UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Distributed Control Systems

Distributed Simplex for Cooperative Multi-robot Assignment

Professor: **Giuseppe Notarstefano**

Student:
Nicola Franco

Academic year 2018/2019

Abstract

An implementation of a distributed version of the well-known simplex algorithm, is given in this report. The aim is to solve degenerate linear programs on asynchronous peer-to-peer networks with distributed information structures. So, a network of agents is considered, where every agent knows only a subset of information, and they run simplex algorithm by updating and sharing their basis, until they will agree on a common solution. It is shown how the multi agent assignment problem can be efficiently solved by using a distributed structure, and it is provided simulations in the robot operating system environment.

Contents

List of Figures

Introduction

The present report provides a methodological approach to define and solve a multi-agent assignment problem. The goal is to generate a distributed algorithm, with local and sharing information structure, used to solve the assignment problem.

Motivations

The interest in performing complex task via multi agent systems, has raised the interest in solving distributed optimization problems. The assignment problem directly motivates the distribution of information throughout the network and emphasizes an important challenge for distributed algorithms: the non-uniqueness of optimal solutions or, more formally, the degeneracy of optimization programs.

Contributions

As main contribution, an implementation of a distributed version of the well known simplex algorithm to solve these degenerate linear programs in asynchronous networks with time-varying directed communication topology. Then, a proof that a network of agents running the algorithm will agree in finite time on a common optimal solution of the linear program even if it is highly degenerate.

Organization

In Chapter 1, it is defined the set-up in order to understand and implement the distributed algorithm which it is made as more general as possible.
In Chapter 2, an explanation of the structures used in the code.
In Chapter 3, it is shown a simulation platform by using the developed software, to simulate a fleet of wheeled-robots.

Chapter 1

Set-up and Description

In this chapter, a generic overview of the problem and then more in deep, providing a detailed description of the implemented solution. The framework is an integer optimization problem, that has been rewritten in a suitable form to be treated as a minimization problem. Here, the centralized version of the problem useful to check the correctness of the distributed algorithm.

1.1 The algorithm structure

We are interested in a network of autonomous agents labeled $V_a = 1, 2, .., N$ which are able to receive information from their neighbours, and a set of tasks $U_a = 1, 2, .., N$. Neighbour relations between distinct pairs of agents are characterized by directed graph $\mathbb{G}_c$ with N vertices.
The assignment problem can be illustrated with a bipartite assignment graph $\mathbb{G}_a = \{V_a, U_a; E_a\}$, where an edge $(i, k) \in E_a$ exists if and only if agent i can be assigned to the task k. The cost $c_{ik} \in \mathbb{Z}$ for agent i to perform the task k is a weight on the edge (i, k). For each agent i and task k a binary decision variable $x_{ik} \in \{0, 1\}$ is introduced, which is 1 if agent i is assigned to task k and 0 otherwise.

The assignment problem corresponds to the *integer optimization problem* with the constraint that a full assignment is achived:

$$\begin{aligned} \min_{x \geq 0} & \sum_{(i,k) \in E_a} c_{ik} x_{ik} \\ s.t. & \sum_{k|(i,k) \in E_a} x_{ik} = 1, \forall i \in 1, ..., N, \\ & \sum_{i|(i,k) \in E_a} x_{ik} = 1, \forall k \in 1, ..., N, \end{aligned}$$

Note that the linear program has $n = |E_a|$ decision variables with $|E_a| \leq N^2$, and $d = 2N$ equality constraints, as described in [1].

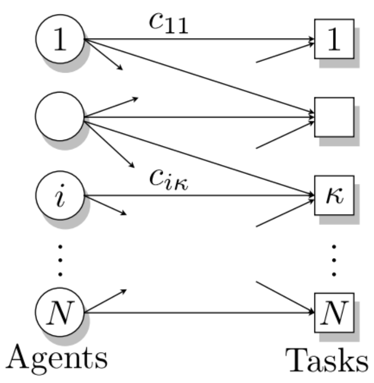


Figure 1.1: Assignment graph

1.2 Distributed linear programming set-up

We can consider the linear program in standard equality form:

$$\begin{aligned} &\min c^T x \\ &s.t. \quad Ax = b, \quad x \geq 0 \end{aligned}$$

Where $A \in \mathbb{R}^{d \times n}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}^n$, are the problem data and $x \in \mathbb{R}^n$ is the vector of decision variables. A column of the linear program is a vector $h_i \in \mathbb{R}^{1+d}$ defined as

$$h_i := [c_i, A_i^T]^T$$

where $c_i \in \mathbb{R}$ is the i-th entry of c and $A_i \in \mathbb{R}^{d \times 1}$ is the i-th column of the matrix A. The set of columns is denoted by $\mathbb{H} = \{h_i\}_{i \in \{1,..,N\}}$.
We assume that every agent initially knows a subset of problem columns, called permanent columns. This means there exists a partitioning

$$\wp = \{\mathbb{P}^{[1]}, \ldots, \mathbb{P}^{[N]}\}$$

of the problem columns with

$$\mathbb{H} = \cup_{i=1}^{N} \mathbb{P}^{[i]} \quad and \quad \mathbb{P}^{[i]} \cap \mathbb{P}^{[j]} = \emptyset$$

and agent i initially knows the column set $\mathbb{P}^{[i]}$, as described in [1].

1.3 Distributed simplex algorithm

The formalization of this distributed algorithm is given as follows. The state of each agent is a lex-feasibile basis, $w^{[i]}(t) = \mathbb{B}^{[i]}$, so a set of basis columns sufficies to fully determine the solution of a linear program, and thus every agent keeps and updates a copy of a possible solution. To initialize the algorithm, every agent creates an artificial basis using a method known as Big-M method. So, the corresponding initial basis $\mathbb{B}_M$ is as follows: it chooses $A_{B_M} = \mathbb{I}_d$ and $c_{B_M} = M \cdot \mathbf{1}$, where the artificial value M is larger than any other cost coefficient in the original problem, i.e., M can be chosen as the largest number realizable by the processor. The state transition function STF performed by each agent to update its basis $\mathbb{B}^{[i]}$ proceeds as follows. Having received columns from its in-neighbours, and having its set of permanent columns $\mathbb{P}^{[i]}$ in its memory, the agent performs two actions:

- it sorts all columns in its memory lexicographically
- it performs the Simplex algorithm, and updates its basis with the new, improved basis.

We provide a pseudo code description of the algorithm in the following table.

```
Problem data:       $\mathbb{G}_c(t), (\mathbb{H}, b), \mathbb{P})$ ;
Processor state:    $\mathbb{B}^{[i]} \subseteq$ with card $(\mathbb{H}) = d$ ;
Initialization:     $\mathbb{B}^{[i]} := \mathbb{B}_M$ ;
Function MSG($\mathbb{B}^{[i]}$,j) **begin**
    **Result:** $\mathbb{B}^{[i]}$
**end**
Function STF($\mathbb{B}^{[i]}, \cup_{j \in \mathbb{N}(i,t)} y_j$) **begin**
    **if** $y_j \neq$ *null for all* $j \in \mathbb{N}_l(i,t)$ **then**
        $\mathbb{H}^{tmp} \leftarrow$lexsort $\{\mathbb{P}^{[i]} \cup \mathbb{B}^{[i]} \cup (\cup_{j \in \mathbb{N}(i,t)} y_j)\}$
        $\mathbb{B}^{[i]} \leftarrow$ Simplex($\mathbb{H}^{tmp}, \mathbb{B}^{[i]}$)
    **end**
    **else**
        $\mathbb{B}^{[i]} \leftarrow$ null
    **end**
**end**
```

```
**Algorithm 1:** Simplex($\mathbb{H}, \mathbb{B}$)
**Data:** A set of columns $\mathbb{H}$, a lex-feasible basis $\mathbb{B} \in \mathbb{H}$
**while** $\exists e \in \mathbb{H}$ *such that* $[r^T_{\{B \cup e\}} c_{\{B \cup e\}}, r^T_{\{B \cup e\}}] \prec 0$ **do**
    $\mathbb{B} \leftarrow$ Pivot($\mathbb{B}, e$)
**end**
```

Algorithm 2: Pivot($\mathbb{B}, e$)

```
Data: Lex-feasible basis $\mathbb{B}$, non-basic column $e \in \mathbb{H}$
if $[r^T_{\{B \cup e\}} c_{\{B \cup e\}}, r^T_{\{B \cup e\}}] \prec 0$ then
    select leaving column $\ell_{lex}(e)$ via lex ratio test
    if $\ell_{lex}(e) \neq \emptyset$ then
        $\mathbb{B} \leftarrow (\mathbb{B} \cup e)\ \ell_{lex}(e))$ ;                // make the pivot
    else
        $\mathbb{B} \leftarrow$ null ;                              // problem is unbounded
end
```

The procedure of replacing a basic column with a non-basic one is called *pivot*, and in order to handle degeneracy a variant of the simplex is given by taking,

$$[r^T_{\{B \cup e\}} c_{\{B \cup e\}}, r^T_{\{B \cup e\}}] \tag{1.1}$$

with reduced cost defined as:

$$\bar{c}_e = c_e - A_e^T (A_B^{-1})^T c_b =: r^T_{\{B \cup e\}} c_{\{B \cup e\}} \tag{1.2}$$

where $c_e \in \mathbb{R}$ and $A_e \in \mathbb{R}^d$ refer to the problem data related to column e. Starting with a lex-feasible basis, the simplex method is defined such that after a pivot iteration the new basis is again lex-feasible. Let $\mathbb{B}$ be a lex-feasible basis and e the entering column, let the leaving column be chosen by the *lexicographic ration test*,

$$\ell_{lex}(e) = arg \quad lexmin[A_B^{-1}b, A_B^{-1}]_{\bullet j}/(A_B^{-1}A_e)_{\bullet j} \quad | \quad (A_B^{-1}A_e)_{\bullet j} > 0 \tag{1.3}$$

where the subscript $\bullet j$ denotes selection of the j-th row of the matrix, respectively vector, then the next basis is again lex-feasible, as described in [1].

Chapter 2

Code explanation

The implementation of distributed simplex is done by using robot operating system, and in order to establish communication between agents, it is defined a ring digraph, shown in figure 2.1. So, the i-th agent receives the basis from the (i-1)-th agent and sends its basis to the (i+1)-th agent, except for the first and last agent.

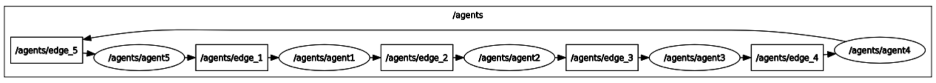


Figure 2.1: Ring graph

2.1 Agent class

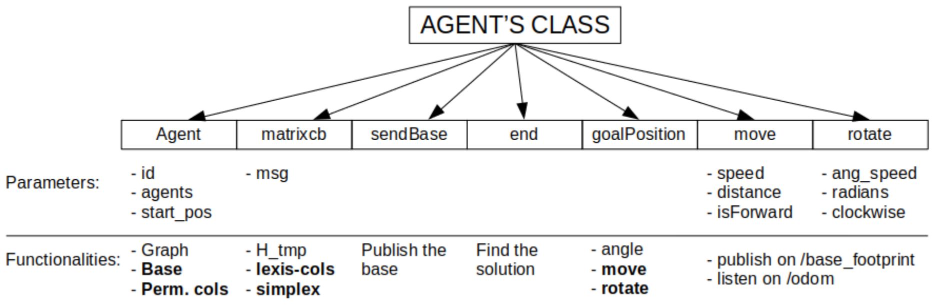


Figure 2.2: Agent's class

Let's start looking to the main program, **main.cpp**, at beginnig it receives an identification number, used to distinguish between nodes, and then it is created an object of *Agent's* class. Initialization of Agent's class consist of a basis matrix $\mathbb{B}$, permanent columns matrix $\mathbb{P}$, and the matrix $\mathbb{H}$, a publisher and a subscriber with the id number given at node, in order to create the ring digraph. At last element, a publisher is used to send velocity's comand to robot basis, and move the robot to the desired target position. Let's consider a case with three agents, so matrices of the first agent, will be as follows:

$$\mathbb{B} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ M & M & M & M & M & M \end{pmatrix} \mathbb{P}_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} \end{pmatrix} \tag{2.1}$$

$$\mathbb{H}_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} & M & M & M & M & M & M \end{pmatrix} \tag{2.2}$$

Where the cost vector $[c_{11}, c_{12}, c_{13}]$ is founded by estimate distance from each targets. The simplex algorithm is implemented inside **matrixcb** function, which starts only when receives a basis message from neighbours, and by looking at the first iteration, the basis its updated only with its permanent columns:

$$\mathbb{B}_1 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ c_{12} & M & M & c_{11} & M & M \end{pmatrix} \tag{2.3}$$

Then, after first iteration, agents receive an updated basis with permanent columns, and they need to perform a lexicographic ordering of them. The function **matrixcb** starts by creating the $\mathbb{H}_{tmp}$ matrix with columns received in a basis message from its neighbour and its permanent columns, so the function lexicographic sorting all the columns of the matrix $\mathbb{H}_{tmp}$, to provides a unique ordering of them. The lexicographic ordering ensures that all agents optimize with respect to the same lexicographic objective.

$$\mathbb{H}_{tmp_1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & \dots \\ c_{11} & c_{11} & c_{12} & c_{12} & c_{13} & M & c_{21} & c_{23} & M & \dots \end{pmatrix} \tag{2.4}$$

Then the Simplex algorithm is defined with two parameters $\mathbb{H}_{tmp}$ and $\mathbb{B}$, inside the file **Simplex.cpp**. At beginning it initializes all matrices to zero, and enters inside the while cycle until all columns of $\mathbb{H}_{tmp}$ are checked. If vector e is already inside the basis $\mathbb{B}$, is skipped, else it used to check the reduced cost as described previously. If the reduced cost is lexico-negative, a leaving column is choosen by lexicographic radio test to be changed with vector e. The basis obtained by simplex function, is stored to a temporary basis $\mathbb{B}_{tmp}$ and it is checked with the previous one $\mathbb{B}$.

Algorithm 3: matrixcb($\mathbb{B}_{msg}$)

```
Data: A set of columns received from neighbours $\mathbb{B}_{msg}$, a
      lex-feasible basis $\mathbb{B}$ and permanet columns $\mathbb{P}$
for cols($\mathbb{H}_{tmp}$) do
    $\mathbb{H}_{tmp} \leftarrow \mathbb{B}_{msg}$
end
$\mathbb{H}_{tmp} \leftarrow \mathbb{P}$
$\mathbb{B}_{tmp} \leftarrow$ Simplex($\mathbb{H}_{temp}$, $\mathbb{B}$)
if $\mathbb{B}_{tmp} == \mathbb{B}$ then
    count++;
    if count > artificial_parameters then
        solution $\leftarrow$ end($\mathbb{B}$)
end
$\mathbb{B} \leftarrow \mathbb{B}_{tmp}$
```

If the basis sended by the previous agent is changed, then all the agents are connected and I can start to converge to the common solution. After a while, if the basis is not more changing then **matrixcb** function will stop to execute, and the **end** function will be used to define the solution. Inside the common basis, it will be present one of the permanent columns of matrix $\mathbb{P}$, by looking only to the columns related to our agent, so from the top, the columns with the ones to the same row as our agent id.

Algorithm 4: end($\mathbb{B}$)

```
Data: Base matrix $\mathbb{B}$, permanet columns $\mathbb{P}$ and the agent's id
$\mathbb{B} \leftarrow$ lex-order($\mathbb{B}$)
for $i$ = cols($\mathbb{B}$,id) do
    for $j$ = cols($\mathbb{P}$) do
        if $\mathbb{B}_i == \mathbb{P}_j$ then
            solution $\leftarrow \mathbb{B}_i$
        end
    end
end
```

$$\mathbb{B}_1 = \mathbb{B}_2 = \mathbb{B}_3 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ c_{12} & M & c_{21} & M & c_{33} & M \end{pmatrix} \tag{2.5}$$

$$\textbf{solution} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ c_{12} \end{pmatrix} \tag{2.6}$$

This solution is related to the target, c_{12} is the distance from the second target, and by using it inside the function **goalPosition**, the robot sets the distance to cover and the angle to turn, then it sends these parameters to functions **rotate** and **move**.

Chapter 3

Simulations

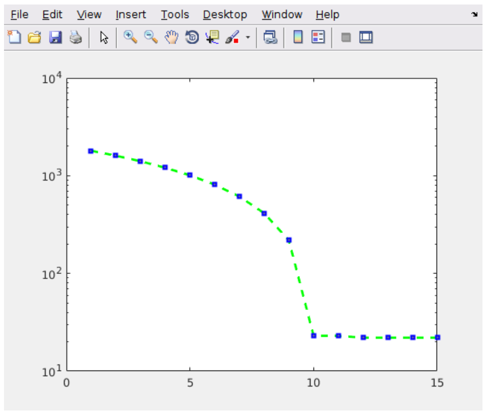
Several results are provided in this section, in order to show the convergency property of the algorithm and then, some simulations to better understand the structure of the simplex and how it works. Firstly, a graph explanation of the decreasing cost, associated to the basis, given at each iteraction. Secondly, a representation of the terminal given in order to show the initialization phase, the common basis and the choosen solutions. At the end, a video simulation with three and four agents in gazebo, by using TurtleBot robots.

3.1 Base cost

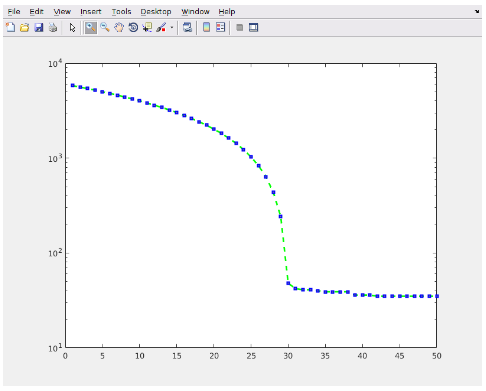
Matlab, is the instrument used to computing the cost associated to the basis, as in (1.2), and it is usefull to show results with graphics. So at each iteraction, the cost of the basis is given by:

$$c^{[i]} = b^T ({\mathbb{B}_{\mathbb{A}}}^{-1})^T c_{\mathbb{B}} \tag{3.1}$$

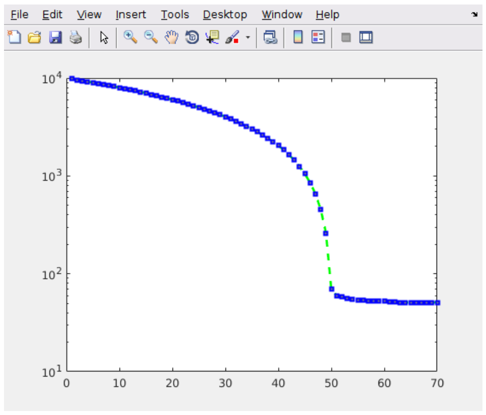
where b is a vector of ones, $\mathbb{B}_{\mathbb{A}}$ is the part of the basis matrix related to the constraints, $c_{\mathbb{B}}$ is the row of the basis associated to costs and i is the iteration number. Simulations, in figure 3.1, show that the cost associated to the basis is decreasing at each iteration, and the maximum number of iterations needed is greater or equal to the number of agents. It is used a ring graph and it is plotted the cost of the first agent in images 3.1a, 3.1b and 3.1c, where number of agents are 10, 30 and 50 respectively. In the y-axis the cost computed, as in (3.1), is plotted in logarithmic scale, where the value of M is equal to 100, and in x-axis is plotted the number of iterations. In the last image 3.1d, it is shown the cost of last iterations, with 50 agents, until a common solution is reached, and from then on the basis is not more changing.



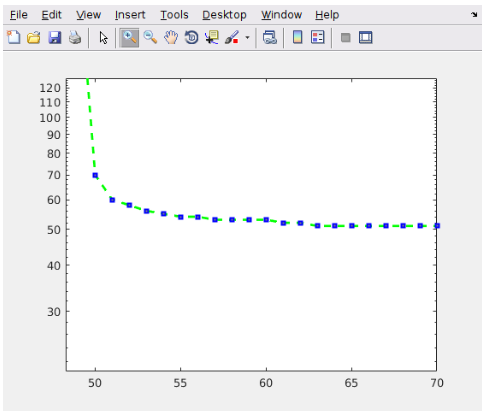
(a) 10 agents



(b) 30 agents

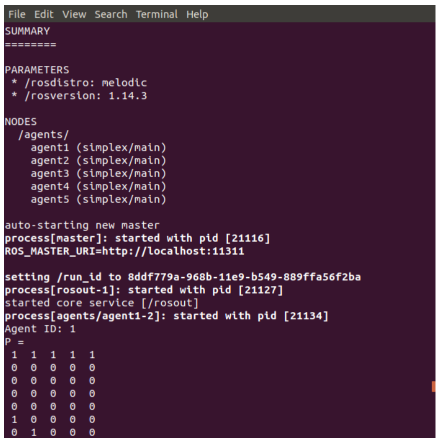


(c) 50 agents



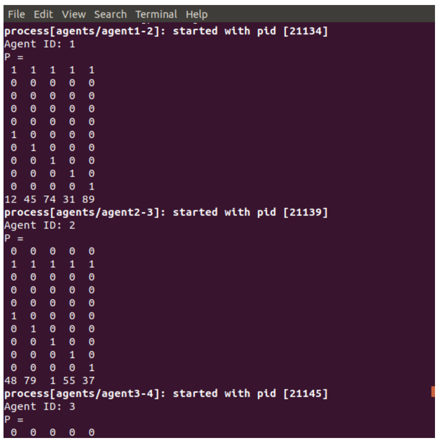
(d) 50 agents

Figure 3.1: The decreasing cost associated to the base by Matlab simulations



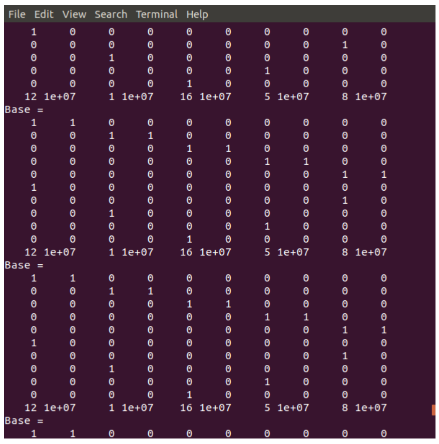
```
SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES
  /agents/
    agent1 (simplex/main)
    agent2 (simplex/main)
    agent3 (simplex/main)
    agent4 (simplex/main)
    agent5 (simplex/main)

auto-starting new master
process[master]: started with pid [21116]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 8ddf779a-968b-11e9-b549-889ffa56f2ba
process[rosout-1]: started with pid [21127]
started core service [/rosout]
process[agents/agent1-2]: started with pid [21134]
Agent ID: 1
P =
 1  1  1  1  1
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 1  0  0  0  0
 0  1  0  0  0
```

(a) Agents start



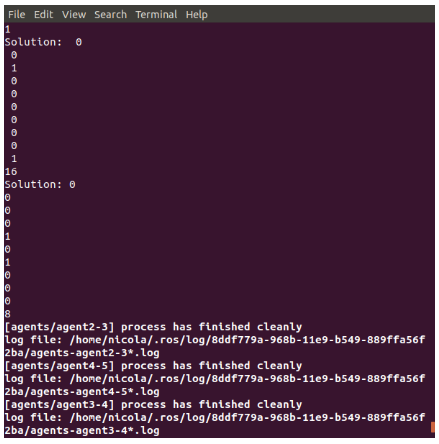
```
process[agents/agent1-2]: started with pid [21134]
Agent ID: 1
P =
 1  1  1  1  1
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 1  0  0  0  0
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 0  0  0  0  1
12 45 74 31 89
process[agents/agent2-3]: started with pid [21139]
Agent ID: 2
P =
 0  0  0  0  0
 1  1  1  1  1
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
 1  0  0  0  0
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 0  0  0  0  1
48 79  1 55 37
process[agents/agent3-4]: started with pid [21145]
Agent ID: 3
P =
 0  0  0  0  0
```

(b) Permanent columns



```
    1      0      0      0      0      0      0      0      0      0
    0      0      0      0      0      0      0      0      1      0
    0      0      1      0      0      0      0      0      0      0
    0      0      0      0      0      0      1      0      0      0
    0      0      0      0      1      0      0      0      0      0
   12  1e+07      1  1e+07     16  1e+07      5  1e+07      8  1e+07
Base =
    1      1      0      0      0      0      0      0      0      0
    0      0      1      1      0      0      0      0      0      0
    0      0      0      0      1      1      0      0      0      0
    0      0      0      0      0      0      1      1      0      0
    0      0      0      0      0      0      0      0      1      1
    1      0      0      0      0      0      0      0      0      0
    0      0      0      0      0      0      0      0      1      0
    0      0      1      0      0      0      0      0      0      0
    0      0      0      0      0      0      1      0      0      0
    0      0      0      0      1      0      0      0      0      0
   12  1e+07      1  1e+07     16  1e+07      5  1e+07      8  1e+07
Base =
    1      1      0      0      0      0      0      0      0      0
    0      0      1      1      0      0      0      0      0      0
    0      0      0      0      1      1      0      0      0      0
    0      0      0      0      0      0      1      1      0      0
    0      0      0      0      0      0      0      0      1      1
    1      0      0      0      0      0      0      0      0      0
    0      0      0      0      0      0      0      0      1      0
    0      0      1      0      0      0      0      0      0      0
    0      0      0      0      0      0      1      0      0      0
    0      0      0      0      1      0      0      0      0      0
   12  1e+07      1  1e+07     16  1e+07      5  1e+07      8  1e+07
Base =
    1      1      0      0      0      0      0      0      0      0
```

(c) Common Base



```
1
Solution:  0
 0
 1
 0
 0
 0
 0
 0
 0
 1
16
Solution: 0
0
0
0
1
0
1
0
0
0
8
[agents/agent2-3] process has finished cleanly
log file: /home/nicola/.ros/log/8ddf779a-968b-11e9-b549-889ffa56f
2ba/agents-agent2-3*.log
[agents/agent4-5] process has finished cleanly
log file: /home/nicola/.ros/log/8ddf779a-968b-11e9-b549-889ffa56f
2ba/agents-agent4-5*.log
[agents/agent3-4] process has finished cleanly
log file: /home/nicola/.ros/log/8ddf779a-968b-11e9-b549-889ffa56f
2ba/agents-agent3-4*.log
```

(d) Solution of one agent

Figure 3.2: The simplex algorithm is implemented in ROS with 5 agents

3.2 Terminal

Here, it is shown a simulation of five agents sharing the basis by using the robot operating system. From a launch file, they are spawned, as shown in 3.2a, with a ring communication graph, after an initialization of the columns 3.2b, they start to share the basis and at the end, they reach the common base as shown in figure 3.2c. By using function **end**, every agent is able to choose the column given as solution, and to print it 3.2d.

3.3 Gazebo

Gazebo is the simulation environment used to test the algorithm with a fleet of wheeled-robots. The simulations are given with a maximum of 4 agents, and it is used TurtleBot as wheeled mobile robot. The generation of the constrained vector is given by calcute distance between the initial robot position and target positions. Where targets are positionated at each corner of arena, so at positions (10,10),(-10,10),(10,-10) and (-10,-10). After launching robots inside gazebo, it is runned the simplex in order to establish the communication, as shown in figure 3.4, and to select the target for each of them. The function **goalPosition** uses solution provides from **end** function, and sets the distance to cover and the angle to turn, then it sends these parameters to functions **rotate** and **move**.

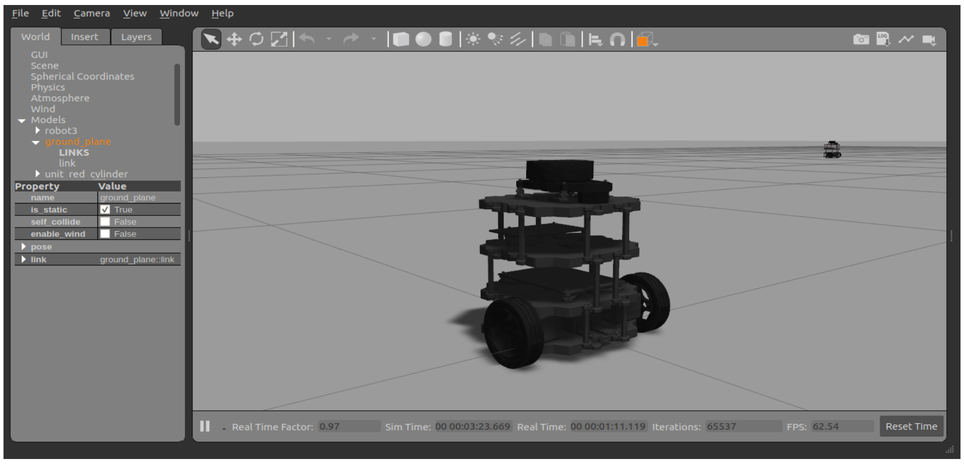


Figure 3.3: Turtlebot

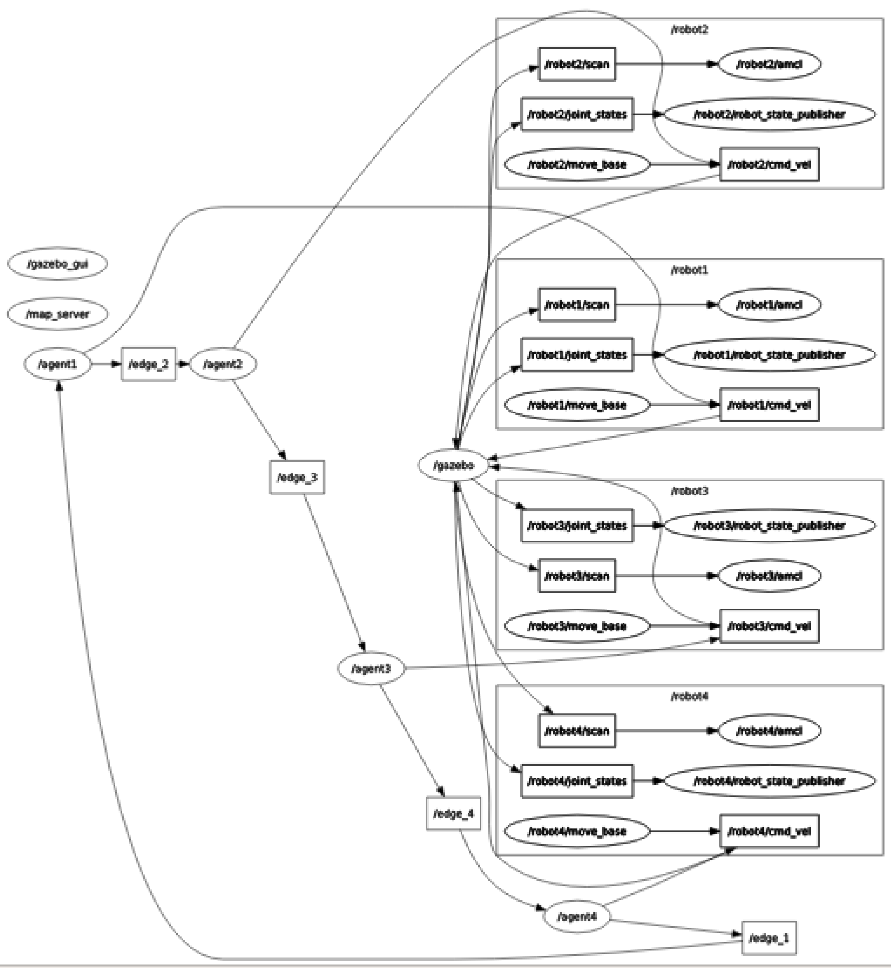


Figure 3.4: Gazebo graph

Bibliography

[1] Mathias Bürger, Giuseppe Notarstefano, Francesco Bullo and Frank Allgöwer. *A distributed simplex algorithm for degenerate linear programs and multi-agent assignments.* Automatica, 2012.