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

Correlatore:
Riccardo Muradore

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 present and evaluate the principal difference of central and distributed allocator task coordinator. In these applications we address off-line coordination, by casting the Multi-Robot logistics problem as a task assignment problem and proposing two solution techniques: Cyclic Greedy Strategy Single Robot Single Task (CGS1:1), which is a baseline greedy approach, and Cyclic Optimaze Strategy Single Robot Multiple Task (COS1:N), which is based on merging task for improve the spend time.

And the last one is address on-line coordiantor, that is based on token passing (TP) approach. 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 previous off-line and on-line methods.

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

Chapter 1

Introduction

One of the fundamental areas in Robotics is multi-robot systems. 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.
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 Multi-Robot systems in general and, more specifically, agents in logistic missions are herein introduced to lay the groundwork to approach the problem in hands. Finally, an overview of the document is given.

1.1 Context and Motivation

In recent years, robotics has been one of the scientific fields with the most substantial advances. Within the diverse areas that it embraces, mobile robotics has had great focus in the last decades from roboticists (i.e., researchers on robotics) around the world. In particular, issues like autonomous navigation, path planning, self-localization, coordination of robots, cooperative dynamics, mapping, exploration and coverage have become popular and have benefited from the progress of artificial intelligence, control theory, real-time systems, sensors' development, electronics, communication systems and systems integration [Parker, 2008].
Nowadays, we expect to see robots with many different shapes 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. Com-

bining 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' expendability. In particular, multi-robot allocator task for logistic applications has high utility and is considered as a contemporary area with some relevant work presented in the last decade, especially in terms of strategies for coordinating teams of robots. However, many of the studies in the literature present unrealistic simplifications, strong limitations or questionable applicability as illustrated later on. Therefore, there is an eminent potential to explore in this context.

Moreover, the allocator task for logistic applications problem is very challenging in the context of Multi-Robot systems, because agents must navigate autonomously, coordinate their actions in a distributed or centralized way and acquire information about the surrounding space, possibly with communication constraints and independently of the number of robots in the team and the enviroment's dimension. 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 multi-robot systems, in general.

1.2 Multi-Robot Systems

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 multi-robot system. In some cases, due to the need of combining different tasks and the dynamics of the environment.

Some characteristics of multi-robot systems 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.

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 The Multi-Robot system 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 Multi-Robot system. To understand this problem, it is important to firstly introduce the definition of logistic application.

Definition 1. *Industrial Logistics, the set of operations related to the procurement, destination and storage of materials and products of large industry; the coordination and provisioning of people or things for the purpose of higher production efficiency.*

Many real-world applications of Multi-Robot systems require agents to operate in known common environments. The agents are constantly engaged with new tasks and have to navigate between locations where the tasks need to be executed. On the other hand, the Multi-Robot system for logistic applications, given a set of agents attend to stream of incoming pickup-and-delivery tasks.

Chapter 2

Background and Related Works

In this section, we detail the main issues for Multi-Robot system 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 Industrial Applications

In this thesis I also focus on industrial scenarios where robots have a high degree of autonomy and operate in a dynamic environment. In this work, I consider a similar setting where a set or robots are involved in trasportations tasks for logistics. However, I focus on the specific problem of task assignment ...

2.2 Coordination in Multi-Robot system

Coordination for Multi-Robot system (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.
Given my focus on logistic scenarios, here I restrict my attention to coordination approaches based on optimization and specifically on task assignment as this the most common framework for my reference application domain.

Chapter 3

Problem

In this section I detail my reference scenario for MRS coordination and formalization problem.

3.1 Description

My reference scenario is based on a warehouse that stores items of various types. Such items must bo composed together to satisfy orders that arrive based on customers' demand. The items of various types are stored in particular section 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 my domain a set of robots is responsible for for transporting items from the loading bays 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. Specifically, here we propose to take this interferences into account in the task assignment process and assign tasks to robots so to reduce the possible interferences among the transportation robots.

3.2 Fomalization

In this section I formalize the MRS coordination problem described above as a task allocation problem where the robots must be allocated to transportation tasks. In my formalization if transportation tasks are more than the available robots at each time step only a subset of tasks will be allocated. However, since the task allocation process is repeated over time robots effectively serve a sequence of tasks. In more detail, my 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, and we assume to have no knowledge on future orders. The orders induce a set of $N \times M$ trasportation tasks $T = t_{i,j}$, with $t_{i,j} =< d_{i,j}, dst_{i,j}, P_{i,j} >$, 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. I 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.

Chapter 4

Solution

In this section I propose the approaches with centralized coordination. The first strategy, mentioned above, is the CGS1:1 which consider only one task allocated for one robot. The second strategy extends the first, the main concept of this strategy is merging the tasks for optimaze the capacity of robot.

4.1 Cyclic Greedy Strategy Single robot : Single task (CGS1:1)

4.2 Cyclic Optimaze Strategy Single robot : Multiple task (COS1:N)

Chapter 5

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.

5.1 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.

5.1.1 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]
```

5.1.2 The Stage 2D Simulation

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"/> <!-- stage_pkg:=stage for ROS Groovy -->
    <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)/$(arg map).world" output="screen" />
    </group>
    <group if="$(arg custom_stage)">
        <node name="stageros" pkg="$(arg stage_pkg)" type="stageros"
        args="$(find patrolling_sim)/maps/$(arg map)/$(arg map).world" output="screen">
            <param name="base_frame" value="base_link" />
            <param name="laser_topic" value="base_scan" />
            <param name="laser_frame" value="base_laser_link" />
        </node>
    </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 ]
)
```

5.2 Localization and Navigation

5.2.1 Mapping

5.2.2 Localization

5.2.3 Navigation

Trasformations

Sensor Information

Odometry Informations

Base Controller

5.2.4 Global and Local planner algorithms

Global planner

Local planner

5.3 Cost-maps configurations

5.4 Recovery behaviours

Chapter 6

Experiments

Chapter 7

Conclusions and Future Work

Acknowledgements