exact localization. Stage is designed to be sufficiently simple to allow an easy set-up but at the same time it is intended to be just realistic enough to enable users to move controllers directly between Stage robots and real robots.
Stage is made available in ROS with the `stageros` node which wraps the simulator and exposes its functionality to the rest of the system. The following code reports how it is launched:

```
<?xml version="1.0" encoding="UTF-8" ?>
<launch>
    <arg name="map" default="grid" />
    <arg name="stage_pkg" default="stage_ros"/>
    <arg name="custom_stage" default="false" />
    <group unless="$(arg custom_stage)">
        <node name="stageros" pkg="$(arg stage_pkg)" type="stageros"
        args="$(find patrolling_sim)/maps/$(arg map)/$(arg map).world"
        output="screen" />
    </group>
</launch>
```

The `*.world` file specified tells Stage everything about the world, from obstacles (usually represented via a `*.pgm` image), to robots and other objects. In particular, after the definition of some parameters related to general camera and GUI options, we specify the static map on which the robot has to navigate (we will describe its characteristics shortly) and finally we include two specific files which aims defining the properties of respectively the laser sensor and the robot. The last instruction just throws the robot in the map by indicating it's x, y, z and θ coordinates, this is summarized in:

```
include "../hokuyo.inc"
include "../crobot.inc"
include "../floorplan.inc"
include "../cpoint.inc"
window
( size [ 460 180 1 ]
  rotate [ 0.000 0.000 ]
  center [ 11.5 4.0 ]
  scale 20
  show_data 1)
floorplan
( size [23.0 8.0 1]
  pose [11.5 4.0 0 0]
  bitmap "model5.pgm")
include "robots.inc"
include "point.inc"
```

The first included file (hokuyo.inc) defines the physical and technical properties of the particular laser range finders support that we adopt: we define it to have a circular shape and to be mounted on top of the robot base which has the same circular shape. As for the sensor properties we specify the following parameters described in:

```
define hokuyo ranger
(sensor(
    range [ 0.0 5.0 ] # the max/min range reported by the scanner, in meters.
    fov 230 # the angular field of view of the scanner, in degrees.
    samples 1081 # the number of laser samples per scan.)
  # model properties
  color "orange"
  size [ 0.1 0.1 0.1 ]
  block( points 4
    point[0] [0 0]
    point[1] [0 1]
    point[2] [1 1]
    point[3] [1 0]
```

```
z [0 1]))
```

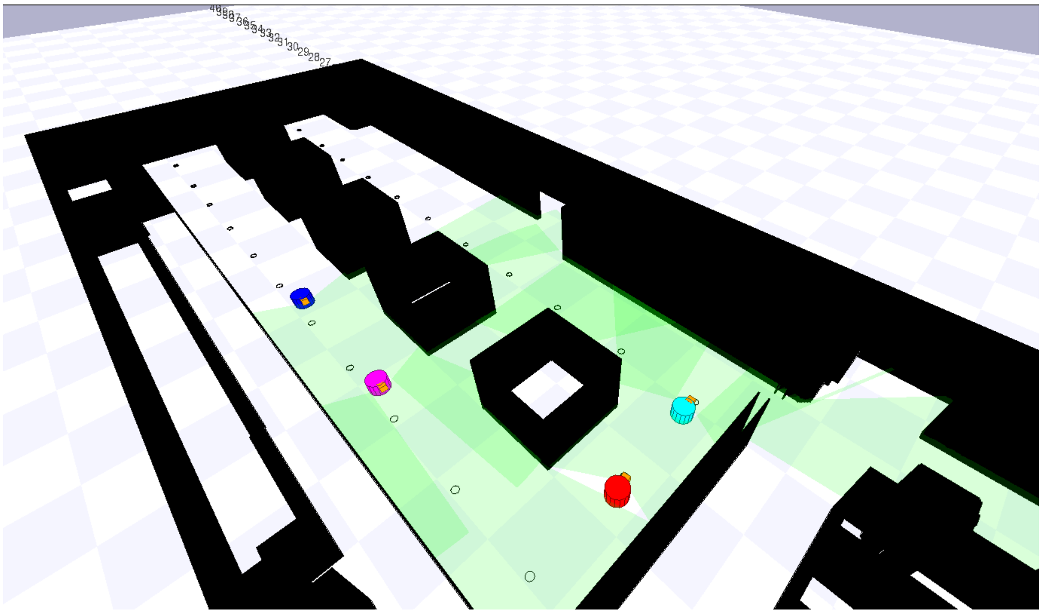
The second included file (crobot.inc) defines the physical properties of the robot, as mentioned above we define it to have a circular shape which is suffice for our purpose of having a mobile camera that moves around the world:

```
define crobot position(
size [0.3 0.3 0.2]
origin [0 0 0 0]
gui_nose 0
drive "diff"
# This block approximates a circular shape of a Robot
block( points 16
    point[0] [ 0.225 0.000 ]
    point[1] [ 0.208 0.086 ]
    point[2] [ 0.159 0.159 ]
    point[3] [ 0.086 0.208 ]
    point[4] [ 0.000 0.225 ]
    point[5] [ -0.086 0.208 ]
    point[6] [ -0.159 0.159 ]
    point[7] [ -0.208 0.086 ]
    point[8] [ -0.225 0.000 ]
    point[9] [ -0.208 -0.086 ]
    point[10] [ -0.159 -0.159 ]
    point[11] [ -0.086 -0.208 ]
    point[12] [ -0.000 -0.225 ]
    point[13] [ 0.086 -0.208 ]
    point[14] [ 0.159 -0.159 ]
    point[15] [ 0.208 -0.086 ]
    z [0 1])
hokuyo( pose [0.15 0 -0.1 0] )
# Report error-free position in world coordinates
localization "gps"
#localization_origin [ 0 0 0 0 ]
# Some more realistic localization error
localization "odom"
```

```
    odom_error [ 0.01 0.01 0.0 0.1 ])
```

4.2 Localization and Navigation

Purpose of this section is to describing how we address the two main problems in the context of mobile robotics: *localization* and *navigation*. The former deals with tracking the pose of the robot during its motion allowing it to localize itself in the map, the latter deals with driving the robot from a starting position to a goal position trying to avoid potential obstacles.



4.2.1 Mapping

The robot navigation system can be initialized with or without an a priori, *static map*. When initialized without a map, the robot only knows about obstacles that it has seen, and will make optimistic global plans through areas that it has not yet visitied which may traverse unknown space, potentially intersecting unseen obstacles. As the robot receives more information about the world, it replans accordingly to avoid obstacles. Initialized with a static map, the robot will make informed plans about distant parts of the environment, using the map as prior obstacle information. In our case we provide ROS with a static map of real laboratory in University of Verona. Doing so requires us to set the *map_server* node that reads a map from disk and offers it via a ROS service:

```
<!-- Run the map server -->
<node name="map_server" pkg="map_server" type="map_server"
args="$(find patrolling_sim)/maps/$(arg mapname)/$(arg mapname).yaml" />
```

The current implementation of *map_server* convert color values in the map image data into ternary occupancy values: *free*(0), *occupied*(100), and *unknown*(-1).
Maps manipulated by the tools in the *map_server* package are stored in a pair of file: the `*.yaml` file describes the map meta-data and names the image file while the image file encodes the occupancy data.
The `*.yaml` file given as an argument to the *map_server* node is reported below with a brief description of it's parameters.

```
# path to the image file containing the occupancy data.
image: src/patrolling_sim/maps/model5/model5.pgm
# resolution of the map, meters/pixel.
resolution: 0.014100
# the 2-D pose of the lower-left pixel in the map, as (x, y, yaw).
origin: [0.000000, 0.000000, 0.000000]
# whether the white/black free/occupied semantics should be reversed.
negate: 0
# pixels with occupancy probability greater than
# this are considered completely occupied.
occupied_thresh: 0.65
# pixels with occupancy probability less than
# this are considered completely free.
free_thresh: 0.196
```

4.2.2 Localization

Both in the case of a given static map or without a given static map, we require that the robot's pose be tracked in a consistent global coordinate frame. When not using a map, the robot's pose is usually estimated by integrating wheel odometry, possibly fused with data from an inertial measurement unit (IMU). When using a map, as in our case, the robot is usually localized using a probabilistic technique, in particular we adopt the Adaptive Monte Carlo Localization (`amcl`) system:

```
<group if="$(arg use_amcl)">
    <!--- AMCL -->
    <include file="$(find patrolling_sim)/params/amcl/amcl_diff.launch" />
    <!-- Override AMCL Frame Params to include prefix -->
    <param name="/$(arg robotname)/amcl/base_frame_id"
        value="$(arg robotname)/base_link"/>
    <param name="/$(arg robotname)/amcl/odom_frame_id"
        value="$(arg robotname)/odom"/>
    <!--common map frame for all robots -->
    <param name="/$(arg robotname)/amcl/global_frame_id" value="map"/>
</group>
```

`amcl` is a probabilistic localization system for a robot moving in 2D. It implements the Adaptive (or KLD-sampling) Monte Carlo Localization approach which uses a particle filter to track the pose of a robot against a known map. The map of the environment where the robot has to localize itself must be given to the robot beforehand. In our case the map is provided by tha *map_server* node. This map is the so called *occupacy map* like before mentioned: it contains a value for every location which indicates the probability that this location is occupied by an object such as a wall. `amcl` takes in a laser-based map, laser scans and outputs pose estimates. On startup, `amcl` initializes its particle filter according to the parameters provided. In this respect we remark that the `amcl` node is launched only after having setting up three categories of ROS parameters that we use to configure its behaviour: *overall filter*, *laser model* and *odometry model*. The complete configuration file is reported in section 6.

4.2.3 Navigation

At high level the navigation system is quite simple. It takes in data from sensors, odometry, and a navigation goal, and outputs velocity commands that are sent to a mobile base. The low-level architecture of system, however, is complex and consist of many components that interacts together.
For navigation purposes we rely on the *move_base* package which provides an interface with the entire ROS navigation stack. The *move_base* package provides an implementation of an *action* that, given a goal in the world, will attempt to reach it with a mobile base. It links together a *global* and *local* planner to accomplish its navigation task, it also maintains two costmaps, one for the global planner and one for the local planner. An architecture view of the node its interaction with other components is show in Figure below.

The pre-requisites of navigation stack, along with a brief description of it's main components, are provided in the section below.

Trasformations

Robotics systems often need to track spartial relationships between *frames* for a variety of reasons: between a mobile robot and some fixed frame of reference for localizzation or, as in our case, between the frame related to the mobile base and the one related to the laser sensor. To simplify the treatment of spartial frames, a trasformation system has been written for ROS, called `tf`. The `tf` system constructs a dynamic *trasformation tree* which realtes all frames of reference in the system. At an abstract level, a trasformation tree define *offsets* in terms of both translation and rotation between different coordinate frames.
Referring to our case of a simple robot consisting of a circular mobile base with a single laser support mounted on top of it we define two coordinate frames: one corresponding to the center point of the base of the robot and one for the center point of the laser support that is mounted on top of the base. We've called the coordinate frame attached to the modile base `base_link` and the coordinate frame attached to the laser support `base_laser_link`. Once this configuration in defined, every time that we have some data from the laser in the form of distances from the laser's center point and we want to

take this data and use it to help the mobile base avoid obastacles in the world we invoke the `tf` system which in the general case transforms the information towards the `base_link` coordiante frame. Below we reported the Figure which rappresent the configuration of `base_link` and the `base_laser_link` frames. Our robot can thus use this information to reason about laser scans in the `base_link` frame and safely plan around obstacles in its environment.

Sensor Information

The navigation stack uses information from sensors to avoid obstacles in the world, it assumes that these sensors are publishing either `sensor_msgs/LaserScan` or `sensor_msgs/PointCloud` message over ROS. Publishing data correctly from sensors over ROS is important for the navigation stack to operate safely. If the navigation stack receives no information from a robot's sensors, then of course it will drive blindly and, most likely, hit things. Our `sensor_msgs/LaserScan` message type, like many other messages sent over ROS, contain `tf` frame and time dependent information. To standardize how this information is sent, the `Header` message type is used as field in all such messages. The three field in the `Header` type are show below.

```
# Standard metadata for higher-level flow data types
uint32 seq
time stamp
string frame_id
```

The `seq` field corresponds to an identifier that automatically increases as messages are sent from a given publisher. The `stamp` field stores time information that should be associated with data in a message. The `frame_id` field stores `tf` frame information that should be associated with data in a message. The body of the `sensor_msgs/LaserScan` message holds informations about any given scan and contains the associated geometric informations in the following form:

```
# Laser scans angles are measured counter clockwise, with 0 facing forward
Header header
float32 angle_min # start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
```

```
float32 angle_increment # angular distance between measurements [rad]
float32 time_increment # time between measurements [seconds]
float32 scan_time # time between scans [seconds]
float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]
float32[] ranges # range data [m] (Note: values < range_min or > range_max
should be discarded)
float32[] intensities # intensity data [device-specific units]
```

Odometry Informations

The navigation stack requires that odometry information be published using `tf` and the `nav_msgs/Odometry` message. The last message in required because `tf` alone does not provide any information about the velocity of the robot, it is thus required that any *odometry source* publishes both this kind of information to *move_base*.
The `nav_msgs/Odometry` message stores an estimate of the position and velocity of a robot in free space:

```
# The pose in this message should be specified in the coordinate frame given
by header.frame_id.
# The twist in this message should be specified in the coordinate frame
given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

The *pose* in this message corresponds to the estimated position of the robot in the odometric frame of reference along with an optional covariance for the certainty of that pose estimate. The *twist* in this message corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for certainty of that velocity estimate.
In our case since the `stageros` node represents the odometric source for the system,it is going to publish all the relevant trasformations between the coor-

dinate frames it manages towards the `/tf` topic which maintains the current trasformation tree for our system, and a `nav_msgs/Odometry` message to the `/odom` topic so that the navigation stack can retrieve velocity information from it.

Base Controller

As previously specified the navigation stack assumes that it can send velocity commands using a `geometry_msgs/Twist` message assumed to be in the base coordinate frame of the robot on the `/cmd_vel` topic. This means there must be a node subscribing to the `/cmd_vel` topic that is capable of talking $(v_x, v_y, v_\theta) \longleftrightarrow$ (`cmd_vel.linear.x`, `cmd_vel.linear.y`, `cmd_vel.angular.z`) veloccities and converting them into motor commands to send a mobile base. In this work the `stageros` node subscribes to the `/cmd_vel` topic simulating the movement of a real robot in a real environment.

4.2.4 Global and Local planner algorithms

The `base_local_planner` is responsible for computing velicty commands to send to the mobile base of the robot given a high-level plan. In particular, this package adheres to the `BaseLocalPlanner` interface specified in the `nav_core` package which provides common interfaces for both *global* and *local* action planning.
As we shall see, the local planner is seeded with the high-level plan produced by the global planner and the purpose of the `nav_core` package is just to provide common interfaces for navigation.

Global planner

The global planner is given the obstacle and cost information contained in the global costmap, information from the robot's localization system, and a goal in the world. From these, it creates a high-level plan for the robot to follow to reach the goal location. In more detail, the global planner will create a series of way-points for the local planner to achieve, these way-points assumes that the robot is circular in shape, and may in fact be infeasible for a

more general case. Also, the global planner doesn't take the dynamics of the robot into account so it can produce plans that are dynamically infeasible as well. While this ensures that the global planner returns in a small amount of time, it also means that the planner is *optimistic* in the plans that it creates. However our robot is not much affected by this problems because it is circular in shape and does not have a particular dynamics to take into account.
The global planner used for this nagivation system is `ROS navigation` stack This planner, as started before, assumes a circular robot and operates on a costmap to compute a navigation function that can later be used to find a *minimum cost plan* from a start point to an end point in a grind.
As outlined before, the global planner may produce a path for the robot that is infeasible, such as a plan that turns through a narrow way too tightly, causing the corners of the robot to hit the surrounding obstacles. Because of its shortcomings, the global planner is used only as a high-level guide for navigation in an environment and we also need a *local planner* for navigation.

Local planner

The local planner is responsible for generating velocity commands for the mobile base that will safely move the robot towards a goal. The local planner is seeded with the plan produced by the global planner, and attempts to follow it as closely as possible while taking into account the kinematics and dynamics of the robot as well as the obstacle information stored in the costmap.
The local planner used for this navigation system is `base_local_planner`. This package supports any robot footprint that can be represented as a convex polygon or circle, so it is fine for our circular-shaped robot. The `base_local_planner` package provides a *controller* that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a trajectory for the robot to move from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells based on their occupancy status. The controller uses this value function to determine v_x, v_y

and v_θ velocities to send to the robot.
In order to generate safe velocity commands to send to the mobile base, the local planner we adopt makes use of a technique known as the *Dynamic Window Approach* (DWA) to forward simulate and select among potential commands based on a cost function. The basic idea is as follows:

- First it discretely samples the robot's control space (which means samples from the set of achievable velocities v_x, v_y and v_θ for just one simulation step given the acceleration limits of the robot);
- Then for each sampled velocity, it performs forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time;
- Next it evaluates (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as *proximity to obstacles, proximity to the goal, proximity to the path produced by the global planner and speed* and it discards illegal trajectories (those that collide with obstacles);
- Finally it picks the highest-scoring trajectory and sends the associated velocity to the mobile base;
- Rinse and repeat.

In order to score trajectories efficiently, a map grid is used. For each control cycle, a grid is created around the robot (the size of the local costmap), and the global path is mapped onto this area. This means some of the grid cells will be marked with distance 0 to a path point, and distance 0 to the goal. A propagation algorithm then efficiently marks all other cells with their *Manhattan distance* to the closest of the points marked with zero. The goal of the global path may often lie outside the small area covered by the map grid, so when scoring trajectories for proximity to goal, what is considered is the "local goal", meaning the first path point which is inside the area having a consecutive point on the global path outside the area.
We report here our configuration file for the local planner we adopt, it is worth noting that through setting the weights on each component of the cost

function differently it is possible to change drastically the behaviour of the robot: for example it is possible to force it to stay close to the projection of the global path into the local costmap, or to stay quite far from obstacles changing consequently it's trajectory across the map. The values below works well for our case since they take into account both the shape of the robot and the needs for computation. For a detailed description parameters see code below.

```
controller_frequency: 5.0
TrajectoryPlannerROS:
# Robot Configuration Parameters
max_vel_x: 1.00
min_vel_x: 0.10
max_trans_vel: 1.00
min_trans_vel: 0.10
max_rot_vel: 1.0
min_in_place_rotational_vel: 0.1
acc_lim_th: 0.75
acc_lim_x: 0.50
acc_lim_y: 0.50
holonomic_robot: false
# Goal Tolerance Parameters
yaw_goal_tolerance: 6.28
xy_goal_tolerance: 0.40
# Controller Parameters
pdist_scale: 0.6
gdist_scale: 0.8
occdist_scale: 0.01
meter_scoring: true
# Forward Simulation Parameters
sim_time: 1.5
vx_samples: 6
vtheta_samples: 20
# Trajectory Scoring Parameters
heading_lookahead: 0.325
dwa: true
```

```
# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05
```

4.3 Cost-maps configurations

The navigation stack uses two costmaps to store information about obstacles in the world. One costmap is used for *global planning*, meaning creating long-term plans over the entire environment, and the other is used for *local planning* and obstacle avoidance.
In particular, the ROS package `costmap_2D` provides an implementation of a 2D costmap that takes in sensor data from the world, build a 2D occupacy grid of the data and inflates costs in a 2D costmap based on the occupacy grid and a user specified *inflation radius*.
Because the robots we study are constrained to drive on flat ground, and cannot, for example, step or jump over obstructions, we assemble obstacle data into a planar costmap on which the planners operates. The costmap is initialized with our laboratory static map, but updates as new sensor data comes in to maintain an up-to-date view of the robot's local and global environment.
As previously specified, the laboratory image describes the occupacy state of each cell of the map in the color of the corresponding pixel. Cells with occupacy probability greater than the value stored in `occuped_thresh` are considered completely occupied and are assigned a lethal cost, meaning that no part of the robot's circular footprint is allowed to be inside of the corresponding two-dimensional cell. Then, inflation is performed in two dimension to propagate costs from obstacles out to user-specified *inflation radius*. Cells that are less than one inscribed radius of the robot away from an obstacle are assigned a uniformly high cost, after which an exponential decay function is applied that will cause the cost to decrease with the distance from the obstacles.

4.3.1 Common configuration

There are some configuration options that we want both global and local costmap to follow. In our case the global configuration settings for both the costmaps are as follows:

```
obstacle_range: 0.50
raytrace_range: 3.0
robot_radius: 0.33
inflation_radius: 0.33
observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: base_laser_link,
 data_type: LaserScan, topic: base_scan, marking: true, clearing: true}
```

The first two parameters set thresholds on obstacle information put into the costmap. The `obstacle_range` parameters determines the maximum range sensor reading that will result in an obstacle being put into the costmap. The `raytrace_range` parameter determines the range to which we will ray-trace freespace given a sensor reading.
Next we set the radius of the robot since it is circular and the inflation radius for the costmap. The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred.
The `observation_sources` parameter defines the sensor that is going to be passing information to the costmap, the last line sets it's parameters. The `sensor_frame` parameter is set to the name of the coordiante frame of the sensor, the `data_type` parameter is set to `LaserScan` since this is the type of message used by the topic, and the `topic` parameter is set to the name of the topic that the sensor publishes data on. The `marking` and `clearing` parameters determine whether the sensor will be used to add obstacle information to the costmap, clear obstacle information from the costmap, or do both.

4.3.2 Global configuration

Below are our configuration settings for the *global* costmap along with a description of their semantics:

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 3.0
  publish_frequency: 0.0
  static_map: true
  inflation_radius: 0.66
```

The `global_frame` parameter defines what coordinate frame the costmap should run in, in this case, we'll choose the `map` frame. The `robot_base_frame` parameter defines the coordiante frame the costmap should reference for the base of the robot. The `update_frequency` parameter determines the frequency, in Hz, at which the costmap will run its update loop. The `static_map` parameter determines whether or not the costmap should initialize itself based on a map served by the *map_server*.

4.3.3 Local configuration

Below are our configuration settings for the *local* costmap along with a description of their semantics:

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 3.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 4.0
  height: 4.0
  resolution: 0.05
```

The `global_frame`, `robot_base_frame`, `update_frequency` and `static_map` parameters are the same as described previously. The `publish_frequency` parameter determines the rate, in Hz, at which the costmap will publish visualization information. Setting the `rolling_window` parameter to true means that the costmap will remain centered around the robot moves through the world. The `width`, `height` and `resolution` parameters set the width [m], height [m] and resolution [m/cell] of the costmap.

4.3.4 Local and global coordinate frames

We now briefly discuss why it is important to distinguish between global and local coordinate frames when building a navigation system. A global coordinate frame, such as the one used by the *global planner* and specified through the `global_frame: /map` parameter, is advantageous in that it provides a globally consistent frame of reference, but it is flawed in that it is subject to discontinuous jumps in its estimation of the robot's position. For example, when the localization system is struggling to determine the robot's position, it is not uncommon for the robot to teleport, on a single update step, from one location to another that is one meter away.

A local coordinate frame, such as the one used by the *local planner* and specified through the `global_frame: /odom parameter`, has no such jumps, but presents its own flaws in that it is prone to drifting over time.

In theory, all planning and obstacle avoidance could be performed in the global frame, but this may lead to problems when discrete jumps in localization occur. To illustrate this, consider the case in which a robot navigates through a narrow way with limited clearance on either side. If the navigation system attempts to plan in a global frame, localization jumps may drastically affects the robot's obstacle information and a jump in the robot's position of just a few centimetres to either side, combined with new sensor data, may actually be enough for the robot to consider that way as unfeasible. If the robot instead operates in a local frame, nearby obstacles are not affected by jumps in localization, and the robot can traverse through the way independently of any difficulties with the localization system. Thus, we use the global coordinate frame to create high-level plans for the robot but also a local coordinate frame for local planning and obstacle avoidance. To relate the two frames, the plan produced by the global planner is mapped from the global coordinate frame into the local coordinate frame on every cycle.

4.4 Recovery behaviours

The navigation system as described until now works well most of the time, attempting to achieve a goal pose with its base to within a user-specified tolerance, but there are still situations where the robot can get stuck. One common cause of failure for the navigation system is *entrapment*, where the robot is surrounded by obstacles and cannot find a valid plan to its goal. When the robot finds itself in this situation, a number of increasingly aggressive recovery behaviours are executed to attempt to clear out space, Figure shows the relations among them. First, obstacles outside of a user settable region will be cleared from the robot's map. If this fails, the robot performs an in-place rotation to attempt to clear out space. If this too fails, a more aggressive map reset is attempted where the robot clears all obstacles that are outside of its circumscribed radius. After this, another in-place rotation is performed to clear space, and if this last step fails, the robot will abort on its goal which it now considers infeasible. After each of these behaviour completes, `move_base` will attempt to make a plan. If planning is successful, `move_base` will continue normal operation. Otherwise, the next recovery behaviour in the list will be executed.

4.5 `logistic_sim` package

The `logistic_sim` is the extension of the `patrolling_sim` package [1]. In this thesis, the problem of allocation task for logistic applications a given environment with an arbitrary number of robots is studied. In this section, details are given on how agents obtain the representation of the environment.

[1] Multi-Robot Patrolling Stage/ROS Simulation Package. http://wiki.ros.org/patrolling_sim

4.5.1 Obtaining a Topological representation

In `patrolling_sim` package the topological map represent the area to travel by a graph $G = (V, E)$ with vertices $v_i \in V$ and edges $e_{i,j} \in E$, enabling robots to assess the topology of its surroundings. In this representation, vertices represent the load, unload bays and the travel locations, the edges represent the connectivity between those locations. The cost of an edge $|e_{i,j}|$ is defined by the metric distance between vertex v_i and v_j. $|V|$ and $|E|$ represent the cardinality of the set V and E, respectively. Seeing as undirected graphs are assumed, then: $|E| \leq \frac{|V|\cdot(|V|-1)}{2}$. A path π is composed of an array of vertices in V.
Since the topological maps considered in this context represent real-world 2D environments, it is assumed that G has the following properties:

- *Undirected*: where $|e_{i,j}| = |e_{j,i}|$ and the edge weights safety the triangle inequality.
- *Connected*: where $\forall v_h, v_i \in V, \exists x = \{v_h, \cdots, v_i\}$.
- *Simple*: where two neighbor vertices v_i and v_j are connected by a unique edge $e_{i,j}$ and no graph loops exist.
- *Planar*: where a pair of edges $e_{g,h}, e_{i,j} \in E$ never crosses each other.

As a consequence of there properties, G is usually *non-complete*; for every pair $v_h, v_i \in V$ there may not exist an edge $|e_{h,i}|$ connecting each pair of vertices.
In this work, it noteworthy that any generic planar graph may be addressed, but for our logistic application in the laboratory map the topological graph G is a *regular grid*.

4.5.2 `actionLib` and `move_base`

The `actionLib` package provides tools to create servers that execute long-running tasks that can be preempted and create clients that interact with servers. Then this library is a standardized interface for interfacing with preemtable tasks. It is used when a node A sends a request to node B to perform some task. If the task is "instantaneous", *services* are suitable. The *actions* are more adequate when task takes time and we want to monitor, have continuous feedback and possibly cancel the request during execution. The action client and server communicate with each other using a predefined action protocol. This action protocol relies on ROS topic in a specified namespace in order to trasport messages.

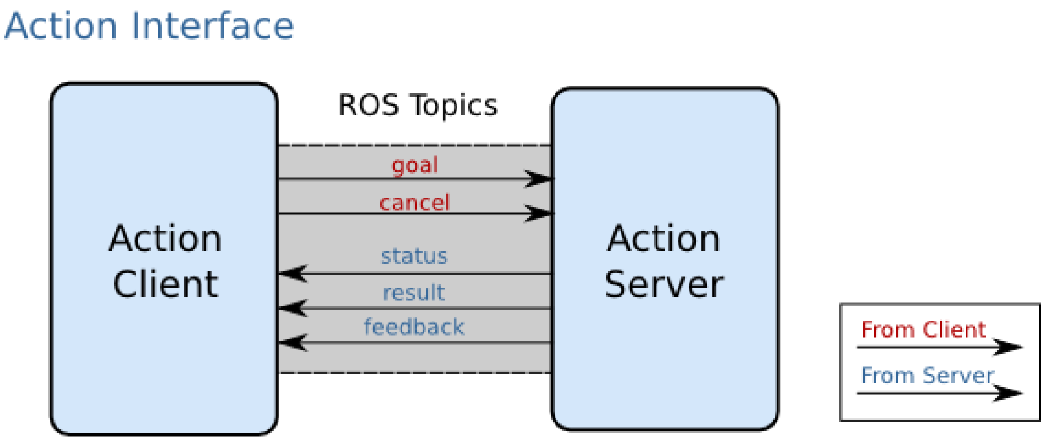


Figure 4.1: Client-Server Interaction

The communication messages used in this library is:

- `goal` - used to send new goals to server.
- `cancel` - used to send cancel requests to server.
- `status` - used to notify clients on the current state of every goal in the system.
- `feedback` - used to send clients periodic auxiliary information for a goal.
- `result` - used to send clients one-time auxiliary information upon completion of a goal.

Action templates are defined by a name and some additional properties through an `*.action` structure defined in ROS. Each instance of an action has a unique *Goal ID* which provides the action server and the action client with a robust way to monitor the execution of a particular instance of an action.

In our system, its used a `SimpleActionServer` that implements a single goal policy. So only one goal can have an active status at a time. When recived new goals preempt previous goals based on the stamp in their *Goal ID* field. And a `SimpleActionClient` which implements a simplified *ActionClient*.

As previously mentioned, the `move_base` package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base.

An example of `move_base` *ActionServer*:

- **Action Subscribed Topics:**
  - `move_base/goal` (*move_base_msgs/MoveBaseActionGoal*): a goal for `move_base` to pursue in the world.
  - `move_base/cancel` (*actionlib_msgs/GoalID*): a request to cancel a specific goal.
- **Action Published Topics:**
  - `move_base/feedback` (*move_base_msgs/MoveBaseActionFeedback*): feedback contains the current position of the base in the world.
  - `move_base/status` (*actionlib_msgs/GoalStatusArray*): provides status information on the goals that are sent to the `move_base` action.
  - `move_base/result` (*move_base_msgs/MoveBaseActionResult*): result is empty for the `move_base` action.

4.5.3 `task_planner` package

We extend the `patrolling_sim` package creating a new extern package which comunicate with it. The `task_planner` generate the finite task set, its receive the initialization message from each robot and depending on the message start the different algorithm. Then this package implements the centralized coordinator and return the `mission` for the robots.

In Figure 4.2 we can see the communication between robots (`/patrol_robot`N) and `task_planner`.

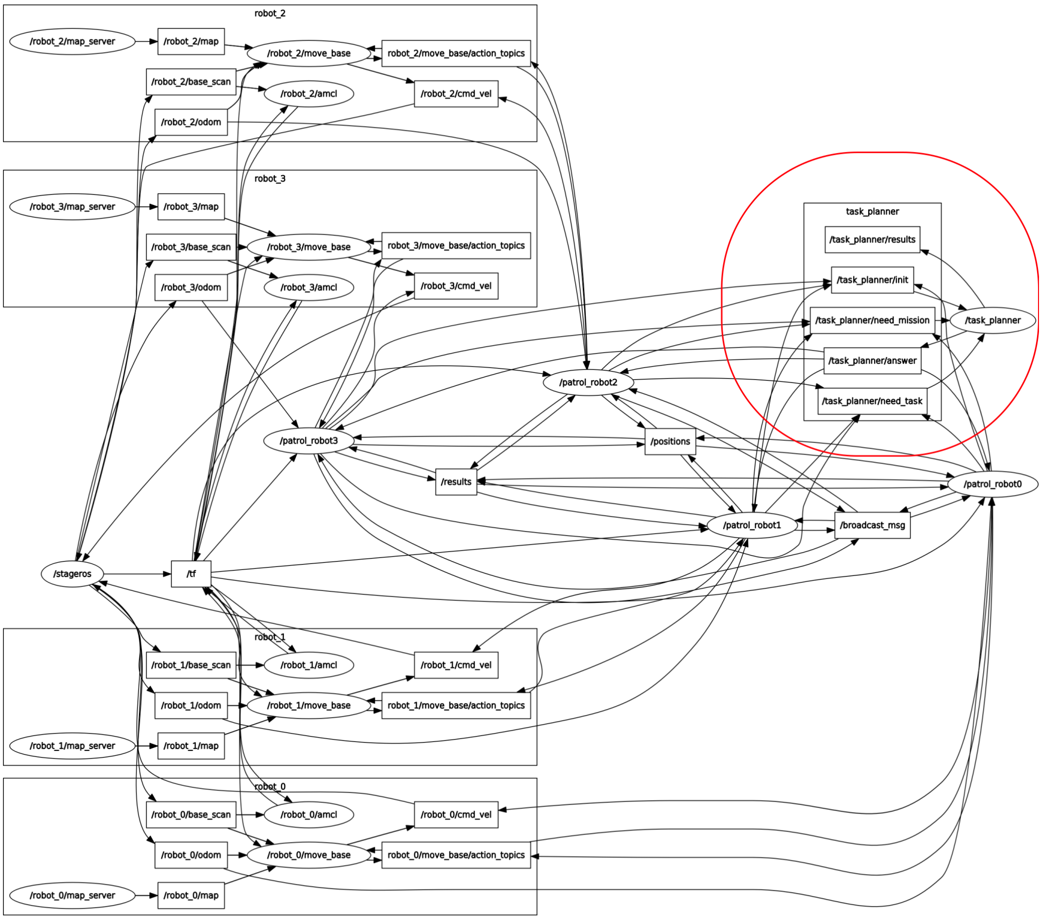


Figure 4.2: Report the `rqt_graph`

The most important topisc are:

- `/task_planner/init`: receive the initialization message.
- `/task_planner/need_task`: receive the task request.
- `/task_planner/need_mission`: receive the mission request.
- `/task_planner/answer`: after computation phase for different algorithm return the subset of task must be allocate.

Chapter 5

Empirical Results

In this chapter we will present the results of the empirical experiments conducted to test the three strategies in a simulated autonomous warehouse with a central coordinator.

The experiments were performed on Intel Core i5-2520M CPU 2.50GHz $\times$ 4 , ram 7,7 GiB [1].

We were made 10 executions for each configuration. The configuration of our experiments are:

- the size of the task set $\mathcal{T}$, the fisrt number reported on configuration column.
- the capacities C of the robots team. Note that in the basic method (SR:ST) we do not consider the capacity of robots because a task is assigned to each robot regardless of the demand of the tasks.
- the last number reported on configuration column is the number of the team size.

All of this experiments is performed on ICE laboraty map [2].

The second column of the table 5.1 reported the algorithm used to that experiment. For see the different results we sorted the algorithms for the average time to notice the improving performance aspects.

[1] For more about architecture used https://support.lenovo.com/my/it/solutions/pd015734

[2] See on site the Computer Engineering for Industry 4.0 http://www.di.univr.it/

The next column reported the average time of the experiments and the standard error of the mean (SEM)[3].

The study sample can be described entirely by two parameters: the mean and the standard deviation (σ). The σ represents the variability within the sample; the larger the σ, the higher the variability within the sample. Although it is clear that samples should always be summarized by the mean and σ. The $\bar{\sigma}$ is used in inferential statistics to give an estimate of how the mean of the sample is related to the mean of the underlying population. Although the σ and the $\bar{\sigma}$ are related, they give two very different types of information. Whereas the σ estimates the variability in the study sample, the $\bar{\sigma}$ estimates the precision and uncertainty of how the study sample represents the underlying population. In other words, the σ tells us the distribution of individual data points around the mean, and the $\bar{\sigma}$ informs us how precise our estimate of the mean is.

The formula for the sample standard deviation is:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}}$$

where $\{x_1, \cdots, x_N\}$ are the observed values of the sample items, $\bar{x}$ it the mean value of these observations, and N is the number of observations in the sample.

The standar error of the mean ($\bar{\sigma}$) can be expressed as:

$$\bar{\sigma} = \frac{\sigma}{\sqrt{N}}$$

where σ is the standar deviation of the population and N is the size (number of observations) of the sample.

In the next columns are reported the average time of the simulation ($\overline{Time}$), tha average interference ($\overline{Interference}$) and the average distance traveled ($\overline{Distance}$). In the last column are reported the average $\bar{\sigma}$ for the team robots. In other worlds for one experiment are calculated the $\bar{\sigma}$ of the distance for every robot and after is calculated the average.

[3]See https://en.wikipedia.org/wiki/Standard_error

From the table 5.1 shown we can notice, as previously expected considerations, that the strategy GSP1:N approximates the SPS1:N strategy. Based on the complexity of the strategies used, the results respects inital expectations. Because using a polynomial complexity it is possible to have a solution close to the calculated solution for a more complex and complete problem. In other words, by calculating all the possible solutions we can have a good task allocation spending less computational time.

In the last column ($\bar{\sigma}(Distance)$) we can see that the task allocation is not always the same. It means that in different experiments, which the same configuration, are assigned different tasks for different robots. If the value of the $\bar{\sigma}$ is equal to zero then it means that the task allocation is always the same for 10 times, that define a tightly allocation. In contrast, higher value of $\bar{\sigma}$ define a loosely allocation.

Configuration	**Algorithm**	$\overline{Time}$	$\overline{Interference}$	$\overline{Distance}$	$\bar{\sigma}(Distance)$
6/-/2	SR:ST	218.32[±6.19]	63.45	3747.90	87.8
6/3/2	GSP1:N	194.52[±6.42]	49.65	3401.15	251.37
	SPS1:N	177.00[±1.99]	49.34	3132.5	0
6/5/2	GSP1:N	142.08[±1.39]	42.2	2714.25	206.43
	SPS1:N	138.98[±2.41]	39.38	2601.25	156.47
6/-/4	SR:ST	124.52[±3.12]	42	2194.75	114.2
6/3/4	GSP1:N	117.44[±1.85]	35.75	1769	43.83
	SPS1:N	115.28[±4.10]	33.5	1702.5	23.67
6/5/4	GSP1:N	93.4[±1.01]	29	1688.5	34.5
	SPS1:N	91.8[±2.14]	30.75	1546.5	35.8
9/-/2	SR:ST	292.24[±3.06]	85.5	5201.5	34.76
9/3/2	GSP1:N	265.72[±2.64]	71.5	4491.5	0
	SPS1:N	240.74[±10.42]	75.5	4232.5	310.43
9/5/2	GSP1:N	232.84[±4.71]	68.85	4041.25	236
	SPS1:N	168.34[±2.03]	50.5	3132.5	0
9/-/4	SR:ST	178.55[±4.23]	52	2755.75	135.8
9/3/4	GSP1:N	152.55[±2.87]	46.75	2200	113.4
	SPS1:N	134.23[±3.25]	40.63	2182.5	27
9/5/4	GSP1:N	134.23[±3.26]	40.6	2098.3	93.45
	SPS1:N	93.05[±5.15]	32.25	1530.25	0
21/-/2	SR:ST	629.10[±8.84]	154.6	11773.5	229.75
21/3/2	GSP1:N	561.93[±8.00]	134.3	10133.16	201.2
21/5/2	GSP1:N	497.45[±6.15]	126	9079	210.4
21/-/4	SR:ST	402.12[±5.06]	132.25	6232.35	295.1
21/3/4	GSP1:N	343.23[±6.10]	98.23	5231.25	342.2
21/5/4	GSP1:N	294.40[±7.60]	77.63	4683.25	367.5

Table 5.1: Results of our experiments on ICE Laboratory

Chapter 6

Conclusions and Future Work

In this thesis we have shown how our system can perform pickup-and-delivery tasks. We evaluate the performance of our system in a realistic simulation enviroment, precisely on the ICE Laboratory build with ROS and stage. In particular we compared our task assignment approches with the baseline greedy approch.

We have discovered that a coalition formation problem can approximate the results of a set partion problem in less time complexity.

Because we are focused on a centralized coordinator in the future works we want to perform a distributed coordiantor using a recent stategy studied before to implement our approaches. In more detail on [3] they used a Token Passing strategy with kinematic constraints but consider only one task assigned to one robot at time. We want to extend this limitations considering the possibility of assigning more tasks in one travel. That strategy should be more flessible, adaptive at the situation on the traveling orders then fault-tolenace.

The scope set at the beginning of the thesis have been reached. The results comfirmed the expectations.

Then in conclusion the GSP1:N algorithm approximate the SPS1:N strategy on the same task set, calculating the solution in polynomial time complexity.

Apendices

Appendix A

ROS: amcl configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<launch>
<arg name="robotname" default="robot_0" />
<node pkg="amcl" type="amcl" name="amcl" args="scan:=base_scan">
  <!-- Publish scans from best pose at a max of 10 Hz -->
  <param name="odom_model_type" value="diff"/>
  <param name="odom_alpha5" value="0.1"/>
  <param name="transform_tolerance" value="0.2" />
  <param name="gui_publish_rate" value="-10.0"/>
  <param name="laser_max_beams" value="30"/>
  <param name="min_particles" value="100"/>
  <param name="max_particles" value="500"/>
  <param name="kld_err" value="0.05"/>
  <param name="kld_z" value="0.99"/>
  <param name="odom_alpha1" value="0.2"/>
  <param name="odom_alpha2" value="0.2"/>
  <!-- translation std dev, m -->
  <param name="odom_alpha3" value="0.8"/>
  <param name="odom_alpha4" value="0.2"/>
  <param name="laser_z_hit" value="0.5"/>
  <param name="laser_z_short" value="0.05"/>
  <param name="laser_z_max" value="0.05"/>
  <param name="laser_z_rand" value="0.5"/>
```

```
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.2"/>
    <param name="update_min_a" value="0.5"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_link"/>
    <param name="resample_interval" value="1"/>
  </node>
  </launch>
```

Appendix B

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	6	-	218.33
robot	**processed task**	**distance**	**interference**
0	3	3739	63
1	3	3757	57

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	6	3	**194.52**
robot	**processed task**	**distance**	**interference**
0	2	3286	48
1	2	3516	51

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	6	3	177
robot	**processed task**	**distance**	**interference**
0	2	2970	47
1	2	3295	52

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	6	5	142
robot	**processed task**	**distance**	**interference**
0	2	3074	47
1	1	2355	38

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	6	5	138.98
robot	**processed task**	**distance**	**interference**
0	2	2847	43
1	1	2355	35

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	6	-	124.52
robot	**processed task**	**distance**	**interference**
0	2	2060	31
1	2	2337	59
2	1	2434	38
3	1	1949	40

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	6	3	117.44
robot	**processed task**	**distance**	**interference**
0	1	1860	32
1	1	1853	31
2	1	1675	40
3	1	1688	40

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	6	3	**115.28**
robot	**processed task**	**distance**	**interference**
0	1	1897	30
1	1	1854	37
2	1	1647	35
3	1	1412	32

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	6	5	93.4
robot	**processed task**	**distance**	**interference**
0	1	1753	32
1	1	1960	35
2	1	2414	34
3	**0**	(627)	(15)

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	6	5	**91.8**
robot	**processed task**	**distance**	**interference**
0	1	1105	29
1	1	2407	37
2	1	1959	33
3	**0**	(715)	(24)

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	9	-	292.24
robot	**processed task**	**distance**	**interference**
0	5	5631	93
1	4	4772	78

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	9	3	265.72
robot	**processed task**	**distance**	**interference**
0	3	4578	69
1	3	4405	74

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	9	3	240.74
robot	**processed task**	**distance**	**interference**
0	3	4412	79
1	3	4571	82

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	9	5	232.84
robot	**processed task**	**distance**	**interference**
0	3	4207	71
1	3	3875	67

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	9	5	**168.34**
robot	**processed task**	**distance**	**interference**
0	2	2970	45
1	2	3295	56

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	9	-	178.55
robot	**processed task**	**distance**	**interference**
0	3	3582	64
1	3	2833	55
2	2	2635	46
3	1	1973	43

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	9	3	152.55
robot	**processed task**	**distance**	**interference**
0	2	2921	61
1	2	2167	49
2	1	1994	39
3	1	1718	38

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	9	3	134.23
robot	**processed task**	**distance**	**interference**
0	2	2826	52
1	1	1960	35
2	1	2135	39
3	1	1809	36

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	9	5	134.23
robot	**processed task**	**distance**	**interference**
0	2	2785	52
1	1	1960	35
2	1	2135	39
3	1	1514	36

Algorithm	**Number of task**	**Capacity**	**Time**
SPS1:N	9	5	**93.05**
robot	**processed task**	**distance**	**interference**
0	2	1374	42
1	1	2067	35
2	1	2054	36
3	**0**	(626)	(16)

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	21	-	629
robot	**processed task**	**distance**	**interference**
0	11	12091	165
1	10	11456	145

Algorithm	**Number of task**	**Capacity**	**Time**
*GSP1:N	21	3	561.93
robot	**processed task**	**distance**	**interference**
0	7	9943	130
1	7	10323	139

Algorithm	**Number of task**	**Capacity**	**Time**
@GSP1:N	21	5	497.45
robot	**processed task**	**distance**	**interference**
0	6	8816	115
1	6	9343	137

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	21	-	402
robot	**processed task**	**distance**	**interference**
0	6	7069	152
1	5	5731	119
2	5	5928	109
3	5	6201	129

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	21	3	343.23
robot	**processed task**	**distance**	**interference**
0	4	6134	97
1	4	5731	103
2	3	5452	95
3	3	4201	85

Algorithm	**Number of task**	**Capacity**	**Time**
GSP1:N	21	5	294.4
robot	**processed task**	**distance**	**interference**
0	3	4905	83
1	3	5455	72
2	3	4146	82
3	3	4227	75

Algorithm	**Number of task**	**Capacity**	**Time**
SR:ST	42	-	1283.2
robot	**processed task**	**distance**	**interference**
0	21	22243	335
1	21	22651	338

Bibliography

[1] L.E. Parker, *Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems*, March 2008.

[2] A. Farinelli, N. Boscolo, E. Zanotto, E. Pagello, *Advanced Approaches for Multi-Robot Coordination in Logistic Scenarios*, April 2017.

[3] H. Ma, W. Hönig, T.K.S. Kumar, N. Ayanian, S. Koenig, *Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery*, December 2018.

[4] T. Andre, D. Neuhold, C. Bettstetter, *Coordinated Multi-Robot Exploration: Out of the Box Packages for ROS*, December 2014.

[5] A. Farinelli, M. Bicego, F. Bistaffa, S. D. Ramchurn, *A Hierarchical Clustering Approach to Large-scale Near-optimal Coalition Formation with Quality Guarantees*, March 2017.

[6] X. Zheng, S. Koenig, *Greedy Approaches for Solving Task-Allocation Problems with Coalitions* , 2008.

[7] M. Orlov, *Efficient Generation of Set Partitions*, March 2002.

[8] A. Farinelli, L. Iocchi, D. Nardi, *Multirobot Systems: A Classification Focused on Coordination*, October 2004.

[9] R. Zlot, A. Stentz, *Market-based Multirobot Coordination for Complex Tasks*, January 2006.

[10] H. MA, J. Li, T. K. S. Kumar, S. Koenig, *Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks*, May 2017.

[11] R. Stranders, A. Farinelli, A. Rogers, N. R. Jennings, *Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm*, 2009.

[12] A. Farinelli, L. Iocchi, D. Nardi, *Multirobot Systems: A Classification Focused on Coordination*, October 2004.

[13] , C. Jones, M. J. Mataric, *Automatic Synthesis of Communication-Based Coordinated Multi-Robot Systems*, October 2004.

[14] , P. R. Wurman, R. D'AndreA, M. Mountz, *Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses*, March 2008.