
Coordenação, Localização e Navegação para Robôs de Serviço em Ambientes Internos.

Raulcézar Maximiano Figueira Alves



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Raulcésar Maximiano Figueira Alves

**Coordenação, Localização e Navegação para
Robôs de Serviço em Ambientes Internos.**

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Carlos Roberto Lopes

Uberlândia

2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

A474c Alves, Raulcézar Maximiano Figueira, 1984-
2017 Coordenação, localização e navegação para robôs de serviço em
ambientes internos / Raulcézar Maximiano Figueira Alves. - 2017.
158 f.

Orientador: Carlos Roberto Lopes.
Tese (doutorado) -- Universidade Federal de Uberlândia, Programa
de Pós-Graduação em Ciência da Computação.
Disponível em: <http://dx.doi.org/10.14393/ufu.te.2017.21>
Inclui bibliografia.

1. Computação - Teses. 2. Robôs autônomos - Teses. 3. Inteligência
artificial - Teses. I. Lopes, Carlos Roberto. II. Universidade Federal de
Uberlândia. Programa de Pós-Graduação em Ciência da Computação.
III. Título.

CDU: 681.3

A minha família e amigos.

Agradecimentos

Agradeço...

A Deus, por esta vida.

Aos meus pais Vagnete Fagundes Ferreira e Zoraene Carmen Figueira Fagundes pela dedicação, confiança, apoio, carinho e amor incondicional em todos os momentos.

Ao meu irmão Victor Figueira Fagundes pela parceria, carinho e amor.

A minha irmã Jozaene Maximiano Figueira Alves Faria, meu cunhado Fernando Faria Lima, e minhas afilhadas Mariana Alves Faria e Fernanda Alves Faria pela alegria, serenidade, e exemplo de família.

A minha esposa Cristiane Soares Campos por ter me apoiado durante mais esta jornada. Você me deu muita força, alegria e amor.

A todos da minha família que sempre me apoiaram e acreditaram em mim.

Aos meus amigos do Programa de Pós-Graduação em Ciência da Computação da UFU por terem feito dessa caminhada algo mais descontraído e prazeroso.

Aos amigos do grupo CORAL de Inteligência Artificial aplicado em Robótica da Carnegie Mellon University por terem me mostrado este maravilhoso mundo novo.

Aos meus amigos do Centro de Tecnologia da Informação da UFU pelo incentivo e amizade ao longo desta etapa da minha vida.

A professora Manuela Veloso por ter me recebido na Carnegie Mellon University durante meu período sanduíche. Sem ela, esta tese não seria possível.

Ao professor Carlos Roberto Lopes pelo profissionalismo, apoio, paciência, amizade e sobretudo por me guiar durante minhas buscas.

“A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê.” (Arthur Schopenhauer)

Resumo

A Robótica tem iniciado uma transição de Robótica Industrial para Robótica de Serviço, movendo-se em direção as necessidades diárias dos seres humanos. Para realizar essa transição, robôs necessitam de mais autonomia para executar tarefas em espaços dinâmicos ocupados por humanos, diferente dos ambientes controlados das fábricas.

Nesta tese, é investigado um problema no qual um time de robôs completamente autônomos deve visitar certos locais em um ambiente interno usado por humanos a fim de executar algum tipo de tarefa. Este problema está relacionado a três importantes questões da Robótica e Inteligência Artificial (IA), que são: coordenação, localização e navegação.

Para coordenar as visitas nos locais desejados, um escalonamento deve ser realizado para encontrar as rotas para os robôs. Tal escalonamento deve minimizar a distância total viajada pelo time e também balancear as rotas. Este problema pode ser modelado como sendo uma instância do Problema dos Múltiplos Caixeiros Viajantes (PMCV). Como este problema é classificado como NP-Difícil, é proposto o uso de algoritmos aproximados para encontrar soluções satisfatórias para o problema.

Uma vez que as rotas estão computadas, os robôs necessitam de se localizar no ambiente para que eles tenham certeza de que estão visitando os lugares corretos. Muitas técnicas de localização não são muito precisas em ambientes internos devido a diferentes tipos de ruídos. Desta forma, é proposto uma combinação de duas delas. Nesta abordagem, um algoritmo de localização WiFi rastreia a localização global do robô, enquanto um algoritmo de localização Kinect estima sua posição atual dentro da área delimitada pela localização global.

Depois de visitar um dado local de sua rota, o robô deve navegar em direção ao próximo. A navegação em ambientes internos ocupados por humanos é uma tarefa difícil, uma vez que muitos objetos móveis e dinâmicos podem ser encontrados no caminho. Para isso, o robô deve possuir controles reativos para evitar colidir com objetos dinâmicos, como pessoas, enquanto ele navega. Além disso, objetos móveis, como mobílias, são passíveis de serem movidos frequentemente, o que muda o mapa utilizado para planejar o caminho

do robô. Para resolver estes problemas, é proposto um algoritmo de desvio de obstáculos e um planejador dinâmico de caminho para ambientes internos ocupados por humanos.

Desta forma, esta tese contribui com uma série de algoritmos para os problemas de coordenação, localização e navegação. São introduzidos: Algoritmos Genéticos (AGs) multi-objetivo para resolver o Problema dos Múltiplos Caixeiros Viajantes, abordagens de localização que utilizam a técnica de Filtro de Partículas (FP) com dispositivos Kinect e WiFi, um Sistema Híbrido Inteligente (SHI) baseado em Lógica Fuzzy (LF) e Redes Neurais Artificiais (RNA) para desvio de obstáculos e uma adaptação do algoritmo D*Lite que permite o robô replanejar caminhos de forma eficiente e requisitar auxílio humano se necessário.

Todos os algoritmos são avaliados em robôs reais e simuladores, demonstrando seus desempenhos em resolver os problemas abordados nesta tese.

Palavras-chave: Robôs Autônomos, Inteligência Artificial, Coordenação, Localização, Navegação, Desvio de Obstáculos, Planejamento de Caminho, Ambientes Internos.

Coordination, Localization, and Navigation for Service Robots in Indoor Environments.

Raulcézar Maximiano Figueira Alves



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uberlândia
2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA – UFU
FACULDADE DE COMPUTAÇÃO – FACOM
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO – PPGCO

The undersigned hereby certify they have read and recommend to the PPGCO for acceptance the Proposal entitled **“Coordination, Localization, and Navigation for Service Robots in Indoor Environments.”** submitted by **“Raulcézar Maximiano Figueira Alves”** as part of the requirements for obtaining the **Doctor’s degree in Computer Science.**

Uberlândia, ___ de _____ de ____

Supervisor: _____

Prof. Dr. Carlos Roberto Lopes
Universidade Federal de Uberlândia

Examining Committee Members:

Prof. Dr. Flavio Tonidandel
Centro Universitário FEI-SP

Prof. Dr. Guilherme A. S. Pereira
Universidade Federal de Minas Gerais

Prof. Dr. Rogério Sales Gonçalves
Universidade Federal de Uberlândia

Prof. Dr. Jefferson Rodrigo de Souza
Universidade Federal de Uberlândia

Abstract

Robotics has started the transition from industrial into service robotics, moving closer towards humans daily needs. To accomplish this transition, robots require more autonomy to perform tasks in dynamic spaces occupied by humans, different from well controlled environments of factory floors.

In this thesis, we investigate a problem in which a team of completely autonomous robots needs to visit certain locations in an indoor human environment in order to perform some kind of task. This problem is related to three important issues of Robotics and Artificial Intelligence (AI), namely: coordination, localization and navigation.

To coordinate the visits in the desired locations, a scheduling must be performed to find routes for the robots. Such scheduling needs to minimize the total distance traveled by the team and also to balance the routes. We model this problem as being an instance of the multiple Traveling Salesmen Problem (mTSP). Since it is classified as NP-Hard, we propose the use of approximation algorithms to find reasonable solutions to the problem.

Once the routes are computed, the robots need to localize themselves in the environment so they can be sure that they are visiting the right places. Many localization techniques are not very accurate in indoor human environments due to different types of noise. Therefore, we propose the combination of two of them. In such approach, a WiFi localization algorithm tracks the global location of the robot while a Kinect localization algorithm estimates its current pose on that area.

After visiting a given location of its route, the robot must navigate towards the next one. Navigation in indoor human environments is a challenging task as many moving and movable objects can be found in the way. The robot should be equipped with a reactive controller to avoid colliding with moving objects, like people, while it is navigating. Also, movable objects, such as furniture, are likely to be moved frequently, which changes the map used to plan the robot's path. To tackle these problems, we introduce an obstacle avoidance algorithm and a dynamic path planner for navigation in indoor human environments.

We contribute a series of algorithms for the problems of coordination, localization, and

navigation. We introduce: multi-objective Genetic Algorithms (GAs) to solve the mTSP, localization approaches that use Particle Filters (PFs) with Kinect and WiFi devices, a Hybrid Intelligent System (HIS) based on Fuzzy Logic (FL) and Artificial Neural Network (ANN) for obstacle avoidance, and an adaptation to the D*Lite algorithm that enables robots to replan paths efficiently and also ask for human assistance if it is necessary.

All algorithms are evaluated on real robots and simulators, demonstrating their performances to solve the problems addressed in this thesis.

Keywords: Autonomous Robots, Artificial Intelligence, Coordination, Localization, Navigation, Obstacle Avoidance, Path Planning, Indoor Human Environments.

List of Figures

Figure 1 – <i>Coordination</i> : route scheduling. This figure illustrates the scenario where the routes (colored lines) of 3 robots (black dots) cover all desired locations (red crosses) of the environment.	19
Figure 2 – <i>Localization</i> : estimated location based on Kinect and WiFi observations.	20
Figure 3 – <i>Navigation</i> : obstacle avoidance and dynamic path planning with human assistance. This picture illustrates a scenario where a cabinet is blocking the current path of the robot (in blue). The robot computes another path (in green), but it chooses to ask for human assistance (red circle) instead.	21
Figure 4 – Robots used in the experiments.	22
Figure 5 – Fuzzy Controller.	31
Figure 6 – Perceptron.	31
Figure 7 – Multilayer Perceptron.	32
Figure 8 – Finding inconsistencies.	36
Figure 9 – Genetic Algorithm Structure.	40
Figure 10 – Shakey (CHAITIN et al., 1970)	41
Figure 11 – Flakey (SAFFIOTTI; RUSPINI; KONOLIGE, 1993)	41
Figure 12 – Rhino (BUHMANN et al., 1995a)	42
Figure 13 – Minerva (THRUN et al., 1999)	43
Figure 14 – Xavier (KOENIG; SIMMONS, 1998)	43
Figure 15 – PR2 (OYAMA et al., 2009)	44
Figure 16 – Roomba (ACKERMAN, 2015)	44
Figure 17 – Fetch & Freight (WISE et al., 2016)	45
Figure 18 – Care-O-bot (REISER et al., 2009)	45
Figure 19 – Relay (PRANSKY, 2016)	46
Figure 20 – Kiva (WURMAN; D’ANDREA; MOUNTZ, 2007)	46
Figure 21 – OSHbot (OSHBOT, 2017)	47
Figure 22 – Tech-i (TECHI, 2017)	48












Figure 23 – CoBots (VELOSO et al., 2015a)	48
Figure 24 – Overview of indoor technologies in dependence on accuracy and cover- age (MAUTZ, 2012).	52
Figure 25 – Turtlebot 2: robotic platform used in our indoor localization experi- ments with Kinect and WiFi devices.	56
Figure 26 – Kinect sensor	57
Figure 27 – Mapping the Laboratory with Turtlebot using Kinect sensor. 	59
Figure 28 – Laser Map of the Laboratory.	59
Figure 29 – Floor Plan of the 3 rd floor in the Gates and Hillman Centers building at Carnegie Mellon University.	60
Figure 30 – AMCL process.	61
Figure 31 – Kinect localization on RViz interface.	63
Figure 32 – WiFi emitter and receiver devices.	63
Figure 33 – Irregular Grid for WiFi Mapping.	66
Figure 34 – Estimating Mean and Standard Deviation Values for RSS on the WiFi Grid.	68
Figure 35 – Perceptual Model: Calculating Join Probabilities.	71
Figure 36 – WiFi Heat Map: Laboratory.	72
Figure 37 – Example - Kinect Localization.	74
Figure 38 – Example - WiFi Localization.	75
Figure 39 – Example - Kinect+WiFi Localization. 	76
Figure 40 – Ground Truth Path.	77
Figure 41 – Trial example on the path. 	78
Figure 42 – Localization error along the path.	79
Figure 43 – Mean error of all approaches.	79
Figure 44 – Robot Kidnapping Experiment. 	80
Figure 45 – Recovering Time.	80
Figure 46 – Accuracy of WiFi localization: beginning of the path.	81
Figure 47 – Accuracy of WiFi localization: end of the path. 	82
Figure 48 – Localization error vs. Number of Access Points.	82
Figure 49 – <i>E-Puck</i> devices	91
Figure 50 – HIS controller	92
Figure 51 – Example of sample generation for the Fuzzy Module	93
Figure 52 – Example of sample generation for the MLP Module	95
Figure 53 – MLP controller	96
Figure 54 – Test environments. 	97
Figure 55 – Average collision number	98
Figure 56 – Replanning Deviations in Obstructed Paths.	103
Figure 57 – Graphical User Interface (GUI) 	104

Figure 58 – Replanning: OPEN list	105
Figure 59 – Simulation with Turtlebot, ROS, Gazebo, and RViz 	106
Figure 60 – Path length	107
Figure 61 – Corridor: 2nd floor of Building-B at UFU 	107
Figure 62 – Willow Garage: virtual environment $\approx 53\text{m} \times 42\text{m}$ 	108
Figure 63 – Taxonomy of Approaches.	112
Figure 64 – Example of an individual.	117
Figure 65 – Inflation. (available online on ROS Costmap: Inflation)	122
Figure 66 – Indoor Robot Map: Laboratory.	123
Figure 67 – Indoor Robot Map: Willow Garage Inc.	123
Figure 68 – Path Matrix.	124
Figure 69 – Control Panel.	125
Figure 70 – Select a Map.	125
Figure 71 – Set Depot.	126
Figure 72 – Set Locations.	126
Figure 73 – Find Paths.	127
Figure 74 – Route Scheduling.	127
Figure 75 – Route scheduling for the mTSP with multi-objective GA.	128
Figure 76 – Problem 01: solution sets given by the best combination of each method.130	
Figure 77 – Problem 02: solution sets given by the best combination of each method.132	
Figure 78 – Problem 03: solution sets given by the best combination of each method.132	
Figure 79 – Problem 04: solution sets given by the best combination of each method.135	
Figure 80 – Problem 05: solution sets given by the best combination of each method.135	
Figure 81 – Route Scheduling System: Laboratory.	137
Figure 82 – Route Scheduling System: Willow Garage Inc. 	138

List of Tables

Table 1 – Comparison of main indoor positioning technologies. [UC] End User Cost, [IC] Installation and Maintenance Cost, [H] High, [L] Low, [ML] Multiple Levels (BRENA et al., 2017).	50
Table 2 – Advantages and disadvantages of the filters techniques.	54
Table 3 – Mean Matrix (M).	65
Table 4 – Standard Deviation Matrix (D).	65
Table 5 – Navigation Time	108
Table 6 – Comparison of main centralized and distributed approaches.	112
Table 7 – Strategies and Algorithms for mTSP (BEKTAS, 2006).	114
Table 8 – Problem 01 - 2 salesmen and 10 cities	129
Table 9 – Problem 02 - 3 salesmen and 20 cities	131
Table 10 – Problem 03 - 5 salesmen and 30 cities	133
Table 11 – Problem 04 - 8 salesmen and 50 cities	134
Table 12 – Problem 05 - 13 salesmen and 80 cities	136
Table 13 – Route Scheduling System: Laboratory.	137
Table 14 – Route Scheduling System: Willow Garage Inc.	138

Acronyms list

AI Artificial Intelligence

AMCL Adaptive Monte Carlo Localization

ANN Artificial Neural Network

APs Access Points

EKF Extended Kalman Filter

FL Fuzzy Logic

GAs Genetic Algorithms

GPS Global Positioning System

HIS Hybrid Intelligent System

MLP Multiple Layer Perceptron

mTSP multiple Traveling Salesmen Problem

PF Particle Filter

RBPF Rao-Blackwellized Particle Filter

ROS Robot Operating System

RSS Received Signal Strength

SLAM Simultaneous Localization and Mapping

STRIPS STanford Research Institute Problem Solver

TSP Traveling Salesman Problem

Contents

1	INTRODUCTION	17
1.1	Motivation	18
1.2	Approach	19
1.3	Contributions	22
1.4	Thesis Organization	26
2	BACKGROUND	27
2.1	Localization	27
2.1.1	Bayes Filter	27
2.1.2	Particle Filter	28
2.2	Navigation	29
2.2.1	Obstacle Avoidance	30
2.2.2	Path Planning	33
2.3	Coordination	37
2.3.1	multiple Traveling Salesmen Problem	37
2.3.2	Genetic Algorithm	39
2.4	Autonomous Robots in Real Indoor Human Environments . . .	41
3	LOCALIZATION	49
3.1	Kinect Localization	57
3.1.1	Related Work	57
3.1.2	Mapping	58
3.1.3	Localization Algorithm	60
3.2	WiFi Localization	63
3.2.1	Related Work	64
3.2.2	Mapping	65
3.2.3	Localization Algorithm	67
3.3	Kinect+WiFi Localization	74

3.4	Experiments	77
3.5	Summary	83
4	NAVIGATION	85
4.1	Obstacle Avoidance	88
4.1.1	Related Work	89
4.1.2	HIS controller	90
4.1.3	MLP controller	96
4.1.4	Experiments	97
4.2	Path Planning	98
4.2.1	Related Work	99
4.2.2	D*Lite with Human Assistance	101
4.2.3	Experiments	103
4.3	Summary	108
5	COORDINATION	111
5.1	Related Work	115
5.2	Using GAs to solve the mTSP	116
5.2.1	Individual Structure	117
5.2.2	Crossover & Mutation	117
5.2.3	Objectives	118
5.2.4	mono-objective GA	118
5.2.5	multi-objective GA	119
5.3	Route Scheduling for a Team of Robots in Indoor Environments	121
5.3.1	Indoor Robot Maps	121
5.3.2	Path Matrix	123
5.3.3	Route Scheduling System	125
5.4	Experiments	128
5.5	Summary	138
6	CONCLUSION AND FUTURE WORK	141
	BIBLIOGRAPHY	145

I hereby certify that I have obtained all legal permissions from the owner(s) of each third-party copyrighted matter included in my thesis, and that their permissions allow availability such as being deposited in public digital libraries.

Raulcésar Maximiano Figueira Alves

CHAPTER 1

Introduction

Although robots already play a very important role in the world today, most of them are used for industrial automation applications. However, many researches aim to develop robots for assisting humans in their everyday tasks. These robots should perform tasks that are dirty, dull, distant, dangerous, or repetitive for humans. They can be categorized as: professional service providers (working in offices, hospitals, etc), domestic service robots (for entertainment, education, personal works at home), security robots (defense, safety, and rescue), and space robots (planet exploration) (TERESA, 2014).

This thesis address a problem in which a team of autonomous robots has to visit some specific locations in an indoor human environment to perform some task, such as delivering mail, picking up trash, finding objects, patrolling, etc. In order to perform any of these tasks properly, these robots must be able to *coordinate*, *localize*, and *navigate* themselves in the environment.

In this problem, all robots remain together at the same place in the environment, as a charging station. When they receive a list of locations, the robots must visit all of them, and then return to the origin. Each location must be visited only once by one of the robots. Thus, to *coordinate* the visits, a scheduling has to be done to find routes for the robots in order to cover all locations. To optimize the visits, the scheduling needs to minimize the total distance traveled by the team. But, for some instances, the scheduling can generate routes in which a robot travels much more than others. In such a case, this specific robot can run out of battery before finishing its route. Therefore, the scheduling also needs to take into account the balance of the distance traveled by all robots.

Another important aspect in this scenario is the *localization*. Once the routes are computed, the robots need to follow them to visit all locations. However, they need to *localize* themselves in real-time so they can be sure that they are visiting the right places. There is a lot of difficulties regarding *localization* for indoor human environments. For instance, outdoor techniques, such as Global Positioning System (GPS), are not applied as satellite signals cannot come across buildings. Depth cameras and laser range finders are commonly used in robots localization, but they can be occluded by furniture and

humans. Techniques based on WiFi signal strength are also used for indoor localization, however, the minimal error is greater than that of other approaches.

Furthermore, after visiting a given location, a robot must *navigate* towards the next location of its route. The problem is, indoor human environments are composed by many movable objects such as tables and chairs, and moving objects like people. So, to *navigate* safely in the environment, robots need a reactive controller to avoid hitting these objects. Also, when an object blocks their paths, they have to replan their trajectories efficiently to reach the desired location. But, sometimes the new path can be much longer than the original. In this case, robots can ask for help. For example, they can ask a human to move away or move a chair. This can be seen as a symbiotic behavior since robots are performing tasks for humans, and humans are helping robots to complete their tasks.

Thus, the contributions of this thesis for the problems of *coordination*, *localization*, and *navigation* for autonomous robots in indoor environments are the following. First, a system composed by multi-objective algorithms to schedule routes for a team of robots. These algorithms treat the tradeoff between minimizing the total distance traveled by the team and balancing the routes of the robots. Second, a localization technique that uses Kinect and WiFi devices. The idea is to put together the benefits of the global localization given by WiFi with the local precision of Kinect localization. Finally, an hybrid intelligent system for obstacle avoidance, and a path planning algorithm that replans efficiently if the robot is blocked and also asks for human assistance.

1.1 Motivation

This thesis seeks to answer the question,

How can a team of completely autonomous robots can perform tasks at specific locations in an indoor human environment considering the challenges of *coordination*, *localization*, and *navigation* in such complex, noisy, and dynamic place?

Eventually, we will be able to have robots helping us in our daily tasks at home or in the office. But to make it possible, we must solve some problems inherent to human environments. For instance, indoor environments are very complex. They have rooms, walls, corridors, corners, and doors that must be taken into account during a route scheduling or a path planning. These environments change over time as they are composed by movable objects used by humans, thus affecting the accuracy of some visual localization techniques. Some localization approaches based on WiFi signals can be affected by the use of internet from human devices, which increases the localization error. Movable objects and humans are also a problem for navigation, requiring dynamic algorithms to compute trajectories and reactive controllers to avoid obstacles.

1.2 Approach

We introduce different types of Artificial Intelligence algorithms to tackle the problems addressed in the thesis question.

In the *coordination* problem, we have a team of robots in an indoor human environment. Initially, they are located all together in a charging station area. Eventually, this team receives a list of locations to be visited. Once the visits are done, all robots must come back to the charging station. In this problem, there is no need to send more than one robot to the same location. Another criterion is that the total distance traveled by the team should be minimized as much as possible.

This problem can be seen as an instance of the multiple Traveling Salesmen Problem (mTSP). Our approach is to use a scheduling algorithm to generate routes for the robots in order to cover all locations, as shown in Figure 1. But depending on the disposal of the locations in the environment, the scheduling may generate routes in which some robots travel much more than others, causing a low battery issue on them. Furthermore, in the worst case, these robots will not be able to finish their routes. The reason why this happens is because the scheduling uses the total distance minimization as single goal. Our idea is to add the balance of the distance traveled by all robots as a second goal. However, these two goals are conflicting because when we try to minimize the total distance, the routes can become imbalanced. On the other hand, when we try to balance them, the total distance can increase. To deal with this, we use multi-objective Genetic Algorithms to solve the mTSP regarding these two goals.



Figure 1 – *Coordination*: route scheduling. This figure illustrates the scenario where the routes (colored lines) of 3 robots (black dots) cover all desired locations (red crosses) of the environment.

The next problem we explored is the *localization*. Many localization techniques use devices mounted on the robots to scan the environment in order to build a map. Then, during the execution, the robots capture observations from these devices and try to match them with the map to infer their current locations.

Laser range finders and depth cameras, like Microsoft Kinect, have been the best choice for robot localization in indoor environments. However, when these environments are used by humans, we may find a lot of moveable objects like furniture. Such objects appear static during the mapping, but are likely to be moved around frequently, which changes the map and cause mistakes on the localization.

Indoor localization based on WiFi signal strength is also a widely studied problem as most of human environments are equipped with WiFi network. But the minimum error of WiFi based localization may be greater than that of laser range finder or depth cameras.

Hence, in our approach, we introduce a localization algorithm that uses both Kinect and WiFi sensors, as illustrated by Figure 2. We keep two Particle Filters, one for each sensor. The WiFi Particle Filter gives the global location of the robot since it is not affected by changes in the environment as the Kinect approach, and it is frequently updated using signals of Access Point devices from different parts of the environment. It also bounds the range of particles in the Kinect Particle Filter. Thus, we extract the estimated location of the robot using the Kinect Particle Filter, which has a great local precision.

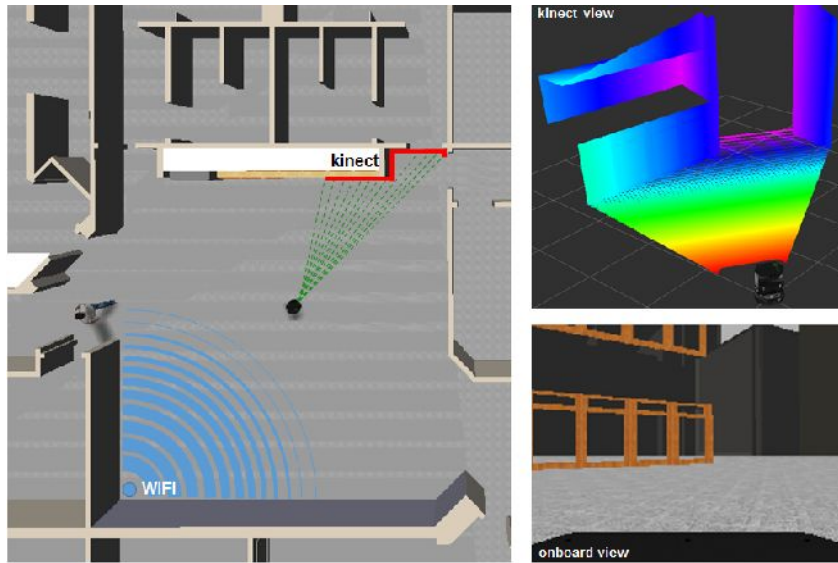


Figure 2 – *Localization*: estimated location based on Kinect and WiFi observations.

Our algorithm has another feature that can tackle the initialization and kidnapping problem. For some robotic platforms, when they are turned on, or completely lost due to device failures or improper motions, they require human intervention, i.e., someone must inform them where they are. In our algorithm, when the WiFi Particle Filter, which gives

the global location, reaches a low threshold, all particles from both Particle Filters are spread all over the environment. After that, the algorithm waits for the convergence of the WiFi Particle Filter. Then, it moves the Kinect particles to that place and takes the estimated location.

We also provide new methods for WiFi mapping and localization to be used in indoor open areas, such as the concept of irregular grid maps, interpolation strategies to estimate WiFi signals continuously along the map, algorithms to replace particles and infer the current pose of the robot on the map.

The last problem we addressed is that of *navigation*. Since humans, objects, and robots are coexisting at the same environment, we desire a reactive controller to avoid collision while robots are navigating. We built an hybrid intelligent system based on Fuzzy Logic and Artificial Neural Network to detect obstacles and select the best motion for the robots to not hit on them.

Furthermore, we are also interested in how efficiently the path planning can be when a robot is navigating to some location and suddenly gets blocked by an obstacle. We treat this problem using a dynamic path planning algorithm. This algorithm aims to replan the path but not from scratch. Instead, it tries to fix the path taking into account the current search space. However, depending on the configuration of the environment and how blocked the robot is, the replanned path may be much longer than the original or, in the worst case, the destination cannot be reached anymore. So, our idea is to adapt this algorithm to detect this situation and ask for human assistance. For example, the robot can ask a human to move the object that is blocking it (Figure 3).

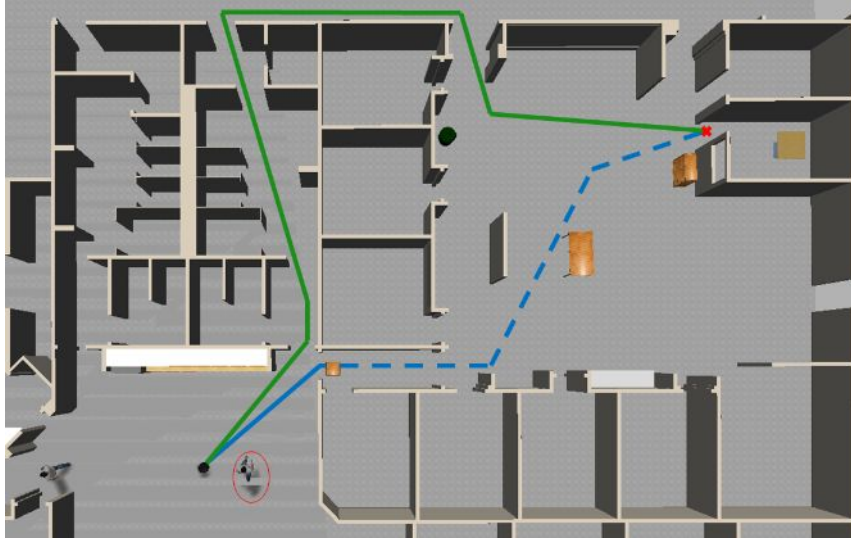

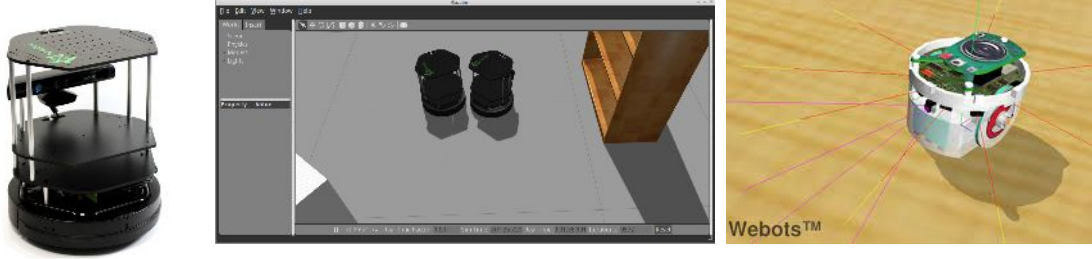


Figure 3 – *Navigation*: obstacle avoidance and dynamic path planning with human assistance. This picture illustrates a scenario where a cabinet is blocking the current path of the robot (in blue). The robot computes another path (in green), but it chooses to ask for human assistance (red circle) instead.

This thesis includes extensive experimental evaluation of all algorithms developed using real robots and virtual simulators. Videos of the experiments are available on the text via QRcodes, such as , so the reader can either scan or click on it to open.



(a) Real Turtlebot. (b) Turtlebots on Gazebo simulator. (c) Epuck on Webots simulator.

Figure 4 – Robots used in the experiments.

1.3 Contributions

The key contributions of this thesis are as follows.

□ Using Genetic Algorithms to minimize the distance and balance the routes for the multiple Traveling Salesman Problem

The Traveling Salesman Problem (TSP) has been used to model many real world applications. In this problem, a salesman travels using the shortest route between the cities that he must visit and returns to the depot. If it is required to use more than one salesman in the solution, the problem is called multiple Traveling Salesmen Problem (mTSP), and the objective is to minimize the total distance traveled by them. We noticed that when only the total distance is minimized, depending on the distribution of the cities, some salesmen tend to travel much more than others. Thus, we introduce a new way to model mTSP by the means of Genetic Algorithms (GAs) components so we can generate approximate solutions to reduce both the total distance and the difference between the distance traveled by each salesman. Since there are more than one objective to be optimized, two approaches were evaluated. A multi-objective GA and a mono-objective GA with a fitness function that combines both objectives. In order to find the best results for each approach, different methods for crossover and selection were used. Experimental results show the effectiveness of the proposed GAs. The results further indicate that, considering both objectives, the multi-objective GA generates more balanced solutions for almost all problem instances.

Paper Submission

Status: published (ALVES; LOPES, 2015)

Conference: Congress on Evolutionary Computation (CEC), Japan - 2015

Qualis: A1

□ **Route Scheduling for a Team of Service Robots in Indoor Environments with Genetic Algorithms**

Service robots aim at helping humans in their daily tasks at home or in the office. Some of these tasks can be performed by a team of robots that need to visit specific locations in the environment. In order to coordinate the visits as efficiently as possible, we model this problem as being an instance of the multiple Traveling Salesmen Problem (mTSP), in which salesmen are robots and cities are locations to be visited. Based on our previous work (ALVES; LOPES, 2015), we built a centralized planner system to schedule routes for a team of robots using our Genetic Algorithms (GAs). The main novelties in this approach are as follows. We use real indoor laser maps acquired by robots to schedule the routes. Inflation techniques are applied on the maps in order to yield safer paths based on the diameter of the robots. Instead of using euclidean distance to estimate the length of the routes, we create a path matrix with A* algorithm to store feasible paths between all possible pairs of locations. The GAs compute routes for the robots and calculate the objectives using the path matrix. Then, the sequence of locations to be visited by each robot can be sent via ROS messages to real or virtual robots.

Paper Submission

Status: in submission

□ **Checkpoint Coverage with WiFi Localization for Indoor Service Robots**

Coverage is a fundamental aspect for many tasks executed by service robots. The general strategy involves performing a trajectory through an environment that ensures complete coverage of the area while minimizing path repetition. In order to guarantee that the robot is visiting the right places and it is moving properly around the area, it is required an accurate localization algorithm. However, some environments are full of dynamic objects, which can be a problem for localization techniques such as rangefinder and depth camera localization. We introduce a Checkpoint Coverage supported by WiFi localization to handle this issue. First, a WiFi map is created by collecting WiFi data from discrete points of the target area. Then, using this map, a Monte Carlo Localization (MCL) algorithm estimates the robot's location during the coverage. Finally, a real-time algorithm manages the list of checkpoints that the robot should visit. Experiments were carried out in two different areas at Carnegie Mellon University (CMU). The results show a very low

localization error while the robot performs a path that covers the checkpoints of both areas.

Paper Submission

Status: published (ALVES; LOPES; VELOSO, 2016)

Conference: Workshop on Autonomous Mobile Service Robots (WSR-1) held in the 25th International Joint Conference on Artificial Intelligence (IJCAI), USA - 2016

Qualis: no Qualis for WSR-1

□ **Indoor Localization for Service Robots using Kinect and WiFi sensors**

Indoor Localization for Autonomous Robots with different sensors and algorithms is a widely studied problem. However, every sensor and algorithm has limitations, thus we believe that no single localization algorithm can be perfect to all situations. We present a new approach that use two Particle Filters (PFs) to estimate robots location within indoor environments, one for WiFi data and another for Kinect data. PF is a general Monte Carlo (sampling) method for performing inference in state-space models where the state of a system evolves in time and information about the state is obtained via noisy measurements made at each time step. The reason why we don't merge both data into a single PF, as other works, is because the WiFi PF gives a better estimate about the global localization, and the Kinect PF is very accurate locally. Therefore, we use WiFi particles to estimate the global position of the robot in the environment. Then, we use Kinect particles, in a certain range of the global position, to better estimate the actual position of the robot. Furthermore, we introduce new adaptations to the WiFi localization method proposed by (BISWAS; VELOSO, 2010) for indoor open areas. First, the concept of irregular grids for WiFi mapping is applied, where WiFi signal strengths are collected at discrete points in a grid fashion. Then, a Perceptual Model estimates WiFi signal strengths for any location in the environment using different interpolation strategies. Also, to infer the the robot's location, we select the cell of the grid with highest particle weight, and then the center of mass is taken as the current location of the robot. We performed different types of experiments to compare the following approaches: WiFi Localization, Kinect Localization, and Kinect+WiFi Localization. We also investigated the initialization and kidnapping problem using the Kinect+WiFi Localization approach.

Paper Submission

Status: in submission

□ **Obstacle avoidance for mobile robots: a Hybrid Intelligent System based on Fuzzy Logic and Artificial Neural Network**

Obstacle avoidance is one of the most important aspects of autonomous mobile robots. This task is composed by two phases. First, the robot must detect obstacles in the environment with its sensors. Then, it must choose an appropriate movement to go through the environment without colliding. However, the noise produced during the sensors reading can lead the robot to take wrong decisions. We introduce a novel robot obstacle avoidance controller using a Hybrid Intelligent System (HIS) based on Fuzzy Logic (FL) and Artificial Neural Networks (ANN). The FL treats the data of infrared sensors and then feeds an ANN that decides which movement the robot must perform. The HIS was compared to another approach in which the data of infrared sensors are used directly in an ANN. Empirical results show that HIS avoids more collisions and improves the smoothness of navigation.

Paper Submission

Status: published (ALVES; LOPES, 2016)

Conference: IEEE International Conference on Fuzzy Systems (FUZZ), Canada - 2016

Qualis: B1

□ **Indoor Navigation with Human Assistance for Service Robots using D*Lite**

Most path planning algorithms assume that the environment is completely known before the robot begins its traverse towards the goal destination. Usually, they use distance functions and heuristics to find the lowest cost path to the robot. However, indoor human environments are very dynamic in the sense that objects can move over time, thus blocking or releasing paths while the robot is navigating. The algorithm D*, also known as Dynamic A*, behaves like A* except that the arc costs can change as the algorithm runs. D* and its variants, such as D*Lite, have been widely used for mobile robot and autonomous vehicle navigation including the Mars rovers Opportunity and Spirit. Unfortunately, depending on how blocked the robot is, the new path computed by D* or D*Lite can be too long or, in the worst case, it cannot reach the destination anymore. To overcome this issue, we introduce a new adaptation to D*Lite which allows robots to ask for human assistance to move objects on its surrounding when its path is blocked, so it can try to generate shorter paths. The algorithm also computes whether and how long the robot can wait for help until it decides to take the new path. Experiments were carried out using simulation and a real robot.

Paper Submission

Status: in submission

1.4 Thesis Organization

The thesis is organized as follows:

- **Chapter 2 - Background** We introduce relevant background on indoor autonomous robots regarding the main topics of this thesis, namely: coordination, localization, and navigation.
- **Chapter 3 - Localization** We address the indoor localization problem for autonomous robots using Kinect and WiFi sensors. First, we explain how we map the environment using both sensors. Then, we show how we use Particle Filters to estimate the robot's location in real time on the generated maps.
- **Chapter 4 - Navigation** We introduce algorithms for navigation in indoor human environments. First, we present a Hybrid Intelligent System (HIS) based on Fuzzy Logic (FL) and Artificial Neural Network (ANN) that works as a reactive controller to avoid obstacles. Then, we show an adaption to the D*Lite algorithm that enables robots to replan their paths efficiently and also ask for help, within a time window, to move blocking objects on the way.
- **Chapter 5 - Coordination** We present how we model the coordination problem for a team of robots as an instance of the multiple Traveling Salesmen Problem (mTSP). Then, we show the solution for this problem by means of a centralized system that uses Genetic Algorithms (GAs) with two objectives: the minimization of the total distance traveled by the team and the balancing of the routes.
- **Chapter 6 - Conclusion** We conclude the document with a summary of its contributions, and expected directions of future work.

CHAPTER 2

Background

In this chapter, we introduce fundamentals and relevant background on *localization*, *navigation*, and *coordination* concepts related to this thesis. It is not intended to be an exhaustive review of all existing algorithms and methodologies, but a brief introduction about them. More specific background and related work for *localization*, *navigation*, and *coordination* are given separately in their respective chapters.

The remainder of the chapter is organized as follows. Section 2.1 describes the general idea behind probabilistic algorithms for *localization*. In Section 2.2, the *navigation* topic is divided into Obstacle Avoidance and Path Planning. We review some concepts about Fuzzy Logic and Artificial Neural Network as they are used in our reactive controller for Obstacle Avoidance. For Path Planning, we present some heuristic search algorithms widely applied in robotics navigation. A formal description of the multiple Traveling Salesmen Problem used to model our robots *coordination* task and the general structure of Genetic Algorithm are shown in Section 2.3. Finally, Section 2.4 provides an overview of autonomous robots deployed in indoor human environments throughout history.

2.1 Localization

In Artificial Intelligence, there is significant interest in probabilistic algorithms, as deterministic algorithms are not able to handle the uncertainty of many real world problems, which is the case of robot localization.

2.1.1 Bayes Filter

A key concept in probabilistic robotics is the *belief*, which reflects the robot's internal knowledge about the state of the environment. Bayes Filter is the most general algorithm for calculating *beliefs*.

In this algorithm, the new *belief* $bel(x_t)$ for a real state variable x at time step t using the measurement probability $p(z_t|x_t)$, the state transition probability $p(x_t|u_t, x_{t-1})$, and

the previous *belief* $bel(x_{t-1})$, is calculated recursively as shown in Algorithm 1.

Algorithm 1 bayes-filter($bel(x_{t-1}), u_t, z_t$)

```

1: for all  $x_t$  do
2:    $\overline{bel}(x_t) \leftarrow \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx$ 
3:    $bel(x_t) \leftarrow \eta p(z_t|x_t) \overline{bel}(x_t)$ 
4: end for
5: return  $bel(x_t)$ 

```

There are two essential phases in this process, known as *predict* (line 2) and *update* (line 3). The *predict* phase updates the *belief* based on the transition u_t from x_{t-1} to x_t using the probability $p(x_t|u_t, x_{t-1})$. Then, the *update* phase performs another update based on the measurement z_t using the probability $p(z_t|x_t)$. The constant η is just a normalization factor.

Usually, in terms of robot localization in a typical 2D environment, state represents the robot's pose (location and orientation) in the form of a tuple (x, y, θ) , transitions are computed by odometry, and measurements are captured by range sensors.

2.1.2 Particle Filter

The Monte Carlo Localization (MCL) (DELLAERT et al., 1999), also known as Particle Filter localization, is an alternative implementation of the Bayes Filter. It has been shown to be effective at localizing a robot in a number of scenarios. Given a map of the environment, the algorithm estimates the pose of a robot as it moves and senses the environment.

In MCL, the *belief* $bel(x_t)$ is represented by a set of weighted samples, or particles, $\chi_t = \{x_t^i, w_t^i\}_{i=1:M}$, where x_t^i is the state of the particle i at time t , w_t^i its associated weight, and M is the number of particles. The weight represents the importance factor of the particle on the set. The Particle Filter is depicted by the Algorithm 2.

Algorithm 2 particle-filter(χ_{t-1}, u_t, z_t)

```

1:  $\overline{\chi}_t \leftarrow \chi_t \leftarrow \emptyset$ 
2: for  $i \leftarrow 1..M$  do
3:   sample  $x_t^i \sim p(x_t|u_t, x_{t-1}^i)$ 
4:    $w_t^i \leftarrow p(z_t|x_t^i)$ 
5:    $\overline{\chi}_t \leftarrow \overline{\chi}_t + \langle x_t^i, w_t^i \rangle$ 
6: end for
7: for  $i \leftarrow 1..M$  do
8:   draw  $j$  with probability  $\propto w_t^j$ 
9:   add  $x_t^j$  to  $\chi_t$ 
10: end for
11: return  $\chi_t$ 

```

Like Bayes Filter, the MCL also generates the particle set χ_t recursively from χ_{t-1} , assuming the Markov property that the current state's probability distribution depends only on the previous state and not any ones before that. It receives as parameters the previous *belief* χ_{t-1} , the last transition u_t , and the last measurement z_t .

Initially, the current particle set χ_t and the temporary particle set $\bar{\chi}_t$ are empty (line 1). Then, for each of the M particles, the algorithm generates a hypothetical state x_t^i for time t based on the particle x_{t-1}^i , the transition u_t , and the probability $p(x_t|u_t, x_{t-1}^i)$ (line 3). This step corresponds to the *predict* phase on the Bayes Filter. Basically, in this phase, the robot predicts its new location based on the actuation command given, by applying the simulated motion to each of the particles. For example, if a robot moves forward, all particles move forward in their own directions. Since no actuator is perfect in the real world due to drifting on an irregular environment or small differences in wheel radius, the motion model must compensate for noise. As a consequence, the particles may diverge, requiring other steps to sense the environment and remove wrong particles.

Then, the algorithm computes the weight w_t^i as the probability of the measurement z_t under the hypothetical state x_t^i , that is $p(z_t|x_t^i)$, which works as the *update* phase on the Bayes Filter (line 4). This phase is important because when the robot senses its environment, it updates its particles to more accurately reflect where it is. With x_t^i and w_t^i , a new particle is created and added to the temporary particle set $\bar{\chi}_t$ (line 5).

Another important phase appears in this algorithm, called *resampling*. This phase is responsible for the convergence of the algorithm. It draws with replacement M particles from the temporary particle set $\bar{\chi}_t$ and adds them to the current particle set χ_t using a probability function based on the weights (lines 7-10). Particles consistent with sensor readings are more likely to be chosen (possibly more than once) and particles inconsistent with sensor readings are rarely picked. As such, particles converge towards a better estimate of the robot's state.

The convergence also increases as the robot moves around and senses different parts of the environment.

2.2 Navigation

We are interested in providing an efficient and safe navigation for autonomous robots in indoor human environments. The navigation task involves an off-line deliberative phase, which is the planning of a path that will guide the robot from its initial position to the goal position. During the execution, if this path is blocked, it is necessary to replan or repair it.

Human environments are composed by movable and moving objects that frequently change the environment, which causes the computation of paths over and over. Therefore, we divided the navigation task into Obstacle Avoidance and Path Planning.

Moving objects are too dynamic to be considered in a Path Planning, thus we introduce a reactive controller to avoid collision instead of trying to fix the robot's path. Such controller uses Fuzzy Logic to detect obstacles around the robot, and an Artificial Neural Network to determine the robot's motion for traversing the environment without colliding with the objects.

To replan paths efficiently, we use a dynamic search algorithm to adapt to changes in the environment, caused by movable objects, found during the navigation.

2.2.1 Obstacle Avoidance

We use Fuzzy Logic and Artificial Neural Network to create a Hybrid Intelligent System that works as a reactive controller for Obstacle Avoidance. These two AI techniques are described in the next sections.

2.2.1.1 Fuzzy Logic

Fuzzy Logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1, considered to be “fuzzy”. By contrast, in Boolean Logic, the truth values of variables may only be the “crisp” values 0 or 1.

Fuzzy Logic has been studied since the 1920s, but it received great interest with the advent of Fuzzy Sets introduced by (ZADEH, 1965). The definition of a Fuzzy Set is like a superset of the definition of a set in the ordinary sense of the term. Let X be a space of points, with a generic element of X denoted by x . Thus $X = \{x\}$. A Fuzzy Set A in X is characterized by a membership function $f_A(x)$ which associates with each point in X a real number in the interval $[0, 1]$, with the values of $f_A(x)$ at x representing the “grade of membership” of x in A . Thus, the nearer the value of $f_A(x)$ to unity, the higher the grade of membership of x in A .

Membership functions for Fuzzy Sets can be defined in any number of ways as long as they follow the rules of the definition of a Fuzzy Set. The shape of the membership function used defines the Fuzzy Set and so the decision on which type to use is dependant on the purpose. The membership function choice is the subjective aspect of Fuzzy Logic, it allows the desired values to be interpreted appropriately. The most common membership functions are: triangular, trapezoidal, singleton, and Gaussian.

The main application of Fuzzy Logic is in the area of control systems. A fuzzy controller can be broken down into three main process (Figure 7). The first of these is the fuzzification, this uses defined membership functions to process the inputs and to fuzzify them. These fuzzified inputs are then used in the second part, the rule-based inference system. This system uses previously defined linguistic rules to generate a fuzzy response. The fuzzy response is then defuzzified in the final process: defuzzification. This process will provide a real number as an output.

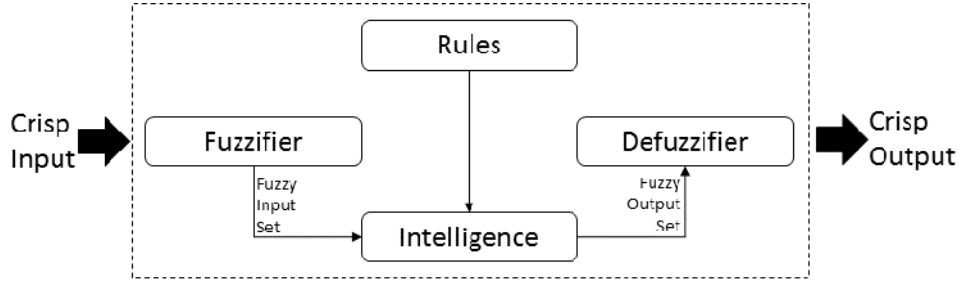


Figure 5 – Fuzzy Controller.

2.2.1.2 Artificial Neural Network

An Artificial Neural Network (ANN) (ROSENBLATT, 1961) is a system that is loosely based on the learning processes found in the neural networks of the human brain. It is capable of processing information by means of connecting together relatively simple information processing units with links that allow each unit to communicate with each other by relatively simple signals. Each link has a numeric weight associate with it and is the primary means of long-term storage in the neural network. The weights are updated during the learning process of the ANN.

The following diagram illustrates the typical structure of a unit:

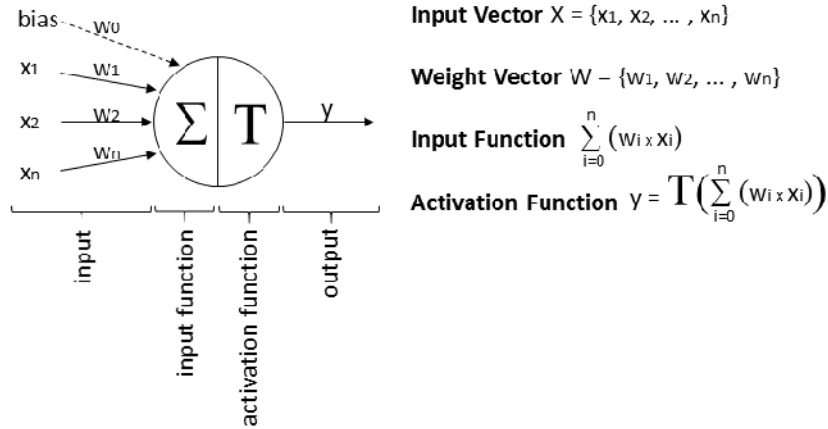


Figure 6 – Perceptron.

There are several ANN architecture in the literature. Maybe the most popular one is the Multilayer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986). It is a feedforward ANN model that maps sets of input data onto a set of appropriate outputs. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing unit) with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.

The MLP is a finite directed acyclic graph, where:

- nodes that are no target of any connection are called input neurons. A MLP that should be applied to input patterns of dimension n must have n input neurons, one for each dimension.
- nodes that are no source of any connection are called output neurons. A MLP can have more than one output neuron. The number of output neurons depends on the way the target values (desired values) of the training patterns are described.
- all nodes that are neither input neurons nor output neurons are called hidden neurons.

Since the graph is acyclic, all neurons can be organized in layers, with the set of input layers being the first layer.

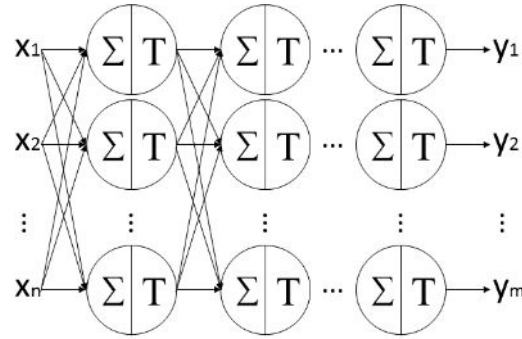


Figure 7 – Multilayer Perceptron.

The operations of the backpropagation neural networks can be divided into two steps: feedforward and backpropagation. In the feedforward step, an input pattern is applied to the input layer and its effect propagates, layer by layer, through the network until an output is produced. The network's actual output value is then compared to the expected output, and an error signal is computed for each of the output nodes. Since all the hidden nodes have, to some degree, contributed to the errors evident in the output layer, the output error signals are transmitted backwards from the output layer to each node in the hidden layer that immediately contributed to the output layer. This process is then repeated, layer by layer, until each node in the network has received an error signal that describes its relative contribution to the overall error.

Once the error signal for each node has been determined, the errors are then used by the nodes to update the values for each connection weights until the network converges to a state that allows all the training patterns to be encoded. The backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

2.2.2 Path Planning

Path Planning is one of the most important issues in the navigation process, and subsequently much research in the field of robotics is concerned with path planning. Usually, this task is performed by search algorithms that aim to find paths from start to goal positions in the configuration space: data structures that show the position and orientations of objects and robots in the environment, including both the free and obstructed regions. The robot's internal map of the environment can be represented by either an occupancy grid, a vertex graph, or a Voronoi diagram. An efficient path is one that minimizes path costs, where the cost is typically the travel distance or time.

2.2.2.1 Search Algorithms

A search consists of finding a sequence of actions that guides an agent from its initial state to its goal state. Since the solution is a sequence of actions, search algorithms work by considering various possible sequences. These sequences starting at the initial state form a search tree, where the initial state is the root, the branches are actions, and nodes correspond to states in the state space. During the search, the algorithm expands the current state by applying all possible actions to find new states. Then, it looks for the most promising state according to its cost function, and moves to that state, marking the previous one as parent state. This process repeats until the goal state is found. This is the essence of search algorithms.

The criteria to evaluate search algorithms are (RUSSELL; NORVIG, 2010):

- ❑ Completeness: is the algorithm guaranteed to find a solution when there is one?
- ❑ Optimality: does the strategy find the best possible solution?
- ❑ Time complexity: how long does it take to find the solution?
- ❑ Space complexity: how much memory is necessary to perform the search?

Algorithms that don't have any additional information about states, beyond that provided in the problem definition, belong to the Uninformed Search category (also called Blind Search). Thus, all they can do is expand states and distinguish the goal state from the rest of them. The main difference between algorithms in this category is the search strategy, i.e., the order in which states are expanded. Most popular algorithms in this category are: breadth-first search, depth-first search, bidirectional search, and uniform-cost search.

The informed search category (also called heuristic search) uses problem-specific knowledge beyond the problem itself to set the order in which states are expanded. Usually, algorithms in this category find solutions more efficiently than those of uninformed search. This knowledge is represented by an heuristic function $h(s)$ that estimates the cost of

reaching the goal state from a given state. Typically this function is extracted from a relaxed solution of the problem. The Dijkstra's algorithm (DIJKSTRA, 1959) and A* (HART; RAPHAEL, 1968) are two popular algorithms of this category, and are commonly used for path planning in robotics. Both of them return an optimal solution, and can be considered as special forms of dynamic programming.

Algorithm 3 A*

```

ComputeShortestPath():
1: while ( $\text{argmin}_{s \in OPEN} (g(s) + h(s, s_{goal})) \neq s_{goal}$ ):
2:   remove state  $s$  from the front of  $OPEN$ ;
3:   for all  $s' \in Succ(s)$ :
4:     if ( $g(s') > g(s) + c(s, s')$ ):
5:        $g(s') = g(s) + c(s, s')$ ;
6:       insert  $s'$  into  $OPEN$  with value  $g(s') + h(s', s_{goal})$  ;

Main():
7: for all  $s \in S$ :
8:    $g(s) = \infty$ ;
9:  $g(s_{start}) = 0$ ;
10:  $OPEN = \emptyset$ ;
11: insert  $s_{start}$  into  $OPEN$  with value  $g(s_{start}) + h(s_{start}, s_{goal})$  ;
12: ComputeShortestPath();

```

A* plans a path from an initial state $s_{start} \in S$ to a goal state $s_{goal} \in S$, where S is the set of states in some finite state space. To do this, it stores an estimate $g(s)$ of the path cost from the initial state to each state s . Initially, $g(s) = \infty$ for all states $s \in S$. The algorithm begins by updating the path cost of the start state to be zero, then places this state onto a priority queue known as the *OPEN* list. Each element s in this queue is ordered according to the sum of its current path cost from the start, $g(s)$, and a heuristic estimate of its path cost to the goal, $h(s, s_{goal})$. The state with the minimum such sum is at the front of the priority queue. The heuristic $h(s, s_{goal})$ typically underestimates the cost of the optimal path from s to s_{goal} and is used to focus the search.

The algorithm then pops the states at the front of the queue and updates the cost of all states reachable from this state through a direct edge: if the cost of state s , $g(s)$, plus the cost of the edge between s and a neighboring state s' , $c(s, s')$, is less than the current cost of state s' , then the cost of s' is set to this new, lower value. If the cost of a neighboring state s' changes, it is placed on the *OPEN* list. The algorithm continues popping states off the queue until it pops off the goal state. At this stage, if the heuristic is admissible, i.e. guaranteed to not overestimate the path cost from any state to the goal, then the path cost of s_{goal} is guaranteed to be optimal.

The Algorithm 3 shows the pseudo-code of A*. It is also possible to switch the direction of the search in A*, so that planning is performed from the goal state towards the start state, which is applied to other algorithms.

Although A^* can find the optimal path in a static environment, it cannot handle situations caused by changes in the environment during the execution of the path. Even if the agent is able to detect these changes on-line, A^* needs to replan another path from scratch considering the current configuration of the environment. However, replanning from scratch every time something changes in the environment can be very computationally expensive.

D^* is the dynamic version of A^* that sacrifices the optimality for computational efficiency in changing environments. Notable implementations of D^* in the field robotics include the DARPA Urban Challenge vehicle from Carnegie Mellon University, and Mars rovers Opportunity and Spirit (WOODEN, 2006).

D^* Lite is a simpler version of D^* , and has been found to be slightly more efficient for some navigation tasks (FERGUSON; LIKHACHEV; STENTZ, 2005).

Algorithm 4 D^* Lite

```

key( $s$ ):
1: return  $[min(g(s), rhs(s)) + h(s_{start}, s); min(g(s), rhs(s))]$ ;

UpdateState( $s$ ):
2: if  $s$  was not visited before:
3:    $g(s) = \infty$ ;
4: if  $(s \neq s_{goal})$  :    $rhs(s) = min_{s' \in Succ(s)} (c(s, s') + g(s'))$ ;
5: if  $(s \in OPEN)$  :   remove  $s$  from  $OPEN$ ;
6: if  $(g(s) \neq rhs(s))$  :   insert  $s$  into  $OPEN$  with  $key(s)$ ;

ComputeShortestPath():
7: while  $(min_{s \in OPEN} (key(s)) < key(s_{start}) \text{ OR } rhs(s_{start}) \neq g(s_{start}))$ :
8:   remove state  $s$  with the minimum key from  $OPEN$ ;
9:   if  $(g(s) > rhs(s))$ :
10:     $g(s) = rhs(s)$ ;
11:    for all  $s' \in Pred(s)$  UpdateState( $s'$ );
12:   else:
13:     $g(s) = \infty$ ;
14:    for all  $s' \in Pred(s) \cup \{s\}$  UpdateState( $s'$ );

Main():
15:  $g(s_{start}) = rhs(s_{start}) = \infty$ ;  $g(s_{goal}) = \infty$ ;
16:  $rhs(s_{goal}) = 0$ ;  $OPEN = \emptyset$ ;
17: insert  $s_{goal}$  into  $OPEN$  with  $key(s_{goal})$ ;
18: forever:
19:   ComputeShortestPath();
20:   Wait for changes in edge costs;
21:   for all directed edges  $(u, v)$  with changed edge costs:
22:     Update the edge cost  $c(u, v)$ ;
23:     UpdateState( $u$ );

```

D^* Lite initially constructs an optimal path from the initial state to the goal state in

exactly the same manner as backwards A*. The Algorithm 4 depicts the pseudo-code of D*Lite. It keeps a least-cost path from an initial state s_{start} to a goal state s_{goal} based on an estimate $g(s)$ of the cost from each state s to the goal. It also keeps an one-step lookahead cost, called Right Hand Side $rhs(s)$ value, which satisfies:

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in Succ(s)} (c(s, s') + g(s')) & \text{otherwise} \end{cases}$$

Here, $Succ(s)$ is the set of successors of s and $c(s, s')$ represents the cost of moving from s to s' . The $rhs(s)$ is equal to the cost to the parent of a node plus the current cost to travel to that node. By comparing this value to the cost to the node we can detect inconsistencies, as depicted by the example of Figure 8.

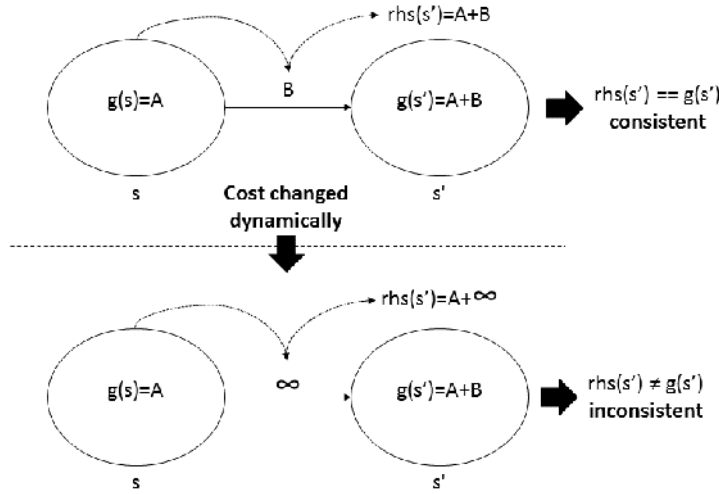


Figure 8 – Finding inconsistencies.

Inconsistency falls into two categories: underconsistency ($g(s) < rhs(s)$) and overconsistency ($g(s) > rhs(s)$). When a node is found to be underconsistent that means that the path to that node was made to be more expensive. In robotics this would correspond to an unexpected obstacle blocking the way. Nodes found to be underconsistent will need to be reset and paths completely recalculated. Likewise, when a path is found to be overconsistent that means that the path to that node was made to be less expensive. This could happen when the robot finds a new shortcut or an obstruction is now cleared.

The algorithm uses a heuristic and a priority queue to focus its search and to order its cost updates efficiently. The heuristic $h(s, s')$ estimates the cost of moving from state s to s' , and needs to be admissible.

The priority queue *OPEN* holds the inconsistent states, where the priority is given by:

$$\begin{aligned} key(s) &= [k_1(s), k_2(s)] \\ &= [\min(g(s), rhs(s) + h(s_{start}, s)), \min(g(s), rhs(s))] \end{aligned} \tag{1}$$

Thus, $key(s) \leq key(s')$ iff $k_1(s) < k_1(s')$ or both $k_1(s) = k_1(s')$ and $k_2(s) < k_2(s')$.

The algorithm expands states from *OPEN* in increasing order, updating their g-values and their predecessors' rhs-values, until there is no state in the queue with a priority less than that of the start state. To detect changes in the current state and replan from there, it searches backwards and consequently focusses its search towards the start state rather than the goal state.

When changes are detected, the states whose paths to the goal are immediately affected by these changes have their path costs updated and are placed on *OPEN* to propagate the effects of these changes to the rest of the state space. In this way, only the affected portion of the state space is processed when changes occur.

2.3 Coordination

We modeled the robots coordination problem addressed in this thesis as being an instance of the multiple Traveling Salesmen Problem (mTSP). The mTSP is a generalization of the well-known Traveling Salesman Problem (TSP), where more than one salesman can be used in the solution. They are examples of combinatorial optimization problems, and have a multiplicity of applications mostly in the areas of routing and scheduling. Since mTSP is classified as NP-Hard, we apply Genetic Algorithms (GAs) to find approximate solutions for different instances of the problem.

2.3.1 multiple Traveling Salesmen Problem

In the mTSP, a set of nodes (locations or cities) are given, and all of the cities must be visited exactly once by the salesmen who all start and end at the single depot node. The number of cities is denoted by n and the number of salesman by m . The goal is to find tours for all salesmen, such that the total traveling cost (the cost of visiting all nodes) is minimized. The cost metric can be defined in terms of distance, time, etc. Some possible variations of the problem are as follows:

- Multiple depots: If there exist multiple depots with a number of salesmen located at each, a salesman can return to any depot with the restriction that the initial number of salesmen at each depot remains the same after all the travel.
- Number of salesmen: The number of salesmen in the problem can be a fixed number or a bounded variable.
- Fixed charges: If the number of salesmen is a bounded variable, usually the usage of each salesman in the solution has an associated fixed cost. In this case the minimization of this bounded variable may be involved in the optimization.

- Time windows: Certain cities must be visited in specific time periods, named as time windows. This extension of mTSP is referred to as multiple Traveling Salesman Problem with Time Windows (mTSPTW).
- Other restrictions: These additional restrictions can consist of the maximum or minimum distance or traveling duration a salesman travels, or other special constraints.

Usually, mTSP is formulated by integer programming formulations. Let $G(V, A)$ be a perfect undirected connected graph with a vertex set $V = \{0, 1, 2, \dots, n\}$ and an edge set $A = \{(i, j) : i, j \in V, i \neq j\}$. If the graph is not perfect, the lack of any edge is replaced with an edge that has an infinite size. For presenting the integer linear programming model for mTSP, the following variables are introduced:

- n = number of nodes to be visited.
- m = number of salesmen.
- C = cost matrix, where:

$$C_{ij} = C_{ji} \wedge C_{ij} + C_{jk} \geq C_{ik}, \forall i, j, k = 1, 2, 3, \dots, n$$

$$X_{ij} = \begin{cases} 1 & \text{if the salesman travels directly for } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

Hence, one of the common integer programming formulations for the mTSP can be written as follows:

- (1) $\min \sum_{i=1}^n \sum_{j=1}^n C_{ij} X_{ij}$
- (2) $\sum_{i=1}^n X_{ij} = 1, j = 2, \dots, n$
- (3) $\sum_{i=1}^n X_{ij} = m, j = 1$
- (4) $\sum_{i=1}^n X_{ij} = 1, i = 2, \dots, n$
- (5) $\sum_{i=1}^n X_{ij} = m, i = 1$
- (6) $\sum_{i \in S} \sum_{j \in N-S} X_{ij} \geq 1 (\emptyset \neq S \subset N = \{2, \dots, n\}), |S| \geq 2$
- (7) $\sum_{i \in N-S} \sum_{j \in S} X_{ij} \geq 1 (\emptyset \neq S \subset N = \{2, \dots, n\}), |S| \geq 2$
- (8) $X_{ij} \in \{0, 1\}$

The objective function (1) minimizes the total distance traveled in a tour. Constraint sets (2) and (3) ensure that the salesmen arrive once at each node and m times at the depot. Constraint sets (4) and (5) ensure that the salesmen leave each node once and the depot m times. Constraint sets (6) and (7) are to avoid the presence of sub-tours for each salesman. Finally, Constraint set (8) defines binary conditions on the variables.

2.3.2 Genetic Algorithm

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems. GAs are search algorithms based on the concepts of natural selection and genetics (GOLDBERG, 1989). They are a subset of a much larger branch of computation known as Evolutionary Computation.

In GAs, there are a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the “fitter” individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “Survival of the Fittest”. In this way the algorithm keeps “evolving” better individuals or solutions over generations, till it reaches a stopping criterion.

Some basic terminology used on GAs are:

- Population: a subset of all the possible solutions to the given problem.
- Chromosomes: is one such solution to the given problem.
- Gene: is one element position of a chromosome that holds the data.
- Fitness Function: function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- Genetic Operators: used to alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

The basic structure of a GA is as follows (Figure 9). It starts by generating a initial population, which can be done with completely random solutions or using a known heuristic for the problem.

Then, a fitness value is computed for each chromosome of the population. The fitness function is simply defined as a function that takes a candidate solution to the problem as input and produces as output how “fit” or “good” the solution is with respect to the problem. Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. Usually, this is the most expensive procedure in the algorithm. In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function.

After that, some chromosomes are chosen for the crossover operation based on their fitness values. Some well-known methods for selection are: Roulette Wheel, Stochas-

tic Universal Sampling, Tournament, Rank, and Random Selection. After selecting the chromosomes, a crossover operator switches their genes and creates new chromosomes.

The next operation in the process is called mutation. In simple terms, the mutation may be defined as a small random adjustment in the chromosome by changing some genes, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability. If the probability is very high, the GA gets reduced to a random search. Mutation is the part of the GA which is related to the “exploration” of the search space, and can recover the search from a local optima.

Only part of this new population must be selected for the next iteration of the algorithm. The Survivor Selection Policy determines which solutions are to be removed and which are to be kept in the next generation. It is crucial as it should ensure that the fitter solutions are not removed of the population, while at the same time diversity should be maintained in the population. Some GAs employ Elitism, which means that the current fittest solution of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest solution of the current population be replaced.

Finally, it is tested if the GA can stop running. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. Therefore, some criteria can be used to stop the algorithm, such as: when there has been no improvement in the population for n iterations, when it reaches an absolute number of generations, and when the objective function value has reached a certain pre-defined value.

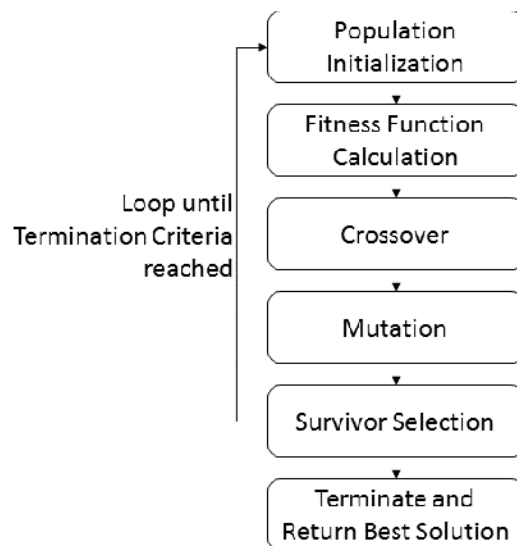


Figure 9 – Genetic Algorithm Structure.

2.4 Autonomous Robots in Real Indoor Human Environments

There have been several autonomous robots working in real indoor human environments, which we review here. From 1966 through 1972, the Artificial Intelligence Center at Stanford Research Institute conducted research on a mobile robot called *Shakey* (Figure 10) (CHAITIN et al., 1970). It was the first robot to actually perform tasks in human environments. Equipped with a limited ability to perceive and model its environment, *Shakey* could perform tasks that required planning, route-finding, and the rearranging of simple objects, most of them using Stanford Research Institute Problem Solver (STRIPS) to transform tasks into sequences of actions.

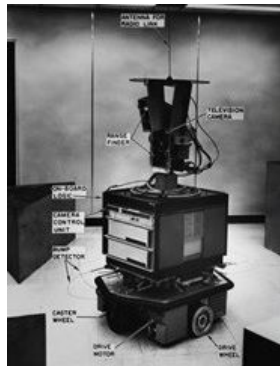


Figure 10 – Shakey (CHAITIN et al., 1970)

Beginning in 1984, *Flakey* (Figure 11) (SAFFIOTTI; RUSPINI; KONOLIGE, 1993) became the second-generation of mobile robot at Artificial Intelligence Center. It had real-time stereo vision algorithms to distinguish and follow people, and a speech recognition system to respond to spoken commands. A Fuzzy controller was responsible to pursue strategic goals under conditions of uncertainty, incompleteness, and imprecision during the navigation.



Figure 11 – Flakey (SAFFIOTTI; RUSPINI; KONOLIGE, 1993)

Rhino (Figure 12) (BUHMANN et al., 1995a) is a mobile robot designed for indoor navigation and manipulation tasks. It was created by the University of Bonn, and then it was used to be the first museum tour-guide robot in the world. *Rhino*'s system is composed by many different modules. The interface modules for sonar, camera, and speech; control the basic communication between the hardware components. On top of that, an obstacle avoidance algorithm analyzes sonar measurements to avoid collisions with obstacles and walls. Global metric and topological maps are constructed on-the-fly using a neural network. A dynamic programming planner explores unknown terrain and navigates to arbitrary target locations. It locates itself by continuously analyzing sonar information. In addition, a fast vision module segments images from two color cameras to find target objects and obstacles that block the path of the robot. *Rhino*'s control flow is monitored by an integrated task planner and a central user interface.



Figure 12 – Rhino (BUHMANN et al., 1995a)

The second-generation museum tour-guide robot was *Minerva* (Figure 13) (THRUN et al., 1999). *Minerva*'s system consists of approximately 20 distributed modules which communicate asynchronously. At the lowest level, various modules interface directly to the robot's sensors and effectors (lasers, sonars, cameras, motors, pan/tilt unit, face, speech unit, touch-sensitive display, internet server, etc.). On top of that, various navigation modules performed functions like mapping, localization, collision avoidance, path planning, and global mission planning. The interaction modules determine the "emotional state" of the robot, control its head direction, and determine what it says (speech, sounds). Finally, the web interface consists of modules concerned with displaying information such as images and the robot's position on the web, and with receiving web user commands.



Figure 13 – Minerva (THRUN et al., 1999)

Xavier robot (Figure 14) (KOENIG; SIMMONS, 1998) has an architecture based on Partially Observable Markov Decision Process (POMDP) which leads to robust long-term autonomous navigation in office environments (with corridors, foyers, and rooms), while it performs tasks requested by users over the web.



Figure 14 – Xavier (KOENIG; SIMMONS, 1998)

PR2 (Figure 15) (OYAMA et al., 2009) is a mobile manipulation platform built by Willow Garage, which was a robotics research lab and technology incubator responsible for the creation of Robot Operating System (ROS). The *PR2* uses laser scan data along with Inertial Measurement Unit (IMU) readings and odometry to build an occupancy grid map using GMapping algorithm, and then localized itself on the map using KLD-sampling algorithm. The *PR2* software system is written entirely in ROS. As such, all *PR2* capabilities are available via ROS interfaces, including both GMapping and KLD-sampling algorithms.



Figure 15 – PR2 (OYAMA et al., 2009)

The *Roomba 980* (Figure 16) (ACKERMAN, 2015), created by iRobot, is a robot vacuum equipped with a camera that allows the robot to navigate using VSLAM (Vision Simultaneous Localization and Mapping). By moving around more efficiently, the robot can cover an entire level of a home with multiple rooms.



Figure 16 – Roomba (ACKERMAN, 2015)

The robotic assembly *Fetch & Freight* (Figure 17) (WISE et al., 2016) (MAHONEY, 2015) are built to work on the floors of warehouses, distribution centers, and light manufacturing facilities. The research editions of *Fetch & Freight* use the ROS Navigation stack, the same used in *PR2*. However, some improvements were made using components provided by their robotic bases. First, a recovery behavior has been added that controls the robot head. This module uses the head pan and tilt stage to “look around” and try to clear obstacles when the robot is unable to find a plan to the goal. In addition, when the robot is navigating, this module tilts the head up and down to get a taller field of view for the head camera, and points the head towards where the robot will be in several seconds of travel. This last addition makes a marked improvement in navigation, primarily through social engineering, as humans move out of the path of the robot since they know where it will be going. Also, a costmap updater was designed to handle the unreliable timing of the head camera.



Figure 17 – Fetch & Freight (WISE et al., 2016)

Care-O-bot (Figure 18) (REISER et al., 2009) is a mobile robot assistant to actively support humans in their daily life. It can be used for a variety of household tasks, for example to deliver food and drinks, to assist with cooking or for cleaning. The main components of the robot are mobility base, torso, manipulator, tray and sensor carrier with sensors. Navigation and collision avoidance currently rely on two laser scanners, which are mounted in the middle of the robot's front and rear. For pose estimation an extended Kalman-Filter is used which correlates line segments extracted from the current scan with a map of the environment.



Figure 18 – Care-O-bot (REISER et al., 2009)

The use of robots in Hotel service was first introduced in Aloft Hotel in Cupertino, California. The Robot named *Relay* (Figure 19) (PRANSKY, 2016) (MAHONEY, 2015) was created by Silicon Valley robotics company Savioke. It comes in a cylindrical shape with a basin and a lid on top. It can carry standard room service items like toiletries, water bottles, and newspapers, and can guide itself to a hotel room. It can also call the elevator via WiFi. A staff dials in a room number and fills the top facing bin with the ordered service item. *Relay's* intelligence is powered by software that its inventor says operates much like Microsoft Kinect's motion-sensing technology. *Relay* can map its route by scanning as far as 40 feet ahead. Higher-resolution image sensors cue *Relay* if unexpected obstacles, such as people or laundry carts on the move within a six-foot radius come on its way.



Figure 19 – Relay (PRANSKY, 2016)

Kiva (Figure 20) is a multi-agent system for warehouse management used by Amazon Inc. (WURMAN; D’ANDREA; MOUNTZ, 2007). It is composed by 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. Each robot is represented in the system by a drive unit agent (DUA) and each shelf by an inventory station agent (ISA). Systemwide resource allocation is centralized in the Job Manager (JM), which also communicates with the customer’s existing warehouse-management system. The JM receives customer orders that need to be filled and assigns drives, pods, and stations to carry out the tasks. Typically, a *Kiva* installation is arranged on a grid with storage zones in the middle and inventory stations spread around the perimeter. The grid constitutes 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 DUAs also maintain a list of highlevel goals and are responsible for prioritizing the goals and accomplishing them as efficiently as possible.



Figure 20 – Kiva (WURMAN; D’ANDREA; MOUNTZ, 2007)

OSHbot (Figure 21) (OSHBOT, 2017) (MAHONEY, 2015) is a customer service robot that helps customers find things in the stores. When the robot is navigating in the store, it knows where all the products are located. When a customer comes into the store they can just talk to the robot, in multiple languages. Mapping is one of the robot’s core functionalities. The robot knows precisely where it is located at all times. There is an integration between that and the planogram, so this way *OSHbot* knows exactly where

these objects are located on the shelves. The robot actually guides customers by its own fully autonomous navigation to the product location. Meanwhile customers are following *OSHbot*, there is a screen on the back for engagement. If customers don't want to follow the robot, *OSHbot* can just show them the product location on the map.



Figure 21 – OSHbot (OSHBOT, 2017)

Techi (Figure 22) (TECHI, 2017) (MAHONEY, 2015) is a collaborative robot, interacting safely alongside people. It can connect to a management system, called Enterprise Manager, which is able to coordinate multiple robots' jobs. These robots have a sonar package on the front and the back of the robot for looking at things that might appear in its path. Instead of LIDAR sensors they use a very unique solution, called Acuity, as LIDAR sensors are not capable of operating successfully in a complex, dynamic environment. Any mobile robot using LIDAR and SLAM can get lost if the route is too different from the reference map that it's using to navigate. With Acuity they use an upward-facing camera placed on top of the vehicle for a 120-degree view. The vehicle looks up at the lights in the ceiling and uses those lights to triangulate its position. As it drives, it keeps looking for the next set of lights to come into view and it uses that to keep it localized in the world. *Techi* robots also have little tablets with eyes, and that tablets communicate with the robots core to anticipate where they are driving. *Techi* robots have been deployed in hospitals, operating throughout the day, making deliveries as needed. When the robots aren't driving, they'll sit and charge to make sure that they are ready to go. The mapping is done using the robot, so it is necessary to walk it around the facility once. The user simply shows the robot every place it could potentially drive, and robot uses the drive navigation laser to do the mapping. Then an accurate facility map is generated, which will be used by all of the robots for path planning and navigation. By the means of Enterprise Manager system, all the robots in the fleet share the same map, so when the map is edited for any reason all of the *Techi* units are automatically updated with the new information. They're all on WiFi. That is how the robots communicate with the Enterprise Manager. Then on the map, the users can designate areas that are forbidden, where they don't want the robot to drive. These areas will be avoided by the path planning algorithm. Or the users can assign one-way aisles, so the robot will always drive in one direction in this aisle. Or even designate two-way directional sections where

the robot will hug the right or the left side of the aisle, or follow any preferred traffic pattern. Once the map is set up, the robots are then on their own to decide how they get from point A to point B based on the map features. So the robot will path plan through the facility and then if an obstacle appears in the path of the robot, for example, a door is shut or there's a gurney parked in the way, or if there's a bunch of boxes or a ladder, or anything that's temporary that it might encounter, the robot will see that with its drive laser and will path plan an alternative route around the facility based on the definition of all the map features.



Figure 22 – Techii (TECHI, 2017)

The Collaborative Robots (*CoBots*) (Figure 23) (VELOSO et al., 2015a) can be viewed as mobile, computing, and sensing robotic platforms, that behave as service robots on multiple floors of the different buildings occupied by the School of Computer Science (SCS) at Carnegie Mellon University (CMU). They can perform different types of tasks, including escorting visitors, transporting objects, and engaging in semi-autonomous telepresence. These tasks can be requested by users through a website, in person through speech or on the robot's touch screen. *CoBots* are coposed by omnidirectional motion, onboard computation, interaction interfaces, carrying baskets, and different combinations of depth sensing (cameras and LIDAR). The robots classify sensed obstacles as map-known long-term features (walls) or map-missing short-term (furniture) and dynamic (people) features. This explicit distinction enables the overall effective episodic non-Markovian localization approach.



Figure 23 – CoBots (VELOSO et al., 2015a)

Localization

In the era of automation, the ability to localize people and devices within indoor environments has become increasingly important for many applications, especially in robotics. With the emergence of global satellite positioning systems, such as Global Positioning System (GPS), the performance of outdoor positioning has become excellent. But the signal from GPS satellites is too weak to come across most buildings, thus making GPS ineffective for indoor localization (FARID; NORDIN; ISMAIL, 2013). Therefore indoor positioning became a focus of research and development during the past decade.

Indoor environments are particularly challenging for positioning for several reasons, such as severe multipath from signal reflection from walls and furniture, Non-Line-of-Sight (NLoS) conditions, high attenuation and signal scattering due to greater density of obstacles, fast temporal changes due to the presence of people and opening of doors, high demand for precision and accuracy, and others. On the other hand, indoor settings facilitate localization in many ways, like the presence of small coverage areas, low weather influences such as small temperature gradients and slow air circulation, fixed geometric constraints from planar surfaces and orthogonality of walls, infrastructures (such as electricity, internet access, and walls) suitable for target mounting, lower dynamics due to slower walking and driving speeds, etc.

Considering the drawbacks of this scenario, apparently there is no overall solution based on a single technology, such as that provided outdoors by satellite-based localization, for indoor environments. Thus, we are still far away from achieving cheap provision of global indoor positioning with an accuracy of 1 meter (MAUTZ, 2012).

Recently, (BRENA et al., 2017) have published a survey that provides a technological perspective of indoor positioning systems, comprising a wide range of technologies and approaches. These technologies can be passive and/or active. The term “passive” means that the infrastructure generates the signal and that the object or person to be located receives it, and “active” means that there is an onboard device capable of generate and treat observations to localize itself in the environment.

Table 1 presents the comparison, made by (BRENA et al., 2017), between the following

Table 1 – Comparison of main indoor positioning technologies. [UC] End User Cost, [IC] Installation and Maintenance Cost, [H] High, [L] Low, [ML] Multiple Levels (BRENA et al., 2017).

Technology	Accuracy	Coverage	IC	UC
Active				
InfraRed	57cm–2.3m	Room	H	L
VLC	10cm	Building (ML)	H	L
Ultrasound	1cm-2m	Room	H	H
Audible Sound	meters	Room	L	L
Wi-Fi	1.5m	Building	L	L
Bluetooth	30cm-meters	Building	L	L
ZigBee	25cm	Building	L	H
RFID	1-5m	Room	H	L
UWB	15cm	Building	H	H
Passive				
Geomagnetic	2m	-	L	L
Inertial	2m	-	L	L
Ambient sound	meters	-	L	L
Ambient light	10cm-meters	-	L	L
Computer Vision	1cm-1m	-	L	L

technologies in terms of accuracy, coverage, and costs:

- ❑ *InfraRed*: is composed of an infrared light emitter diode, which emits an infrared signal as bursts of nonvisible light, and a receiving photodiode to detect and capture the light pulses, which are then processed to retrieve the information.
- ❑ *Visible Light Communication (VLC)*: uses visible light to transmit data. Any type of lamp can be used, but LED lights have been found to be the most appropriate. The transmission of data using visible light is possible due to the ability of the light source to be switched on and off again in very short intervals.
- ❑ *Ultrasound*: uses sound frequencies higher than the audible range to determine the user position using the time taken for an ultrasonic signal to travel from a transmitter to a receiver.
- ❑ *Audible Sound*: it is also possible to use audible sound signals to encode information for location systems. Although it would annoy humans nearby, it can be watermarked with an already available sound such as music in malls and other public places in a manner nondetectable to the human ear.
- ❑ *Wi-Fi*: transmits and receives data using electromagnetic waves, providing wireless connectivity within a coverage area. . Three approaches are commonly used for localization: (i) the propagation model of a known antenna can be used, calculating

the distance to a known base; (ii) the relative strength of several known Wi-Fi Access Point devices is used to solve the position by multilateration method; and (iii) fingerprinting which is a pattern of known Wi-Fi Access Point devices with their relative strengths is matched to a database, of course this requires mapping the signal at specific points in the environment.

- ❑ *Bluetooth*: is a wireless communication technology that uses digitally embedded information on radio frequency signals.
- ❑ *ZigBee*: is a wireless communication standard proposed to specifically address the need for low cost implementation of low data rate wireless networks with ultralow power consumption.
- ❑ *Radio Frequency Identification (RFID)*: uses radio waves to make a specialized circuit produce a response containing a unique identifier. An RFID system consists of RFID readers and RFID tags that send data.
- ❑ *UltraWideBand (UWB)*: is based on the transmission of electromagnetic wave forms formed by a sequence of very short pulses using a very big bandwidth. Two different measures can be used in a UWB positioning system to determine the distance between the target and a reference point: Time of Arrival (ToA) and the Time Difference of Arrival (TDoA).
- ❑ *Geomagnetic*: makes use of Earth's natural magnetic field strength and/or orientation, which will be used to determine the position of a person or object. The position estimation is commonly performed through methods such as fingerprinting.
- ❑ *Inertial*: estimates a future position given an initial one and a speed and direction, also called "dead reckoning". To solve the progressive accumulation errors, modern inertial methods use digital accelerometers and gyroscopes, and generally combine their information with other sensors in order to achieve a good performance.
- ❑ *Ambient sound*: localization by sound without embedded information generally takes already available sounds in the environment as characteristic of a given place and uses a portable microphone to infer the location by matching the current sound against a database of known places.
- ❑ *Ambient light*: uses the position of known light sources like lamps to find a location, but without making use of any embedded information.
- ❑ *Computer vision*: makes use of the information collected by cameras and image processing techniques for identifying and tracking objects. There are 2 main approaches, one where the camera is worn by the user or object to be monitored, and

it captures images or video from its perspective, and another where one or several cameras are fixed in the environment in which the subjects will be monitored.

(MAUTZ, 2012) has also investigated such indoor localization technologies and compared their coverage vs. accuracy, as illustrated by Figure 24.

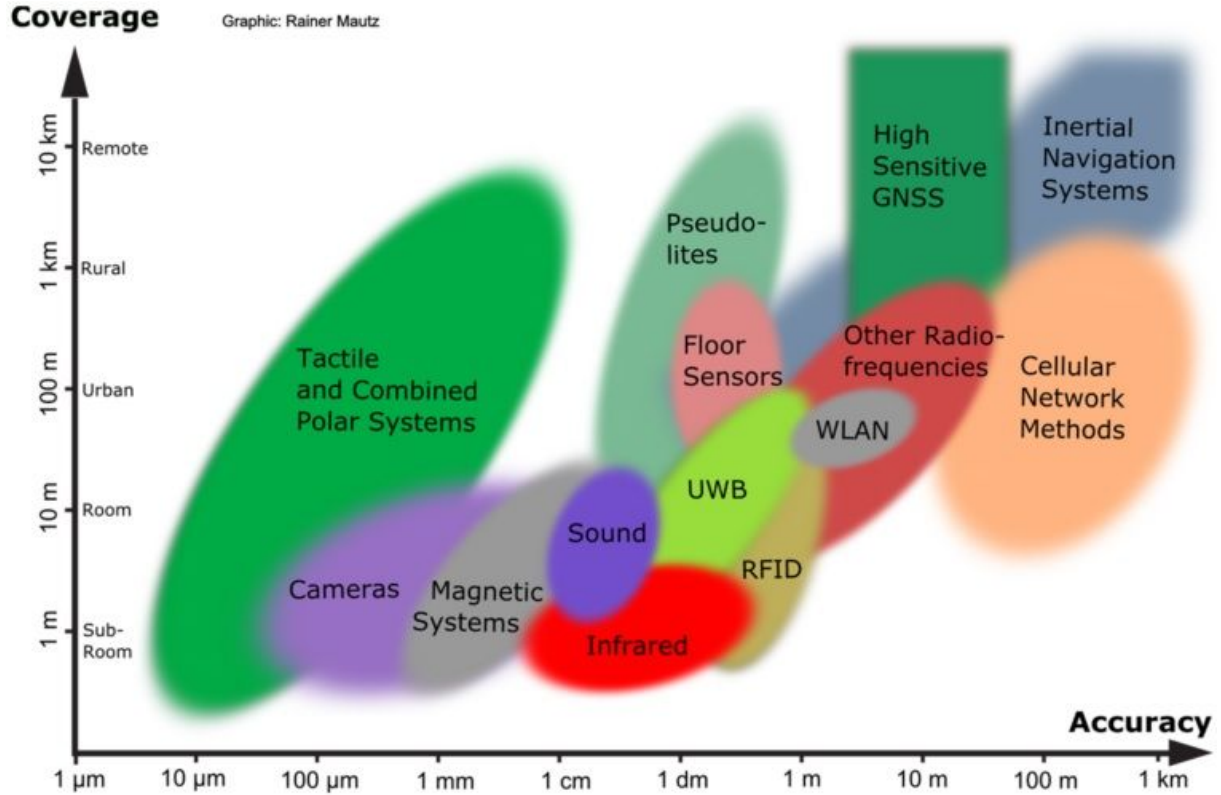


Figure 24 – Overview of indoor technologies in dependence on accuracy and coverage (MAUTZ, 2012).

Some of these technologies and sensors are widely applied to indoor robot localization. One of our goals in the localization field is to provide an accurate positioning system for mobile robots at a low cost.

Usually, a mobile robot relies on observations made by its sensors, estimates motions given by odometry and pre-built internal maps, to reason upon its location in the environment.

There are many map representations in the literature, and the most common are the following:

- *Topological maps*: the world is represented as a connected graph. The nodes in the graph correspond to places of importance and the edges are the connections, such as corridors (ENGELSON; MCDERMOTT, 1992) (KORTENKAMP; WEYMOUTH, 1994) (KUIPERS; BYUN, 1991).

- *Feature maps*: geometric features are used to represent the world. Common examples are points and lines (LEONARD; DURRANT-WHYTE; COX, 1990) (DURRANT-WHYTE, 1988).
- *Grid maps*: instead of restricting the map representation to specific features, typically determined by the user, the world is divided into a grid. Each cell represents a small area/volume of the world and is given a certainty value expressing the chance of that part of the environment being occupied (MORAVEC, 1988) (BUHMANN et al., 1995b) (THRUN; BÜCKEN, 1996).
- *Appearance based methods*: instead of having an intermediate representation of sensor data in the form of grids or features, sensor data is used directly to form a function from sensor data to pose space (WEISS; PUTTKAMER, 1995) (GUTMANN; SCHLEGEL, 1996).

Once there is a map of the environment, algorithms can infer the position of the robot on the map based on observations made by its sensors. In most cases, pose estimation algorithms use Probability Density Functions (PDFs) to handle the pose uncertainty given by motion and sensor noise. In general, these algorithms can be grouped into:

- *Dead-Reckoning*: having an initial estimate and odometric information makes it possible to calculate the motion of the robot. No absolute pose information is used and thus the error in the pose estimate is unbounded.
- *Topological*: just like the map can be represented as a graph, the knowledge about the pose can be kept in a graph structure. The location of the robot is not given as coordinates, but rather as being in a particular place, such as a room.
- *Gaussian PDF*: the most used representation for uncertain robot pose information is the Gaussian PDF, often applied in conjunction with the Kalman Filter.
- *Gaussian Sum PDF*: multiple Gaussians can be used to represent multimodal distributions which arise as soon as the uncertainty becomes too large. A bank of Kalman Filters is one way to implement this approach.
- *Position Probability Grids*: by dividing the pose space into small cells, each representing one possible pose, a discrete approximation of the PDF can be made. With a fine enough discretization of the state space, any functional form of the PDF can be represented.
- *Monte Carlo Methods*: the Monte Carlo methods use a set of samples to represent the PDF. Any PDF can be represented given that the number of samples is finite. Finite sized sample sets allow for an efficient approximation of the PDF.

Two algorithms that belong to these categories have become increasingly popular over the last few years in robotics, the Kalman Filter and Particle Filter. Table 2 shows the advantages and disadvantages of each approach.

The Kalman filter (KF) (MAYBECK, 1982) is an optimal way to fuse observations that follow a Gaussian distribution. One of the reasons the filter performs well is because it uses all available information that it gets, good or bad information, as long as the Gaussian and Linear approaches remain valid. This allows KF to make an overall best estimate of a state. Furthermore, the KF is recursive, which brings the useful property that not all data needs to be kept in storage and re-processed every time when a new measurement arrives. In addition to that, the KF is a data processing algorithm, which is useful for the reason that only knowledge about system inputs and outputs is available for estimation purposes. Variables of interest cannot be measured directly. Due to the use of Gaussian distribution, this filter is uni-modal or a parametric filter, in other words, the probabilistic map only has one bump.

Particle filter (PF) (DELLAERT et al., 1999), also known as Sequential Monte Carlo localization (stated in Section 2.1.2), exploits the idea of representing the posterior distribution through a finite set of particles. When a new observation arrives, the particles are updated in order to represent the new posterior. During this update, the particles with more importance survive to the next iteration, whereas the particles with lowest importance are eliminated. The closer a particle is to the estimated position, the more importance that particle has.

Table 2 – Advantages and disadvantages of the filters techniques.

Technique	Pros	Cons
Kalman Filter	Continuous; Computation time only increases quadratically with system dimension;	Uni-modal; hard to program
Particle Filter	Multi-modal; resource-adaptive algorithm; easy to program; Continuous;	Computation time increases exponential with system dimension;

Kalman Filter based techniques have proven to be robust and accurate for keeping track of the robot's position. However, a Kalman Filter cannot represent ambiguities and lacks the ability to globally (re-)localize the robot in the case of localization failures. Although the Kalman Filter can be amended in various ways to cope with some of these difficulties, recent approaches have used richer schemes to represent uncertainty, moving away from the restricted Gaussian density assumption inherent in the Kalman Filter (DELLAERT et al., 1999). For this reason, we believe that Particle Filter is more

suitable for our work on indoor localization, as we tackle the “Initialization” and “Robot Kidnapped” problems.

As mentioned before, the localization in indoor human environments is very challenging as they are composed by complex static structures and dynamic objects that can move over time. Many approaches have been proposed to solve this problem using various sensors, observation models, map types and algorithms. Some of them are listed below.

Indoor localization techniques that use laser range finders and depth cameras, like Microsoft Kinect, are commonly used in robotics (BISWAS; Veloso, 2012). Unfortunately, indoor human environments consist of many movable objects like chairs and tables, and moving objects like humans, that can occlude features used by such techniques.

This problem is treated in (BISWAS; Veloso, 2014) using an Episodic Non-Markov Localization algorithm. In this approach, observations captured in the environment are classified as Long Term Features, Short Term Features, or Dynamic Features. Long Term Features, such as walls, belong to the static map of the environment, while Short Term Features, like furnitures, are used to refine the localization estimate. But crowded areas still an open problem, as Dynamic Features (like humans) cannot be related to the map and they might occlude other features.

Since the robot can be surrounded by objects that block laser range finders and depth cameras, other approaches like Star Gazer systems offer an alternative solution to the indoor localization problem (LEE; SONG, 2007). Usually, these systems use markers mounted on the ceiling. Then, sensitive cameras positioned on top of the robot observe different point patterns that represent unique locations in the environment. Although these systems can be very accurate, markers should be available all over the place.

On the other hand, wireless based techniques, such as WiFi signal strength from Access Points (APs), are applicable anywhere there is dense WiFi network, which is the case of many human environments today (BISWAS; VELOSO, 2013). Many aspects can impact on the WiFi localization accuracy, such as: quality of APs, quantity of APs, distribution of APs along the environment, noise on the radio frequency, and others. Therefore, the minimal error is greater than that of other approaches. However, it is not affected by changes of dynamic objects as is Kinect localization.

In our work, we aim to use the benefits of Kinect and WiFi localization in a single positioning algorithm. Both technologies are cheap and have good accuracy for indoor localization purpose. In such algorithm, the WiFi localization estimates the global location of the robot as APs are usually well spread along the environment. This global location determines a perimeter for the Kinect approach. Then, the estimate given by the Kinect localization is taken as the robot’s current location, since it has a better local precision.

Both localization approaches use Particle Filter to maintain a set of hypotheses of the robot’s location in real time. Regarding the Kinect localization, it was used the standard system available on our robot. For the WiFi localization, we extended the work

of (BISWAS; VELOSO, 2010) for indoor open areas. We implemented new methods for localization and mapping, including the concept of irregular WiFi grid maps, different interpolation strategies for estimating WiFi signal continuously in the grid, and algorithms to replace particles and infer the robot's pose.

We carried out different experiments. In the first one, we compared the localization accuracy along a path within a indoor area using Kinect, WiFi and Kinect+WiFi. For the second, we tested the ability of our robot to recover its positioning autonomously in case it is lost (Initialization and Kidnapped Robot Problem). The last experiment evaluated the accuracy of WiFi localization with different amounts of APs.

A robotic platform, called Turtlebot 2 (Figure 25), was used for these experiments. This robot consists of an Yujin Kobuki base, wheel encoders, wheel drop sensors, an integrated gyroscope, bump sensors, cliff sensors, a 2200 mAh battery pack, a Microsoft Kinect sensor, an Intel NUC i7, a fast charger, a WiFi dongle, and a hardware mounting kit attaching everything together.



Figure 25 – Turtlebot 2: robotic platform used in our indoor localization experiments with Kinect and WiFi devices.

The remainder of the Chapter is organized as follows. A localization approach based on Kinect sensor is shown in Section 3.1. Another localization approach, based on WiFi sensor, is described in Section 3.2. Both sections, 3.1 and 3.2, present algorithms for mapping and localization using WiFi and Kinect devices, and also respective related works. Section 3.3 presents how we use both Kinect and WiFi localizations in a single algorithm. Experiments using all three approaches and their results are provided in Section 3.4. Finally, a summary is presented in Section 3.5.

3.1 Kinect Localization

Depth cameras are revolutionizing many fields in robotics. They are an alternative solution to laser range finders and stereo vision systems. But only recently, with the advent of Microsoft Kinect (Figure 26), depth cameras started providing decent quality depth information of the surrounding environment at a highly competitive price.

Originally designed by Microsoft as an add-on for the Xbox 360 video game console, Kinect is finding many applications outside of the gaming scope, specially in robot localization. It consists of an infrared laser emitter, an infrared camera and a RGB camera, with which it produces Depth in addition to color of each pixel of a 640 x 480 frame, also known as RGB-D, at a frame rate of up to 30 fps.



Figure 26 – Kinect sensor

3.1.1 Related Work

The robotics community has developed many techniques for indoor localization and mapping using the Kinect sensor.

(HENRY et al., 2010) introduces RGB-D Mapping, a framework that can generate dense 3D maps of indoor environments despite the limited depth precision and field of view provided by RGB-D cameras.

In (BISWAS; VELOSO, 2012) is presented an algorithm to filter depth images into point clouds corresponding to local planes. Then, an observation model matches the plane filtered points to lines in the existing 2D maps for localization.

(JALOBÉANU et al., 2015) introduce a reliable Kinect based navigation in large indoor environments. This approach includes methods for: (1) processing depth frames, (2) continuous incremental mapping to generate reliable local maps from extremely noisy readings, and (3) integrating local maps into a navigation pipeline.

A topological map-building-based method for localization and navigation is proposed in (CHENG; CHEN; LIU, 2015). This method generates topological maps identifying corridors using a Bayesian classifier on depth curves provided by a Kinect sensor. Based on the map and a Markov localization method, the robot can then localize itself and navigate freely in the indoor environment.

Another method of robot navigation in corridors is proposed in (QIAN et al., 2015), which combines the use of geometrical features of the environments and probabilistic occupancy information obtained from a Kinect sensor equipped on a mobile robot. The

proposed method does not require a previously established map of the corridor environment and is suitable to be applied in unknown environments.

In our Kinect localization experiments, we used the GMapping (GRISETTI; STACHNISS; BURGARD, 2007) algorithm to generate laser maps of our indoor environment. Following this, the Adaptive Monte Carlo Localization (AMCL) (FOX, 2001) algorithm was used to estimate the robot's pose during the navigation. We used the standard implementation of both algorithms available on ROS packages of our Turtlebot.

The next sections give an overview of these algorithms and how we used them in our experiments.

3.1.2 Mapping

Most robot localization approaches require a pre-built map to accurately estimate robots position in the environment. Usually, such maps are acquired by the robots using range sensors (e.g. laser sensors, 3D sensors, ultrasonic sensors) along with control sensors (odometry) that enable them to perceive the environment.

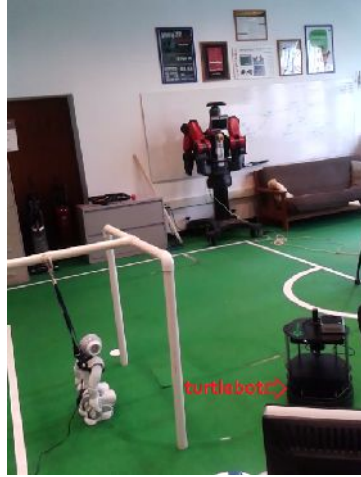
The mapping phase is very complex, because for localization a robot needs a map and for acquiring a map a robot needs to estimate its location continuously. This problem is often referred to the Simultaneous Localization and Mapping (SLAM) problem (THRUN et al., 2002).

The main challenges related to the SLAM problem are: the accumulative error in odometry, the high dimensionality of the entities that are being mapped, the way of determining if sensor measurements taken at different points in time correspond to the same physical object in the environment, the changes on the environment during mapping, and the exploration strategy to cover the environment.

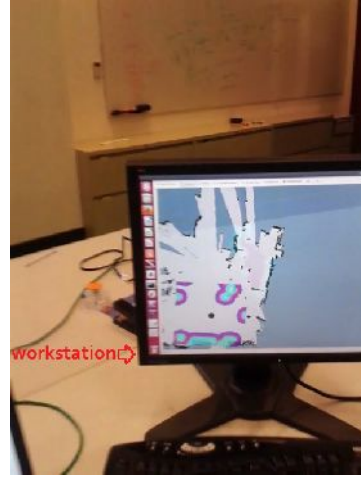
Maps also need a formal representation, such as metric (grid based or featured based), topological, or hybrid, so they can be interpreted later by robots. One popular map representation in robotics is the occupancy grid (MORAVEC, 1988), in which range sensors are used to detect the distance to obstacles whose estimated locations will be stored into a data structure (e.g./ 2D array). When the robot is moving, it keeps updating this data structure, marking cells as occupied, free, or unknown. In this process, filtering techniques are applied, such as Extended Kalman Filter (EKF) and Rao-Blackwellized Particle Filter (RBPF), to improve the estimation of obstacle while moving and removing noise and errors from measurements.

The most widely used laser-based SLAM algorithm in robotics is GMapping, which is an implementation of the RBPF SLAM approach proposed by (GRISETTI; STACHNISS; BURGARD, 2007). This algorithm introduces two improvements to the standard Particle Filter approach for the SLAM problem, an adaptive resampling technique to minimize the particle depletion, and an accurate distribution that takes into account the movement and also the most recent observations of the robot.

We carried out our localization experiments in a Laboratory located on the 3rd floor of the Gates and Hillman Centers (GHC) building at Carnegie Mellon University (CMU). We generated a laser map of the Laboratory running GMapping algorithm on our Turtlebot. Figure 27 (a) shows the Turtlebot scanning the Laboratory with the Kinect sensor. During the mapping, the Turtlebot was teleoperated remotely from our workstation, which also plots the current state of the occupancy grid, as depicted by Figure 27 (b).



(a) Turtlebot in the Laboratory.



(b) Workstation.

Figure 27 – Mapping the Laboratory with Turtlebot using Kinect sensor.



The final laser map of the Laboratory is illustrated by Figure 28. In this image, white pixels represent unknown cells of the occupancy grid that could not be scanned, gray pixels are free cells where the robot can traverse, and black cells are occupied cells representing obstacles found during the mapping, such as walls, tables, chairs, cabinets, and other objects.

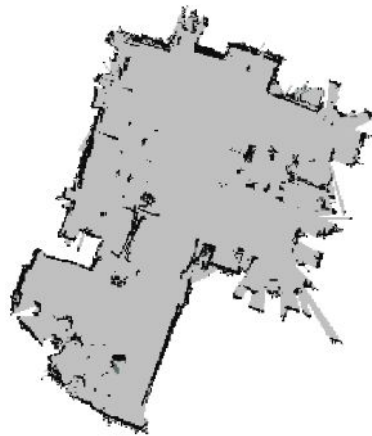


Figure 28 – Laser Map of the Laboratory.

Figure 29 (a) depicts the Floor Plan of the 3rd floor in the GHC building at CMU, where the limits of the Laboratory are dashed in red. In the Figure 29 (b), it is shown how the laser map fits on the Floor Plan.



Figure 29 – Floor Plan of the 3rd floor in the Gates and Hillman Centers building at Carnegie Mellon University.

The quality of the generated map greatly depends on quality of range sensors and odometry sensors. For the Turtlebot, it has either the Kinect or Asus Xtion Live Pro camera with only 4 meters of range, and only 57 degree of laser beam angle, which render is not appropriate for scanning large and open space environments. Thus, if we use Turtlebot to build the map for a small environment where walls are not far from each others, it is likely that we will get an accurate map.

However, if we build the map for a large and open space environment, then it is likely that our map will not be accurate at a certain point of time. In fact, in large and open space environment, we need to drive the robot for long distance, and thus, the odometry will play a more crucial role in building the map. Considering the fact that the odometry of the Turtlebot is not reliable and is prone to errors, these errors will be cumulative over time and will quickly compromise the quality of generated maps.

3.1.3 Localization Algorithm

Once the map is built, robots can use it to estimate their positions in the environment using different localization methods. Particle Filter (PF) based approaches (STACHNISS; BURGARD, 2014) have been the preferred method when the map is generated

as an occupancy grid. They are relatively easy to implement and able to exploit all the measurements available in a range scan. The main problem with PFs is the computational efficiency to maintain a large number of random distribution of particles throughout the state space in order to estimate the location with acceptable accuracy.

The Adaptive Monte Carlo Localization (AMCL) is a PF based algorithm that implements KLD-sampling (FOX, 2001) to manage the number of particles during the localization. The key idea of the KLD-sampling method is to bound the approximation error introduced by the sample-based representation of the PF. The name KLD-sampling is due to the fact that the approximation error is measured by the Kullback-Leibler Distance. This adaptation chooses a small number of particles if the density is focused on a small part of the state space, and it chooses a large number of particles if the state uncertainty is high.

In our experiments with Kinect localization, we used the standard implementation of AMCL available on ROS packages of our Turtlebot. The AMCL process is composed by the following steps: *initialization*, *predict*, *update*, *resampling*, and *estimate* (Figure 30).

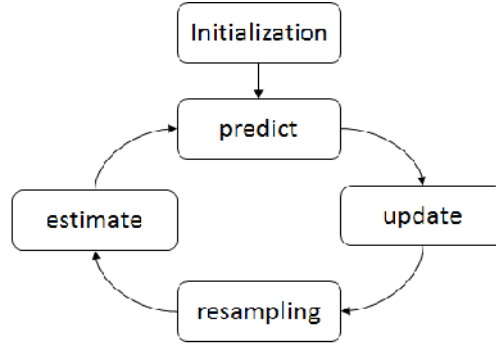


Figure 30 – AMCL process.

- *initialization*: to use AMCL for Kinect localization, it is necessary to give a laser map of the environment as input. Then, we can either set the robot to some position on the map, in which case we are manually localizing it, or we could very well make the robot start from no initial estimate of its position. Usually, we use the RViz interface, also available on ROS, to set the start location of the robot to avoid mistakes as the robot must compare its single initial observation against many similar areas on the map. This also yields a faster convergence of its particles. So, given this initial position, the AMCL randomly spreads the particles from a normal distribution centered on that area.
- *predict*: as the robot moves, the algorithm moves the particles according to the odometry to predict the robot's position after the motion command. Odometry is the use of data from moving sensors to estimate (not determine) change in position

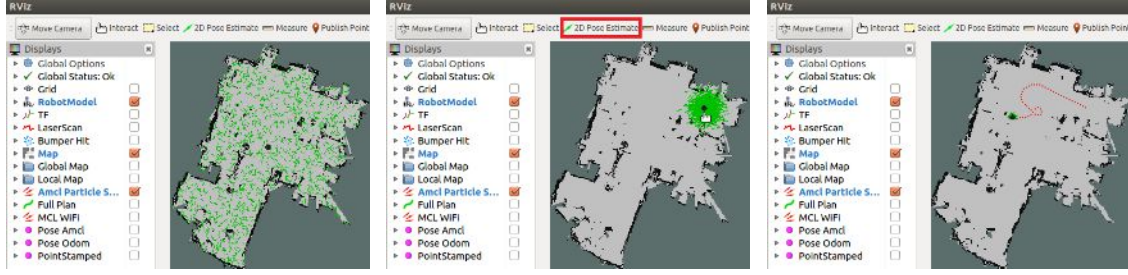
over time. As these data can contain noise, random variations, and other inaccuracies, the odometry estimation in ROS, specially in the Turtlebot, is computed by an Extended Kalman Filter (EKF). Using wheel sensor data, bump sensor data, gyroscope data, robot structure parameters and others, the EKF produces values that tend to be closer to the real values.

- *update*: when the Kinect laser scans are received and transformed into distances and angles, they are compared with the data collected and saved earlier. The weight of a particle is updated by the probability of observing these distances and angles given the particle's location. The particles with the closest readings to the robot will be given a higher weights and will be chosen as possible poses for the robot itself.
- *resampling*: only particles with higher weights will be included in the process of localization, this will make the number of possible particles less and less with time and the robot's pose will be guessed correctly. On the other hand, particles with lower weights should be replaced by others close to the existing ones with higher probability. Also, if the uncertainty is still high, few random uniformly distributed particles should be added to help the robot recover itself in cases where it has lost track of its position. In those cases, without these random particles, the robot will keep on resampling from an incorrect distribution and will never recover. The reason why it takes the filter multiple sensor readings to converge is that we might have disambiguities due to symmetry on the map. To control the number of particles, the KLD-sampling method generates particles until their number is large enough to guarantee that the K-L Distance between the maximum likelihood estimate and the underlying posterior does not exceed a pre-specified bound.
- *estimate*: at any given time, the particle with the highest probability is picked as the current pose of the robot. Although it might not be the actual pose of the robot, it is likely to be close enough for localization purposes.

Figure 31 shows the RViz interface. RViz is a 3D visualizer for the Robot Operating System (ROS) framework. In a lot of cases, RViz is run on a visualization workstation to monitor a robot. It is able to render maps, robots, particles, paths, images from onboard cameras, and many other data types published via ROS messages.

In our experiments, Kinect-based particles are represented by green arrows which mark the location and orientation of the robot's hypotheses. The black dot is the robot's current location computed by AMCL.

Figure 31 (a) illustrates the initialization of AMCL using random particles all over the map. Then, in the Figure 31 (b), we used RViz interface to set the initial position of the robot by clicking on the map. After that, AMCL resets all particles to that area based on a normal distribution centered on the selected spot. If the robot stays at the



(a) Random particles.

(b) Setting initial pose.

(c) Navigating.

Figure 31 – Kinect localization on RViz interface.

same initial position, it will not receive different observations from the Kinect device. Therefore, particle weights will remain the same and the algorithm will not converge. Figure 31 (c) depicts the convergence of the particles while the robot is navigating and sensing different parts of the environment. After performing a path (dashed in red), the robot is well localized and its particle set is more centred and has less particles.

3.2 WiFi Localization

Although Kinect localization is very accurate in indoor environments, in general, it cannot handle changes on the map caused by moving or movable objects. An alternative solution is to use some technique that can be easily implemented in indoor human environments and it is not affected by this problem.

With the increasing of Wireless Local Area Networks (WLAN) in human environments, considerable work has been done on using signal strength measurements from WiFi Access Points (APs) (Figure 32 (a)) for indoor mapping and localization. Usually, these techniques use Radio-Frequency (RF) fingerprinting, where Received Signal Strength (RSS) measures in some RF band are collected using WiFi receivers, such as WiFi dongles (Figure 32 (b)), and compared to a map of previous collected signals.



(a) Access Point (AP).



(b) WiFi dongle.

Figure 32 – WiFi emitter and receiver devices.

3.2.1 Related Work

For instance, (MIROWSKI; HO; WHITING, 2014) introduce methods for creating a WiFi map from RF signals collected along robotic or pedestrian trajectories. The first method consists of naive fixed size grids, where WiFi samples collected by a freely moving robot and falling into a specific spatial location cell into a fingerprint. The next one is a clustering of RSS top-down into cells of increasingly smaller size using decision trees. Finally, in the last method is proposed an iterative bottom-up approach that aggregates smaller-size fingerprint cells into larger grid cells.

In (WU; JEN, 2014) is presented a localization method for mobile robots based on RSS in indoor WLANs. An observation likelihood model is accomplished using kernel density estimation to characterize the dependence of location and RSS. Based on the measured RSS, the robot's location is dynamically estimated using an Adaptive Local Search Particle Filter (ALSPF).

The task of acquiring WiFi maps can be exhausting as most of them are created manually by driving robots through the environment and collecting signal strength measurements along with precise position information. Furthermore, it is necessary to repeat the mapping periodically to adapt to potential changes in the radio environment. (PALANIAPPAN et al., 2011) describe the design and working of a robotic platform capable of conducting repeated surveying of an indoor environment to update the signal maps as frequently as required. Also the robot serves as a testbed to collect data in controlled conditions, such as under uniform motion and with accurate positioning.

(BISWAS; VELOSO, 2010) propose a WiFi signal strength based localization algorithm that uses a parametric graph composed of discrete points for which WiFi signal strengths are collected. During navigation, a Particle Filter (PF) estimates the robot's location in five steps. (1) Predict: given new odometry data for the motion of the robot, the particles are moved using a motion model, where the robot's linear and angular motions are modeled as having Gaussian error. (2) Update: the weights of the particles are updated based on the latest WiFi signal strength observation using a probabilistic perceptual model that applies a linear interpolation to estimate signal strength values on unknown points between discrete known points on the graph. (3) Constrain: particles are constrained to the edges of the graph. (4) Resampling: particles are resampled based on Sensor Resetting Localization method. (5) Infer Location: K-Means clustering is performed on the particle set taking into account the weights of the particles, where the cluster with the largest weight represents the robot's location. This approach was applied to the Collaborative Robots (CoBots) that behave as service robots at Carnegie Mellon University (CMU).

Our WiFi localization is based on that previous approach of (BISWAS; VELOSO, 2010). The main difference is that CoBots use graph representation for navigation and localization in the corridors of the buildings. Thus, hallways can be seen as edges and

discrete known locations as vertices. Instead, we introduce modifications on mapping and localization processes, so the robots can work not only in corridors, but also in indoor open areas, such as atria, lobbies, laboratories, offices, etc.

In the next sections, we show how we generate a WiFi map of the Laboratory and how we use it to localize our Turtlebot by the means of a Particle Filter.

3.2.2 Mapping

A traditional way of building WiFi maps is by collecting repeated RSS measurements at predefined locations, represented as points in the form of (x, y) cartesian coordinates, in a 2D environment. In our approach, the robot collects RSS readings from all reachable Access Points (APs) at discrete points for a predetermined duration. Then, it computes mean and standard deviation values for each pair of discrete point and AP, which compose the WiFi map.

Thus, the map \mathbb{M} of the environment has the following structure $\mathbb{M} = \langle \mathbf{P}, \mathbf{A}, \mathbf{M}, \mathbf{D} \rangle$. The set \mathbf{P} consists in all discrete points for which we collect RSS readings. \mathbf{A} is the set of all accessible APs in the environment. \mathbf{M} and \mathbf{D} are matrices representing the RSS means and standard deviations. They are a combination of $\mathbf{P} \times \mathbf{A}$, i.e., the element \mathbf{M}_{ij} of \mathbf{M} is the mean RSS of AP a_j as measured from the discrete point p_i . Likewise, the element \mathbf{D}_{ij} of \mathbf{D} is the observed standard deviation of the RSS of AP a_j as measured from the discrete point p_i .

Tables 3 and 4 illustrate values for the matrices \mathbf{M} and \mathbf{D} . The mean RSS values stored in the matrix \mathbf{M} are given in dBm, ranging from -20 (strong signal) to -90 (weak signal). Values in the matrix \mathbf{D} are the standard deviation based on the mean value and the reading set. Therefore, a robot can estimate its location during the navigation by comparing its current RSS readings from some APs to a normal distribution based the matrices \mathbf{M} and \mathbf{D} .

\mathbf{P}/\mathbf{A}	a_1	a_2	...	a_n
p_1	-23	-34	...	-88
p_2	-27	-45	...	-61
...
p_m	-89	-63	...	-78

Table 3 – Mean Matrix (\mathbf{M}).

\mathbf{P}/\mathbf{A}	a_1	a_2	...	a_n
p_1	0.6	2.8	...	0.2
p_2	1.3	1.5	...	4.1
...
p_m	3.5	3.6	...	2.9

Table 4 – Standard Deviation Matrix (\mathbf{D}).

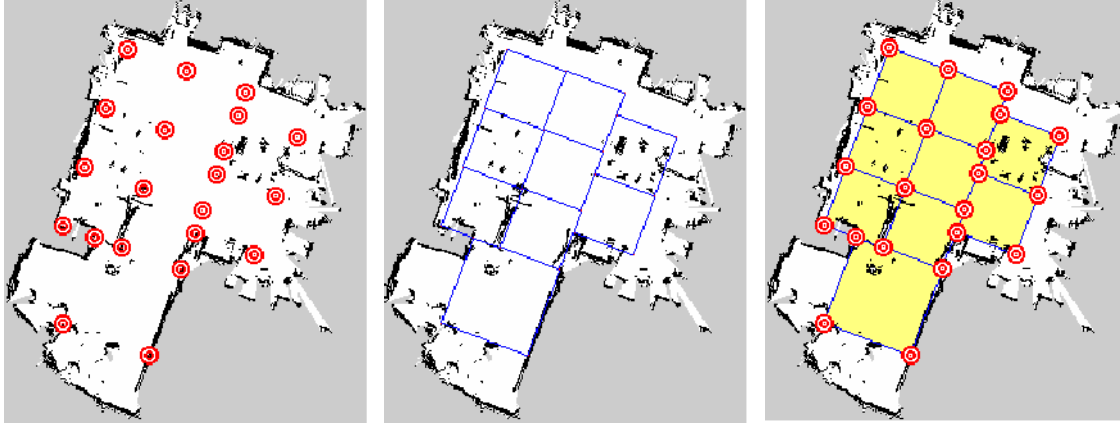
In this sense, WiFi maps consist of RSS values collected from some discrete points. Since robots are navigating in a continuous space, how can they estimate their locations at points where they did not collect any RSS values during the mapping? Usually, in this case, interpolation techniques are applied to estimate values for unknown points based on discrete known points.

There are many interpolation techniques for 2D spaces like linear, bilinear, nearest neighbor, and bicubic interpolation. However, accurate interpolation techniques have high computational cost, which is not desired in real time applications as robot localization, and low cost interpolations are not acceptable as they may increase the localization error. To deal with this tradeoff, in our work, depending on the location of a given point on the map we use different types of interpolation to estimate the RSS mean and standard deviation.

We map our test environment in a grid fashion, where RSS values are collected at points in the corner of each cell. But depending on the shape of the environment, it is not possible to generate perfect grids with cells of same size and orientation. Thus, we used the concept of irregular grids to better fit them on any environment.

In our experiments, the WiFi map of the Laboratory was built based on the laser map generated with Kinect sensor (Section 3.1). Using the coordination system of the laser map, we selected some points in order to form an irregular grid. Then, at each point, our Turtlebot collected RSS readings for 5 minutes.

Figure 33 (a) shows the selected points where RSS were collected, segments linking discrete points are depicted by Figure 33 (b), and Figure 33 (c) presents the full grid on the map.



(a) Discrete Points.

(b) Segments between Points.

(c) Full Irregular Grid.

Figure 33 – Irregular Grid for WiFi Mapping.

Therefore, for calculating the RSS mean and standard deviation for a given point, we first check wheter it lies on any of the discrete points of \mathbf{P} . If so, we can take the respective values from \mathbf{M} and \mathbf{D} . Otherwise, if this point belongs to some segment, we use both points of the segment to perform a linear interpolation and estimate the RSS mean and standard deviation. But if the point is inside of a cell, we use all four points of the cell to calculate RSS mean and standard deviation values by means of a bilinear interpolation.

3.2.3 Localization Algorithm

Given the WiFi map of the environment, we estimate the robot's location in real-time using a Particle Filter (PF). The algorithm maintains a set of particles \mathbb{P} to represent the distribution of where the robot could be, i.e., each particle is a hypothesis of a possible location. A particle $p_i \in \mathbb{P}$ has the following properties $p_i = \langle l_i, \theta_i, w_i \rangle$. These properties are described below:

- l_i : the cartesian location (x_i, y_i) of the particle on the map;
- θ_i : the orientation of the particle;
- w_i : the weight assigned to the particle.

The localization process is repeated automatically or whenever a new odometry data is received. This process involves four different steps: *predict*, *update*, *resampling*, and *infer location*. Basically, the *predict* step moves the particles according to a new odometry data. The *update* step is responsible for extracting a WiFi observation and updating the weight of each particle. Then, we resample particles in the *resampling* step. Finally, the *infer location* step estimates the robot's location based on the particle set.

In our tests in the Laboratory, we applied a passive “sniffing” technique for gathering WLAN received signal strength (RSS) to create WiFi observations, and also to collect data during the mapping phase. To do that, a WiFi dongle (Linksys Dual-Band Wireless-N USB Adapter - AE3000) was mounted on our Turtlebot and used on monitoring mode to sniff out the WiFi network. Our sniffer *script* uses *tShark* commands to capture WiFi information, such as the IP address of an Access Point, its RSS value, and its Radio Frequency Channel. We choose three channels, 1, 6, and 11 as they do not overlap each other. In these channels, 124 APs were discovered, few of them belong to the Laboratory, some of them were nearby, and the rest were spread along the building. An advantage of this approach is that, as we are not performing a triangulation to estimate the robot's position, it is not necessary to know the real location of the APs.

During the experiments, it was noticed that distant APs are more likely to have noise. Also, the time required to perform the localization algorithm using all APs was too long. To solve these issues, we filtered the most relevant APs before we use them in the localization. To do that, we reduce the APs set by selecting the AP with highest mean RSS for each discrete point. To avoid repetition, if an AP was already taken, we picked the next available one. As a result, we reduced the set to 21 APs.

The following sections describe in detail how each step of our localization process works.

3.2.3.1 Predict

The odometry data, denoted as $(\Delta_x^{odo}, \Delta_y^{odo}, \Delta_\theta^{odo})$, is read as incremental changes in the 2D robot's pose. Given the prior pose (x, y, θ) of each particle $p_i \in \mathbb{P}$, the *predict* step computes the new poses (x', y', θ') after the odometry reading using Equation 2. The vector $(\epsilon_x, \epsilon_y, \epsilon_\theta)$ refers to the Gaussian error added to the new pose.

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \cos(\theta + \frac{\Delta_\theta^{odo}}{2}) & -\sin(\theta + \frac{\Delta_\theta^{odo}}{2}) & 0 \\ \sin(\theta + \frac{\Delta_\theta^{odo}}{2}) & \cos(\theta + \frac{\Delta_\theta^{odo}}{2}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\theta^{odo} \end{pmatrix} + \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_\theta \end{pmatrix} \quad (2)$$

3.2.3.2 Update

In order to update the weight of each particle $p_i \in \mathbb{P}$, whenever there is a new motion or not, we need a Perceptual Model. The Perceptual Model is used to calculate the probability of making a particular signal strength measurement S at a location l , $P(S|l)$.

During the execution, the robot spends a time t listening to the APs of the environment to fill an observation set $S = [S_1, S_2, S_3, \dots, S_{|\mathbf{A}|}]$. In our experiments, t was set to 1 second, which permits the collection of almost 30 RSS readings, from different APs or not. Thus, S is filled with the average RSS of each AP, where S_i is the average RSS measured from the AP i .

Once we have an observation S , the robot should be able to calculate the probability $P(S|l)$ from any location l in the environment, not only in the discrete points where RSS were collected during the off-line mapping. Therefore, using the discrete values of mean and standard deviation of each pair of discrete point and AP, we created a continuous Perceptual Model composed by different strategies (**In Range**, **Linear Interpolation**, and **Bilinear Interpolation**) for estimating mean and standard deviation values for any location l on the map in order to calculate the probability $P(S|l)$ by the means of Gaussian Distributions.

Figure 34 illustrates an hypothetical example where points p_1, p_2, p_3 , and p_4 compose a cell in the WiFi grid map.

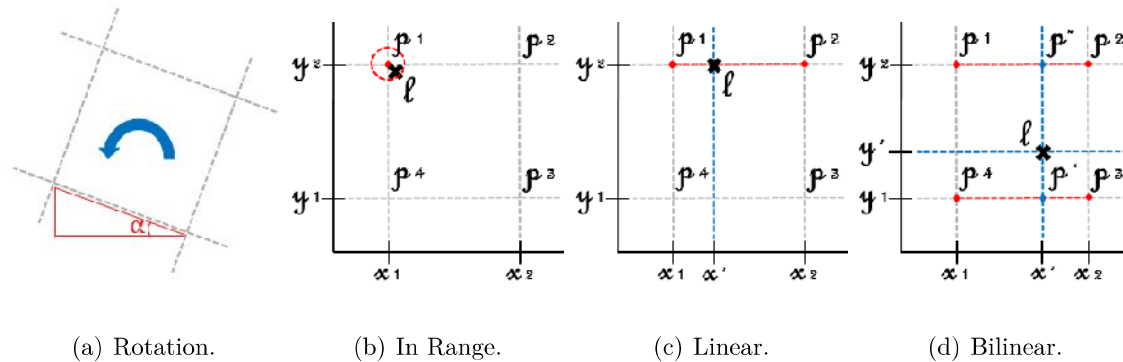


Figure 34 – Estimating Mean and Standard Deviation Values for RSS on the WiFi Grid.

As described in Section 3.2.2, the map \mathbb{M} has the matrices \mathbf{M} and \mathbf{D} which contain the WiFi signal strength means and standard deviations of every Access Point $a_k \in \mathbf{A}$ measured from each discrete point $p_i \in \mathbf{P}$. With this map, we model the WiFi signal strength mean and standard deviation as being piecewise linear along the map, with Gaussian noise.

To better understand the estimation techniques, it is necessary to introduce the following formalisms. Let \mathbf{M}^l denote the vector of mean signal strengths of every Access Point as seen from location l . The component \mathbf{M}_k^l of the vector \mathbf{M}^l is the mean WiFi signal strength of Access Point $a_k \in \mathbf{A}$ as seen from location l . Similarly, let \mathbf{D}^l denote the standard deviations of the signal strengths of each Access Point as seen from location l , with \mathbf{D}_k^l being the standard deviation of the signal strength of Access Point a_k as seen from location l .

Given a location l and the WiFi grid map \mathbb{M} , the Perceptual Model works as follows.

□ **In Range Estimation:** first it checks whether l lies on any point $p_i \in \mathbf{P}$, or if it is near to some point in a certain range. Considering the scenario illustrated by Figure 34 (b), the location l is close to the point p_1 located in a corner of some cell of the WiFi grid. In such a situation, the Perceptual Model uses the mean and standard deviation of p_1 for the location l . Therefore, the mean signal strengths vector $\mathbf{M}^l = [\mathbf{M}_1^l, \mathbf{M}_2^l, \dots, \mathbf{M}_{|\mathbf{A}|}^l]$, and the standard deviations vector $\mathbf{D}^l = [\mathbf{D}_1^l, \mathbf{D}_2^l, \dots, \mathbf{D}_{|\mathbf{A}|}^l]$ at location l and Access Point k are given by:

$$\mathbf{M}_k^l \approx \mathbf{M}_k^1 \quad (3)$$

$$\mathbf{D}_k^l \approx \mathbf{D}_k^1 \quad (4)$$

□ **Linear Interpolation Estimation:** when the location l is not close to any point, the Perceptual Model verifies if it belongs to a segment in some cell of the grid. Once the right segment is found, a Linear Interpolation is performed using the end points of the segment. To facilitate the calculation of both interpolation techniques, the algorithm rotates the cell to the x-axis as shown in Figure 34 (a).

Hence, following the example of Figure 34 (c), the mean signal strengths vector \mathbf{M}^l and the standard deviations vector \mathbf{D}^l at location $l = (x', y_2)$, between the points $p_1 = (x_1, y_2)$ and $p_2 = (x_2, y_2)$, and Access Point a_k are given by:

$$\mathbf{M}_k^l \approx \frac{[(x_2 - x') \times \mathbf{M}_k^1] + [(x' - x_1) \times \mathbf{M}_k^2]}{(x_2 - x_1)} \quad (5)$$

$$\mathbf{D}_k^l \approx \frac{[(x_2 - x') \times \mathbf{D}_k^1] + [(x' - x_1) \times \mathbf{D}_k^2]}{(x_2 - x_1)} \quad (6)$$

The same idea is applied when the location l lies on other segments of the cell.

□ **Bilinear Interpolation Estimation:** finally, if the location l is neither on a point nor on a segment, the Perceptual Model finds the most appropriate cell to perform a Bilinear Interpolation.

The key idea is to perform Linear Interpolation first in one direction, and then again in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location.

Given the means and standard deviations of the points $p_1 = (x_1, y_2)$, $p_2 = (x_2, y_2)$, $p_3 = (x_2, y_1)$, and $p_4 = (x_1, y_1)$ of our example, depicted by Figure 34 (d), the Perceptual Model first do Linear Interpolation in the x-direction for each Access Points $a_k \in \mathbf{A}$ using inner points p' and p'' extracted from the location l . This yields:

$$\mathbf{M}_k^{p'} \approx \frac{[(x_2 - x') \times \mathbf{M}_k^4] + [(x' - x_1) \times \mathbf{M}_k^3]}{(x_2 - x_1)} \quad (7)$$

$$\mathbf{M}_k^{p''} \approx \frac{[(x_2 - x') \times \mathbf{M}_k^1] + [(x' - x_1) \times \mathbf{M}_k^2]}{(x_2 - x_1)} \quad (8)$$

$$\mathbf{D}_k^{p'} \approx \frac{[(x_2 - x') \times \mathbf{D}_k^4] + [(x' - x_1) \times \mathbf{D}_k^3]}{(x_2 - x_1)} \quad (9)$$

$$\mathbf{D}_k^{p''} \approx \frac{[(x_2 - x') \times \mathbf{D}_k^1] + [(x' - x_1) \times \mathbf{D}_k^2]}{(x_2 - x_1)} \quad (10)$$

Then, the Perceptual Model proceed by interpolating in the y-direction to obtain the desired estimate:

$$\mathbf{M}_k^l \approx \frac{[(y_2 - y') \times \mathbf{M}_k^{p'}] + [(y' - y_1) \times \mathbf{M}_k^{p''}]}{(y_2 - y_1)} \quad (11)$$

$$\mathbf{D}_k^l \approx \frac{[(y_2 - y') \times \mathbf{D}_k^{p'}] + [(y' - y_1) \times \mathbf{D}_k^{p''}]}{(y_2 - y_1)} \quad (12)$$

With these estimation techniques, we can model the WiFi signal strength distribution of an Access Point at any location as a Gaussian Distribution using the means and standard deviations.

Suppose there are M locations representing positions of particles on the map. To compute the weights for all particles at locations $l_1, l_2, l_3, \dots, l_{|M|}$ given a new observation $S = [S_1, S_2, \dots, S_{|\mathbf{A}|}]$, it is necessary to calculate the joint probabilities of all Access Points $a_k \in \mathbf{A}$, as illustrated by Figure 35.

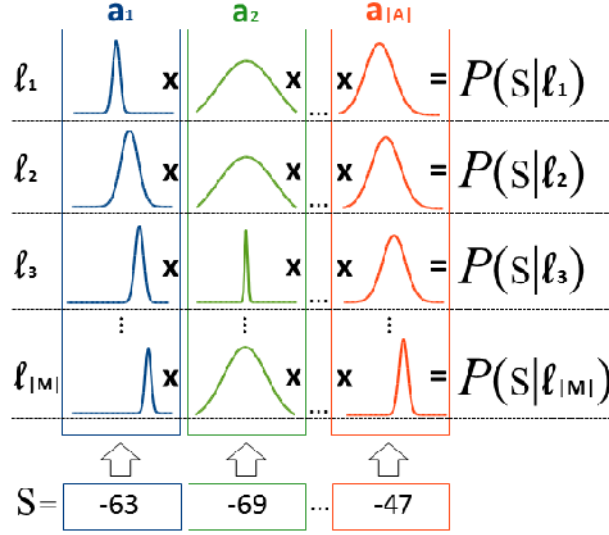


Figure 35 – Perceptual Model: Calculating Join Probabilities.

However, probabilities are often small numbers, and in this case, we have to multiply all of them together, which gives an even smaller number. Besides, it is possible to get into difficulty with the precision of floating point values. Another problem is that the computation can be very slow, which should be avoided in real-time applications. To overcome these issues, we work in the log probability space, i.e., we take the logarithm of the probabilities.

Thus, the probability of making the observation S from a location l , ignoring the unobserved Access Points (i.e. $S_i : S_i = 0$) is given by:

$$\begin{aligned}
 P(S|l) &= \prod_{i=1, S_i \neq 0}^{|A|} \left(\frac{1}{\sqrt{2\pi}\mathbf{D}_i^l} \times \exp^{-\frac{(S_i - \mathbf{M}_i^l)^2}{2\mathbf{D}_i^{l^2}}} \right) \\
 &= \left(\frac{1}{\sqrt{2\pi}\mathbf{D}_1^l} \times \exp^{-\frac{(S_1 - \mathbf{M}_1^l)^2}{2\mathbf{D}_1^{l^2}}} \right) \times \dots \times \left(\frac{1}{\sqrt{2\pi}\mathbf{D}_{|A|}^l} \times \exp^{-\frac{(S_{|A|} - \mathbf{M}_{|A|}^l)^2}{2\mathbf{D}_{|A|}^{l^2}}} \right) \\
 \log(P(S|l)) &= \left(\log\left(\frac{1}{\sqrt{2\pi}\mathbf{D}_1^l}\right) - \frac{(S_1 - \mathbf{M}_1^l)^2}{2\mathbf{D}_1^{l^2}} \right) + \dots + \left(\log\left(\frac{1}{\sqrt{2\pi}\mathbf{D}_{|A|}^l}\right) - \frac{(S_{|A|} - \mathbf{M}_{|A|}^l)^2}{2\mathbf{D}_{|A|}^{l^2}} \right) \\
 &= \sum_{i=1, S_i \neq 0}^{|A|} \log\left(\frac{1}{\sqrt{2\pi}\mathbf{D}_i^l}\right) - \frac{(S_i - \mathbf{M}_i^l)^2}{2\mathbf{D}_i^{l^2}}
 \end{aligned} \tag{13}$$

In order to evaluate how distinct the points are on the WiFi map of the Laboratory, we used their mean RSS values as observations. The behaviour of the estimated probability given by our Perceptual Model $P = (S|l)$ (Equation 13) along the map is depicted by Figure 36.

The discrete point where the robot observed the signals is marked with a cross, and the other discrete points are marked with regular dots. Thus, we calculated the probability value for all discrete points $p_i \in \mathbf{P}$ based on the observation of the point marked with a cross, and the values for the rest of the map were estimated by Bicubic Interpolation, which is more accurate than other interpolation techniques but takes more time to process. The values are color-coded, varying from the minimum to the maximum values found.

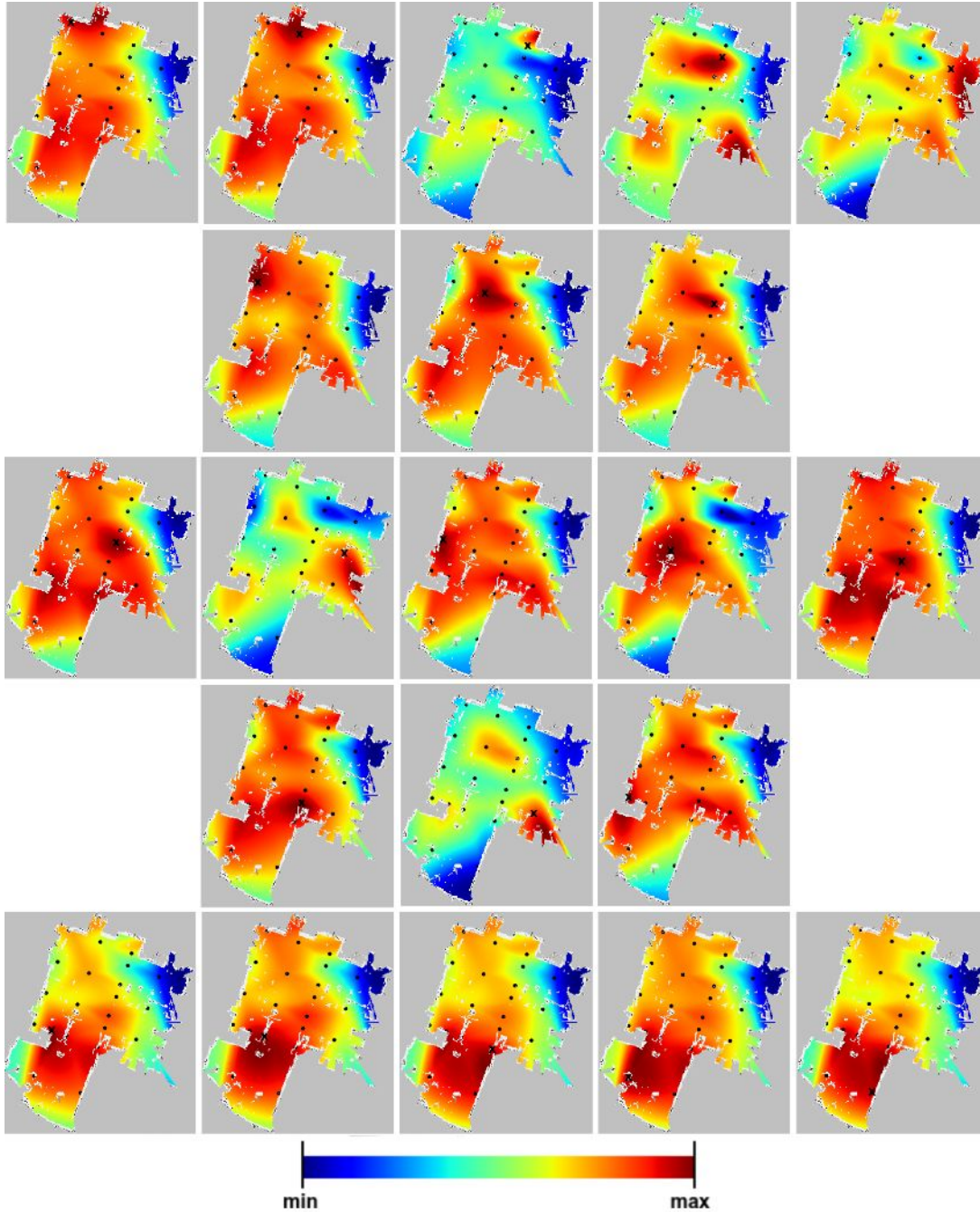


Figure 36 – WiFi Heat Map: Laboratory.

3.2.3.3 Resampling

With the estimate of the Perceptual Model $P(S|l)$ and the Motion Model $P(l_t|l_{t-1}, u_{t-1})$ of the Particle Filter, the localization *belief* of the robot can be recursively updated. But in the meantime, low weight particles must be replaced to ensure the convergence of the algorithm.

The *resampling* step consists of drawing new particles after the *update* step. As result, part of particles will be duplicated, and some rejected - the higher the weight, the greater the chance that the particle will be drawn several times.

Resampling can be compared with a probabilistic implementation of Darwin's theory, which relates to adaptation by natural selection. In this case, the "adjustment" is the particle weight. The same behavior can be seen in Genetic Algorithms (GAs) where the fitness function plays this role.

In our implementation of Particle Filter, we use a resampling algorithm (Algorithm 5) based on the Multinomial Resampling method (DOUC, 2005), also called Simple Random Resampling. But instead of resampling only N particles of the particle set \mathbb{P} , it is able to replace the entire set. In addition to this, particles with weight less than a threshold min_w are forced to be resampled without chance of being duplicated.

Using an uniform distribution based on the particle weights, the algorithm randomly chooses a particle to replace other. This process is similar to the Roulette Wheel on GAs.

Algorithm 5 resample(\mathbb{P})

```

1: for  $i \leftarrow 1..|\mathbb{P}|$  do
2:   if  $w_i < min_w$  then
3:      $w_i \leftarrow 0$ 
4:   end if
5:    $sum_w \leftarrow sum_w + w_i$ 
6: end for
7: for  $i \leftarrow 1..|\mathbb{P}|$  do
8:    $r \leftarrow \text{random.uniform}(sum_w)$ 
9:    $cum_w \leftarrow 0$ 
10:  for  $j \leftarrow 1..|\mathbb{P}|$  do
11:     $k \leftarrow j$ 
12:     $cum_w \leftarrow cum_w + w_j$ 
13:    if  $r \leq cum_w$  then
14:      break
15:    end if
16:  end for
17:   $p_i \leftarrow p_k$ 
18: end for

```

3.2.3.4 Infer Location

So far, the robot has only hypotheses about its location, represented by the particle set \mathbb{P} . To perform tasks like navigating to some point on the map, the robot needs to specify its current location as a single position (x, y, θ) . Therefore, to infer the current position of the robot, we sum the particle weights for each cell of the WiFi grid map. Then, using the cell with highest weight, we calculate its center of mass based on its particles (Equations 14-15).

$$x = \frac{\sum_{i=1}^{|\mathbb{P}_{bestCell}|} (x_i \times w_i)}{\sum_{i=1}^{|\mathbb{P}_{bestCell}|} w_i} \quad (14)$$

$$y = \frac{\sum_{i=1}^{|\mathbb{P}_{bestCell}|} (y_i \times w_i)}{\sum_{i=1}^{|\mathbb{P}_{bestCell}|} w_i} \quad (15)$$

The orientation θ is given by odometry data which uses Kinect, gyroscope, wheel encoders, and others devices to correct the robot's direction.

3.3 Kinect+WiFi Localization

Kinect Localization is the most popular technique in robotics today. It can achieve sub-centimeter accuracy using a very low cost device. However, in indoor human environments, objects can move or be moved, which changes the internal map of the robot. Also, there might be many similar parts on the map generating ambiguous observations for the localization algorithm.

For these reasons, many robotic platforms that use Kinect localization needs a human intervention to set the initial pose of the robot once it is turned on or when it is completely lost in the environment.

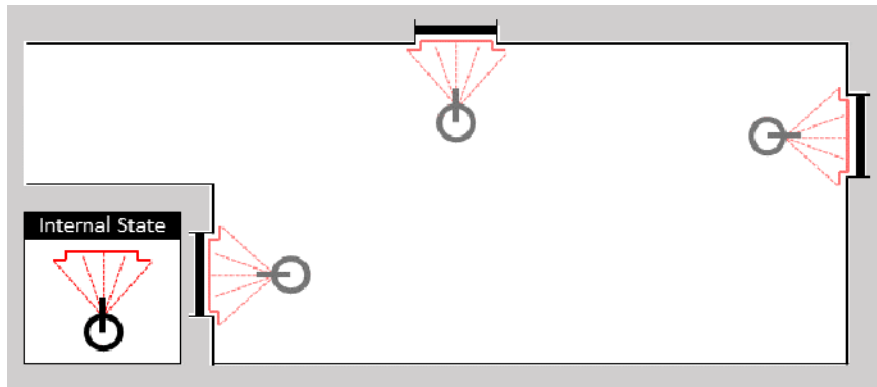


Figure 37 – Example - Kinect Localization.

Figure 37 shows an example of an indoor environment where a robot uses its Kinect to scan the area. In this example, the robot captures a very distinct observation, what

looks like to be the shape of a door one meter ahead. Despite the precision, there are three options in different parts of the map where this observation fits.

On the other hand, WiFi Localization is also a suitable approach for indoor human environments, as it is applicable anywhere there is dense WiFi network. It is not costly either, since many robotic platforms have onboard computers with WiFi data card or WiFi dongle capable of capturing WiFi signals.

But the minimum error of WiFi Localization can be greater than that of Kinect due to noise in the working Radio Frequency of Access Points. Therefore, the estimates of robot's position are never concentrated in a single spot of the map.

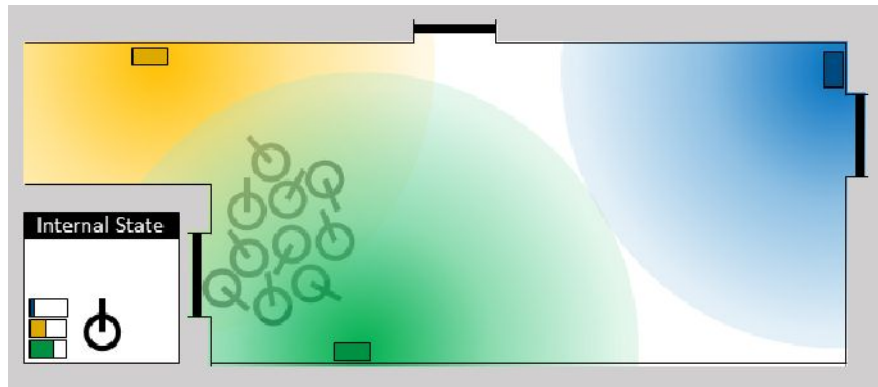


Figure 38 – Example - WiFi Localization.

Following our previous example, suppose that our robot is using WiFi instead of Kinect (Figure 38). The robot can listen to three Access Points (green, blue, and yellow) positioned at distant places along the environment. This time, the robot is not confused by observations from different parts of the environment. The main problem is that, the WiFi signals are not very distinguished locally. Furthermore, using WiFi only, it is not possible to determine the robot's orientation, which is extremely important to keep tracking the robot's pose during the localization process.

Our approach aims to combine the benefits of both localization techniques. The general idea is to use the WiFi approach for global localization on the map. Then, the Kinect approach estimates the current position (x, y, θ) of the robot considering the region determined by WiFi localization. Since Kinect hypotheses are bounded by those of WiFi, our algorithm uses thresholds to check whether the robot is lost, and recovers its position autonomously without human intervention using the global estimate given by the WiFi localization.

In the example depicted by Figure 39, the robot rejects hypotheses of Kinect localization that do not belong to the area delimited by WiFi localization, reducing the uncertainty about its position on the map. Besides, the remaining hypotheses help to estimate the correct position of the robot in that area.

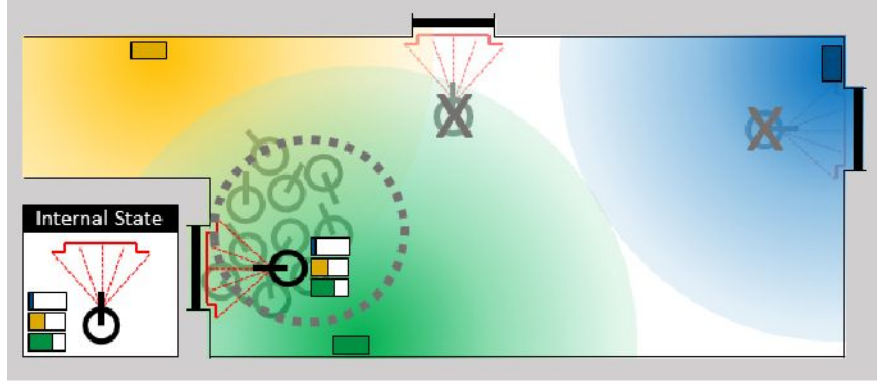


Figure 39 – Example - Kinect+WiFi Localization.



The proposed localization algorithm uses two Particle Filters (PFs), one for each device. The reason why we use two separate PFs, instead of merging their observations in a single PF, as it is done in (CANEDO-RODRÍGUEZ et al., 2016), is because WiFi particles cannot be close to each other in order to compensate for the WiFi noise. On the other hand, Kinect particles should be able to converge almost to the same spot as Kinect observations are not locally noisy.

Algorithm 6 WiFi-Kinect-Localization($\mathbb{M}_{wifi}, \mathbb{M}_{kinect}$)

```

1:  $PF_{wifi}.init(\mathbb{M}_{wifi})$ 
2:  $PF_{kinect}.init(\mathbb{M}_{kinect})$ 
3:  $location \leftarrow \emptyset$ 
4: while (true) do
5:   if ( $PF_{wifi}.convergence() > C$ ) then
6:      $PF_{wifi}.spreadParticles()$ 
7:     while ( $PF_{wifi}.convergence() > C$ ) do
8:       wait
9:     end while
10:     $PF_{kinect}.spreadParticles(PF_{wifi}.location)$ 
11:  else
12:    if ( $Distance(PF_{wifi}.location, PF_{kinect}.location) > D$ ) then
13:       $PF_{kinect}.spreadParticles(PF_{wifi}.location)$ 
14:    else
15:       $location \leftarrow PF_{kinect}.location$ 
16:    end if
17:  end if
18: end while

```

Algorithm 6 shows the pseudo-code of our algorithm. It starts by initializing both Particle Filters, PF_{wifi} and PF_{kinect} , with their respective maps \mathbb{M}_{wifi} and \mathbb{M}_{kinect} (lines 1 and 2). The current location of the robot is set as empty (line 3).

Then, the main loop runs indefinitely while the robot is on (lines 4 to 18). Firstly, the algorithm verifies if the convergence of the WiFi particles are greater than the threshold

C by computing the average distance between all particles of PF_{wifi} and its estimated location $PF_{wifi}.location$ (line 5). If so, the WiFi particles are randomly spread all over the place (line 6). Then, the algorithm waits for the convergence of WiFi particles (lines 7 to 9). This time, the Kinect particles are randomly spread from a normal distribution centered on the estimated location of $PF_{wifi}.location$ (line 10). Usually, this process (from line 5 to 10) happens when the robot is turned on or “kidnapped”, which enables the robot to recover its position autonomously.

In case the WiFi particles are well converged, the algorithm tests if the distance between the locations $PF_{wifi}.location$ and $PF_{kinect}.location$ is greater than the threshold D (line 12), i.e., if the estimated location given by the Kinect approach is out of the perimeter delimited by the WiFi global localization. If so, the algorithm also spreads the Kinect particles based on the estimated location of $PF_{wifi}.location$ (line 13). Otherwise, the current location of the robot is set with $PF_{kinect}.location$ (line 15).

3.4 Experiments

Localization experiments were carried out using a Turtlebot in our Laboratory at CMU. They aim to evaluate the localization approaches addressed in this chapter: Kinect, WiFi, and Kinect+WiFi; especially in indoor open areas which are more challenging than small narrow ones, like corridors, where the precision is better due to walls and corners that prevent noise on WiFi and give more featured observations to the Kinect.

In the first experiment, we compared the localization accuracy of the approaches during the navigation on a fixed path crossing the Laboratory. Seven checkpoints positioned along the path (Figure 40) were used to measure the error between the actual location and the estimated location given by each approach.

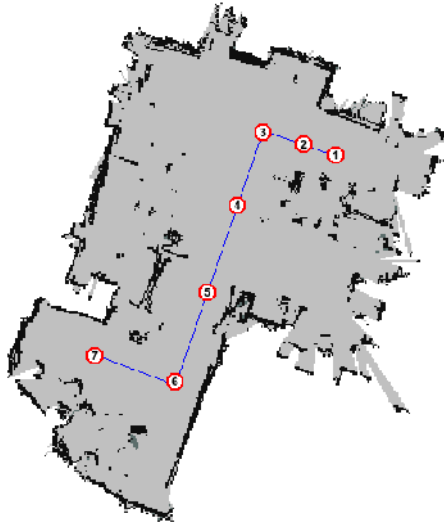


Figure 40 – Ground Truth Path.

Figure 41 shows a trial of our experiment retrieved from RViz. The left column of the figure illustrates the original path performed by the robot. The next column represents the execution with Kinect localization. The third column shows the execution using WiFi localization. Finally, the last column depicts the execution using the Kinect+WiFi approach. In this picture, black dots represent the locations of the robot given by each technique, green arrows are Kinect particles and red arrows are WiFi particles.

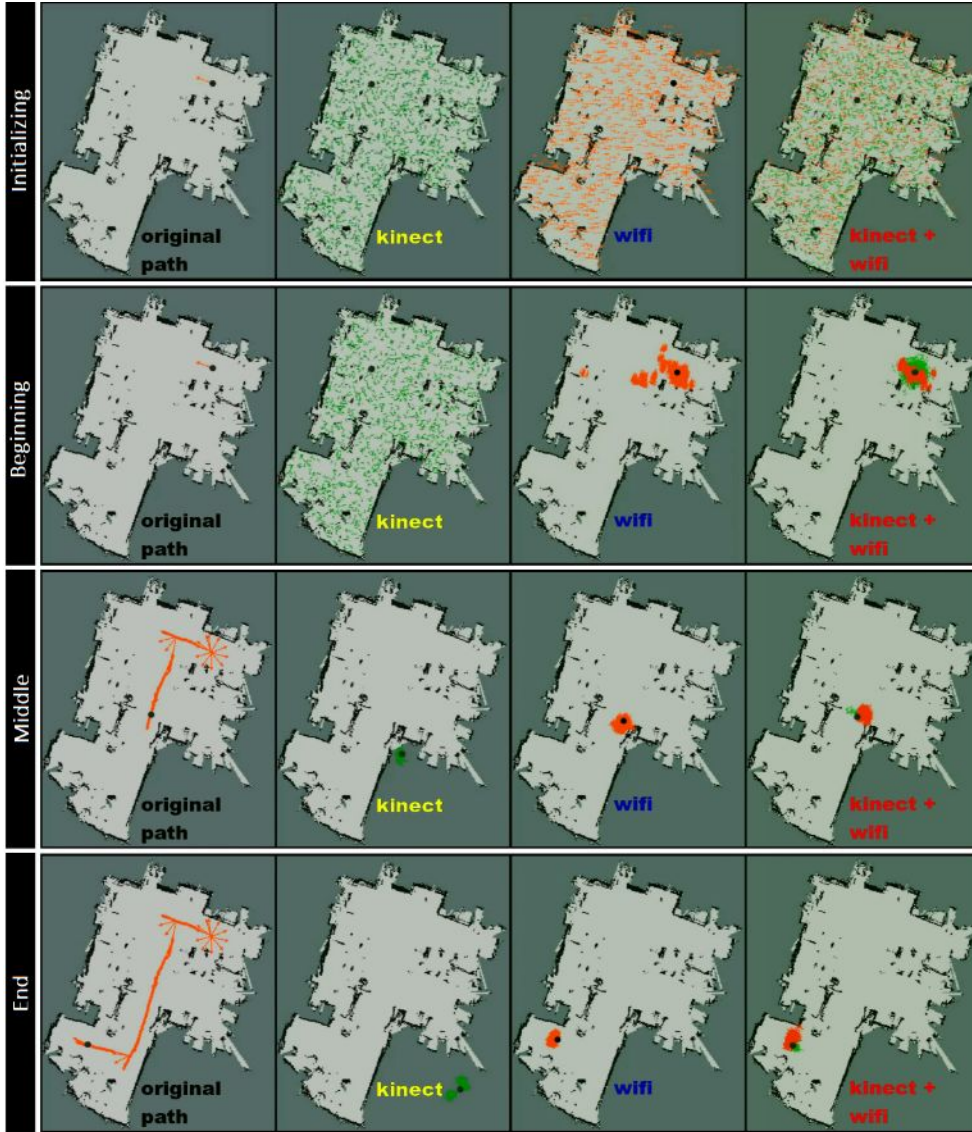


Figure 41 – Trial example on the path.



The first frame, on top, shows the initialization of the experiment where all particles are spread across the map. The second frame depicts the beginning of each approach and their first convergences. The reason why Kinect particles stay almost static in the same places is because the robot is not moving yet, and the Kinect is getting the same observation over and over again. On the other hand, the WiFi algorithm keeps receiving

different observations, which enables the updating of particle weights. Then, the robot makes a turn on its axis and starts the path. The next frame shows the robot in the middle of the path. There, the robot is completely lost in the Kinect approach. This happens when the robot is not well localized before starting the navigation. To overcome this issue, many Kinect approaches need human intervention to set the initial pose of the robot. At this point, the Kinect+WiFi localization is slightly better than the WiFi localization. The last frame, at the bottom, illustrates the end of the path, where the estimate of Kinect+WiFi localization is close to the actual location of the robot. Although the WiFi localization seems to yield the same result, its estimate is a little further from the actual location, and it could get worsen if the path were longer.

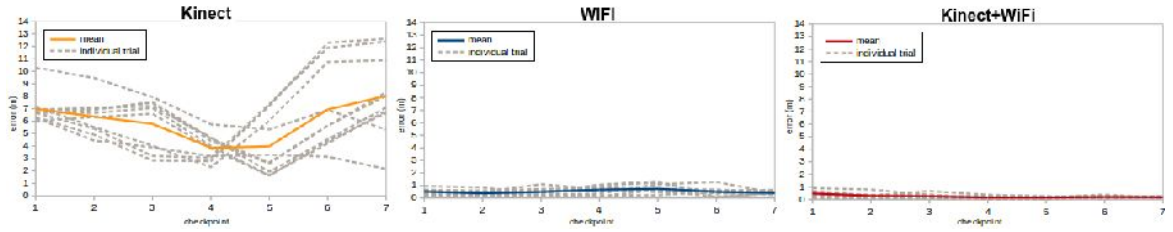


Figure 42 – Localization error along the path.

The robot performed the path 10 times for each approach. Figure 42 shows a plot of the error in the localization at each checkpoint of the path. The dashed lines represent the error for individual trials and the colored ones represent the mean error. Figure 43 shows only the mean error of all approaches together. As depicted by the red line in the chart, the average error for Kinect+WiFi localization was less than 50cm almost all the time.

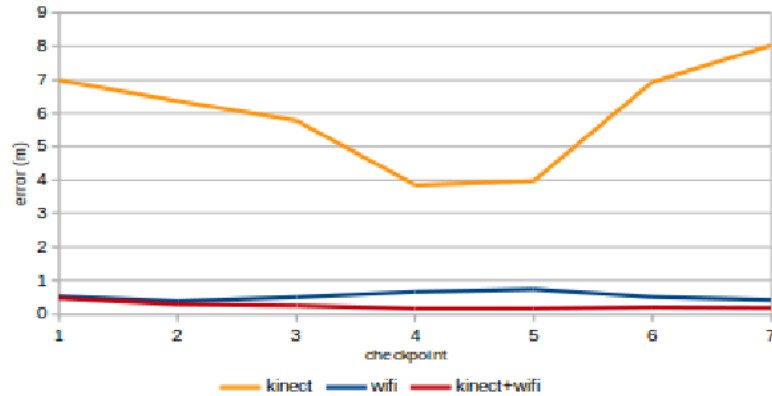


Figure 43 – Mean error of all approaches.

Furthermore, the Kinect+WiFi localization algorithm (Algorithm 6) enables the robot to automatically set its pose in case it is lost. In robotics, this problem is known as the

Kidnapped Robot Problem, which commonly refers to a situation where an autonomous robot in operation is carried to an arbitrary location.

The kidnapped Robot Problem creates significant issues with the robot's localization system, and only a subset of localization algorithms can successfully deal with the uncertainty created; it is commonly used to test a robot's ability to recover from catastrophic localization failures.

In the next experiment, we test this ability of our robot by performing a sequence of kidnappings. Figure 44 illustrates this process. First, the robot was positioned at some place in the Laboratory and its location was properly set. Then, the robot was taken to somewhere else. Despite receiving no new odometry data, the robot senses that its position has changed by perceiving different WiFi observations. In such case, the weights of its WiFi particles start decreasing, and when they reach a threshold, the algorithm randomly initializes the particles again across the map.

When the WiFi particles converge, the algorithm uses the estimated location given by the WiFi Particle Filter to reset the Kinect particles. Then, the Kinect Particle Filter uses Kinect observations to update particles at right places and recover the robot's position.



Figure 44 – Robot Kidnapping Experiment.



In order to evaluate the uncertainty on the localization and how long this process takes, we performed 50 kidnappings with our robot in the Laboratory. The chart of the Figure 45 shows the mean localization error for 5 seconds of recovering. In 2 seconds, the error is less than 1 meter. To keep reducing the error after the recovering, the robot needs to navigate a little so the Kinect particles can converge.

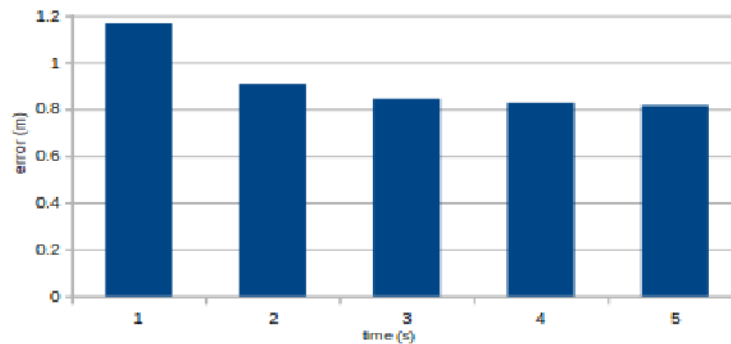


Figure 45 – Recovering Time.

While the robot is navigating, it is reasonable to expect that it would drop WiFi signal strength readings from some Access Points. Therefore, it is difficult to have a perfect observation containing values for all Access Points.

So, in this last experiment, we investigated the impact of dropped signals on the WiFi Localization accuracy. During navigation in another path, we limited the number of Access Points that could be used in the localization. Thus, every time an observation was required by the WiFi localization algorithm, we first randomly removed some Access Points from the observation vector.

The following figures illustrate a trial of the experiment. Using 2, 4, 6, 8, 10, 12, 14, 16, and 18 Access Points, the test starts by randomly initializing WiFi particles across the map. The red arrows represent WiFi particles and the green arrows represent the accurate estimate of the robot's position.

Figure 46 shows the behavior of the particles using different amounts of Access Points at the beginning of the path, a few seconds after the initialization. By looking at the picture, it is noteworthy that the convergence of the particles is better when more Access Points are used.

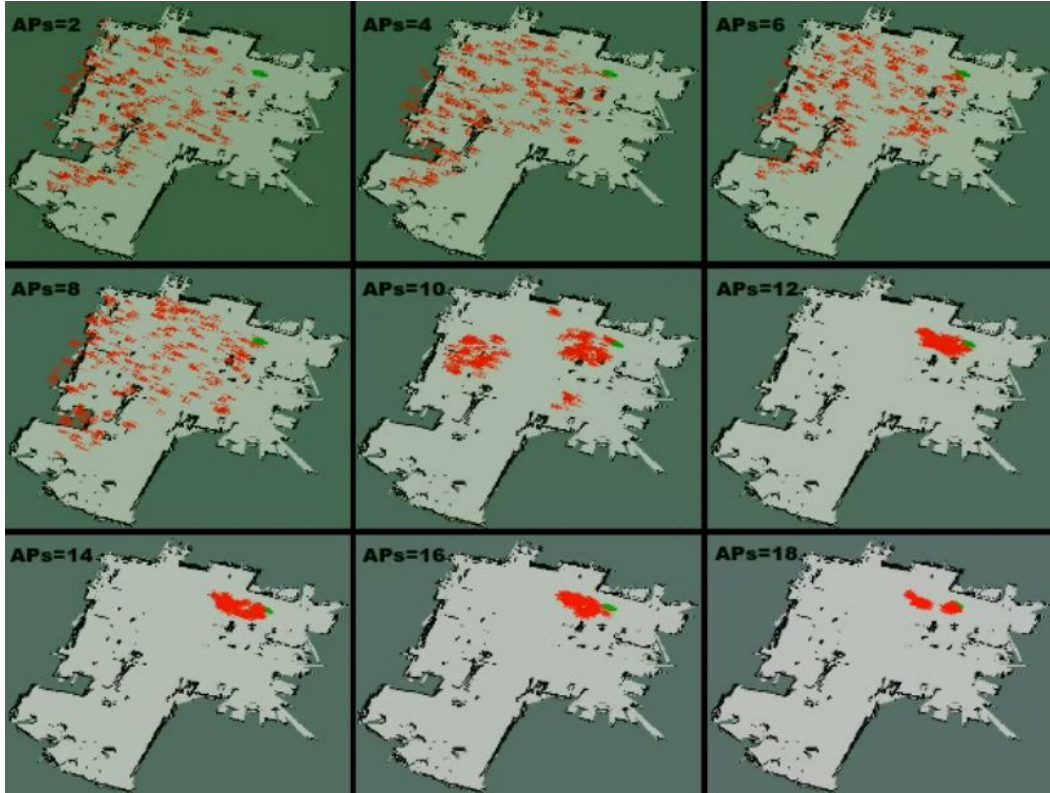


Figure 46 – Accuracy of WiFi localization: beginning of the path.

The next picture (Figure 47) shows the particles at the end of the path. For some instances, where only few Access Points were used, the WiFi particles are distant from the accurate estimate. However, using more than 10 Access Points, the WiFi particles,

depicted by red arrows, are well concentrated and covering the robot's position given by green arrows.

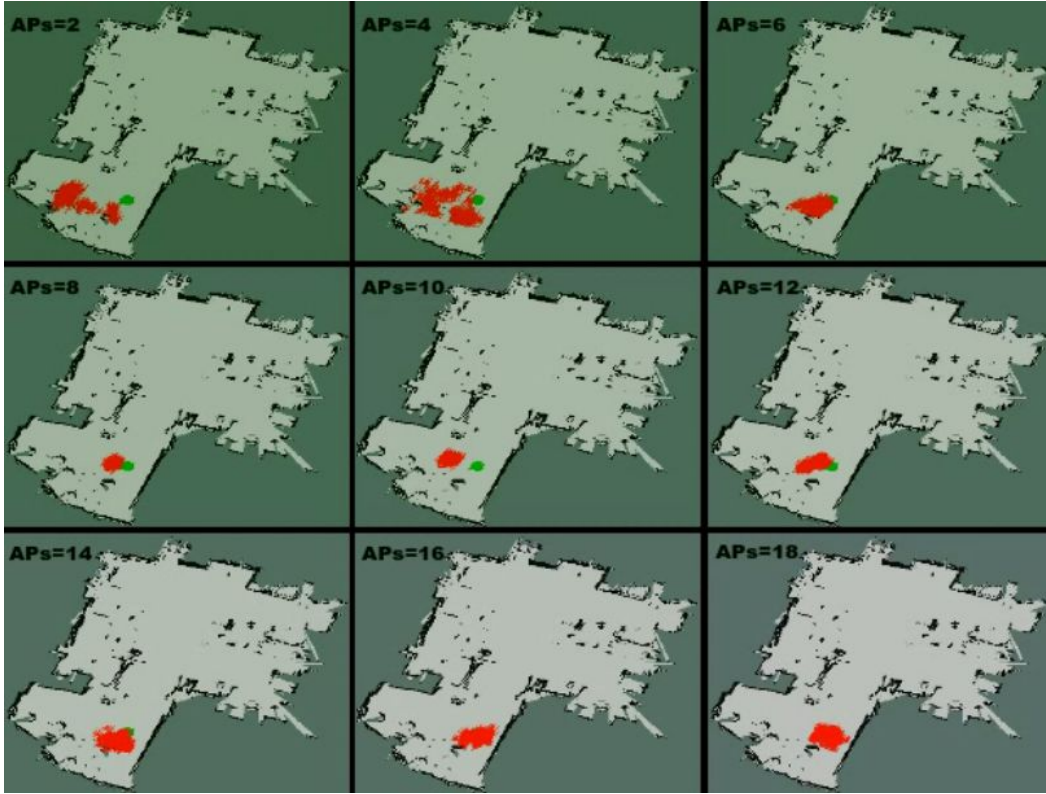


Figure 47 – Accuracy of WiFi localization: end of the path.



The chart of the Figure 48 presents the results of this test. The horizontal axis of the chart displays the number of Access Points used in the localization, and the vertical axis shows the average error along the path. Although our WiFi map is composed by 21 Access Points, it is not possible to listen to all of them everywhere all the time, specially in 1 second of observation. But even with 12 Access Points, the mean error in localization is close to 1 meter.

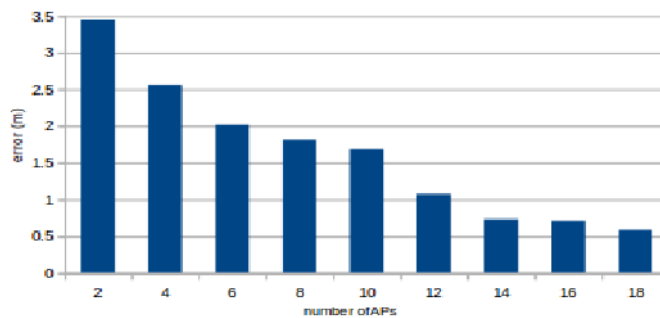


Figure 48 – Localization error vs. Number of Access Points.

3.5 Summary

In this chapter, we presented algorithms for robot localization and mapping. We focused on techniques that can be applied to indoor human environments. We started by showing an approach that uses a Kinect device to map and localize a real robot in an indoor area. To map the area, we used the GMapping algorithm to treat depth images captured from a Kinect and build an occupancy grid. Then, with this grid map, the AMCL algorithm estimates the robot's position in real time by comparing its observations against the map.

Then, we introduced a WiFi localization approach. We proposed the concept of irregular grids to collect WiFi signal strengths at discrete points in the area to generate a WiFi map. Then, our localization algorithm, based on the Particle Filter framework, estimates the robot's position continuously along the map by the means of different interpolation techniques.

Considering that the Kinect approach is more accurate locally and the WiFi approach gives a global location of the robot on the map, we further introduced a localization algorithm that combines both approaches. Our algorithm also enables the robot to autonomously recover its position in case it is lost.

Finally, we presented results on robot localization by running different tests. The experiments showed that the proposed algorithms are well appropriate for localization in indoor human environments, yielding acceptable errors on the estimates.

CHAPTER 4

Navigation

In robotics, a navigation system aims to plan a path to a target location and to execute this plan while the robot avoids unexpected obstacles. Navigation in indoor human environments is a challenging task due to moving and movable objects. In this case, moving objects are dynamic objects able to move fast and autonomously in the environment, such as people. Movable objects are not able to move by themselves, but they can be moved frequently, like furniture.

There are several approaches that treat obstacle avoidance along with path planning:

- *Bug algorithms*: these algorithms, in its original version Bug 1 (LUMELSKY; SKEWIS, 1990) or in the improved version Bug 2 (LUMELSKY; STEPANOV, 1990), are simple ways to overcome unexpected obstacles in the robot motion from a start point to a goal point. When they face an unknown obstacle, they are able to easily produce their own path contouring the object in the 2D surface if a path to the goal exists. The purpose is to generate a collision-free path by using the boundary-following and the motion-to-goal behaviors. In addition, the Bug's family has three assumptions about the mobile robot: i) the robot is a point, ii) it has a perfect localization, and iii) its sensors are precise.
- *Artificial Potential Fields*: the application of artificial potential fields to obstacle avoidance was first developed by (KHATIB, 1986). The concept is that artificial forces, generated by the obstacles and target, are applied to the robot in order to move about the environment collision free. The obstacles and goal position are assigned repulsive and attractive forces respectively. This motivates the robot to move towards the goal while being 'pushed' away from the obstacles. The main disadvantages of the potential field methodology are that the trap situations can occur due to the presence of local minima.
- *Vector Field Histogram*: even though the Artificial Potential Fields method performs quite fast it has its shortcomings. The implemented test-bed shows that often the robot would not move between obstacles too close to each other due to the repellent

effect from both sides, causing the robots to repel away. To solve this problem (BORENSTEIN; KOREN, 1991) developed the Vector Field Histogram technique. The method employs the use of 2D histogram grid to represent the environment, being reduced to single dimension “polar histogram” which is build around the position of the robot in a certain moment. The sectors presented in the polar histogram show the “polar obstacle density”. The direction of the robot is computed by choosing the sector with least concentration of obstacles. The map represented by the histogram grid is renewed from the robot’s sensors with data containing the distance between the robot and obstacles.

- *Elastic Band Concept*: the Elastic Band Concept (QUINLAN; KHATIB, 1993) works by deforming the original obstacles free path supplied by a path planner. The reason for that is that often the path planner computes a path that has sharp turns, which makes it impossible for the robot to steer. The path modified using the Elastic Band concept is shorter and smoother than the original path. This method can adapt to dynamic changes in the environment modifying the path if new obstacles are detected, avoiding the need for a new path preplanning. There are two forces that modify the form of the new path. A force that mimics the stretching of an elastic band eliminating the “slack” called “contraction force”, and an opposite force called “repulsion force” providing more room by repelling the robot away from obstacles.
- *Follow the Gap Method*: works by finding the widest gap among the obstacles and allows the robot to move through center of the obstacles. It also calculates the best heading vector through the gap and finally calculates the final angle (SEZER; GOKASAN, 2012) (ZOHAIB et al., 2014).

But in general, the path planning in robotics is made off-line based on a static known map. So it is reasonable to expect that movable objects have changed the map after the last navigation, especially in indoor environments. Thus, the planning must be able to adapt to changes while the robot is executing the path. On the other hand, moving objects are too dynamic to be considered in the path planning. What must be done is to avoid the robot to collide to these objects while it is navigating by means of reactive controllers.

Among the mobile robot research community reactive behavior and planning behavior are often accepted as opposite approaches. The mobile robot must be able to act accordingly when unforeseen obstacles are found on-the-fly. If the robot rely only on pure path planning the robot is prone to physical collision with an unforeseen obstacle. On the other hand without path planning, with the use of reactive obstacle avoidance method only it will be impossible for the robot to reach its goal location (KUNCHEV et al., 2006).

Considering this scenario, (FOX; BURGARD; THRUN, 1997) divides the navigation problem into “global” and “local”. The global techniques that involve path planning methods rely on availability of a topological map defining the robots workspace and obstacle location. The benefit from using path planning is that the entire path from start to goal can be planned, but this method is not suitable for fast collision avoidance due to its slowness caused by their complexity. Therefore, in our work, we also divide the navigation task into two problems, Obstacle Avoidance (local) and Path Planning (global).

Obstacle Avoidance is one of the most important aspects to ensure the safety of both humans and robots that coexist at the same place. This task is composed by two phases. First, the robot must detect obstacles in the environment with its sensors. Then, it must choose an appropriate movement to go through the environment without colliding. However, the noise produced during the sensors reading can lead the robot to take wrong decisions. Thus, we introduced an obstacle avoidance controller for mobile robots using a Hybrid Intelligent System (HIS) based on Fuzzy Logic (FL) and Artificial Neural Network (ANN). Basically, the FL treats data of infrared sensors and then feeds an ANN that decides which movement the robot must perform.

Path Planning problem, in robotics, consists of finding the lowest-cost path for a robot to start moving from its initial position toward its destination. Although most of path finder algorithms assume that the environment is completely known before the robot begins its traverse, there are a lot of researches focusing on the development of dynamic algorithms to fix the path in case the environment changes. In this scenario, when arc costs change during the traverse then the remainder of the path may need to be replanned. As the robot acquires additional information via its sensors it can revise its plan to reduce the total cost of the traverse. In this sense, the D*Lite algorithm plans optimal traverses in real-time by incrementally repairing paths to the robot’s state as new information is discovered. But, sometimes the new path can be much longer than the original. In this case, robots can ask for help. For example, they can ask a human to move away or move a chair. This can be seen as a symbiotic behavior since robots are performing tasks for humans, and humans are helping robots to complete their tasks. Therefore, we introduce an adaptation to D*Lite which allows robots to ask for human assistance to move objects on its surrounding when its path is blocked. The algorithm also computes whether and how long the robot can wait for help.

The remainder of the Chapter is organized as follows. Section 4.1 presents the development of our Hybrid Intelligent System for obstacle avoidance including all experiments. The proposed path planning algorithm with human assistance is shown in Section 4.2. Finally, a summary is presented in Section 4.3.

4.1 Obstacle Avoidance

A real-time obstacle avoidance controller permits the detection of unknown obstacles simultaneously with the steering of the mobile robot to avoid collisions. In the past few years, many soft computing techniques are proposed by the researchers to solve the robot obstacle avoidance problem in the various environments (PANDEY, 2017):

- *Fuzzy Logic (FL)*: provides a formal technique for representing and implementing the human expert's heuristic knowledge and perception base actions. In fuzzy logic controller behavior based navigation the problem is decomposed into simpler tasks (independent behaviors) and each behavior is composed of a set of fuzzy logic rule statements intended at achieving a well-defined set of objectives. The design and optimization of an ordinal structure model of fuzzy logic controller for application of mobile robot obstacle avoidance have been presented (SAMSUDIN; AHMAD; MASHOHOR, 2011) (PARHI, 2005) (WANG; LIU, 2008) (PRADHAN; PARHI; PANDA, 2009).
- *Artificial Neural Network (ANN)*: is one of the important technique for the mobile robot navigation. This neural network technique is motivated from the human brain. There are many works in robotics applying ANN for wall-following (NICHOLS; MCDAID; SIDDIQUE, 2013), on-line path planning between unknown obstacles (MOTLAGH et al., 2014), steering angle controller in static and dynamic environments (GAVRILOV; LEE, 2007), etc.
- *Genetic Algorithm (GA)*: the implementation of Genetic Algorithm to the mobile robot obstacle avoidance problem needs the development of an appropriate chromosome for the robot path, a path direction mechanism, a methodology to cater for obstacle avoidance and an appropriate constraint definition providing mechanisms to minimize path length as well as providing collision free paths (TAMILSELVI et al., 2012) (CASTILLO; TRUJILLO; MELIN, 2007) (YUN; GANAPATHY; CHONG, 2010).
- *Simulated Annealing (SA)*: the simulated annealing is an iterative search algorithm inspired by the annealing of metals. In this case, the simulated annealing algorithm is used to search a collision free optimal for robots (YANAR; AKYÜREK, 2011) (ZHU; YAN; XING, 2006) (PRECUP et al., 2013).
- *Particle Swarm Optimization (PSO)*: the main objective of the PSO algorithms is to determine a solution of an optimization problem in search configuration field. Path planning for each robot has to be determined in order to avoid collisions between the robots while they are in motion. Several undesirable situations like congestions and deadlocks may obstruct the progress path for the robots. In such condition use of

particle swarm optimization algorithm can be beneficial for efficient path planning and avoiding such undesirable situations in the real world environment (LI; CHEN, 2005) (TANG; EBERHARD, 2011) (ZHANG; GONG; ZHANG, 2013) (HUANG, 2014).

- *Ant Colony Optimization (ACO)*: the objective of the Ant Colony Optimization algorithms is to search an optimal path in a field, to solve many combinatorial optimization problems. In case of any obstacles present in their way, ants move along the contour of the obstacle on either sides and find their path to the food source. As the concentration of pheromone being more along the shorter path, so ants accumulate more pheromone in a given time interval along the shorter path (CHIA et al., 2010) (GANGANATH; CHENG; TSE, 2014) (BI; YIMIN; YISAN, 2009).

Usually, an autonomous robot uses some of these techniques to treat the input signal from its sensors in order to detect the presence of obstacles, and then to compute a motion that moves the robot away, avoiding the collision. But the noise produced in the environment, during the sensor reading, can affect the algorithms that control the robots navigation.

Therefore, in this section, we explain the development of a Hybrid Intelligent System (HIS) for obstacle avoidance that combines two techniques, FL to treat the uncertainty of sensors and ANN to select the best motion for the robot.

4.1.1 Related Work

There are several works in the literature regarding obstacle avoidance for robots with Fuzzy Logic and Artificial Neural Networks.

For instance, Chi and Lee (CHI; LEE, 2011) proposed an ANN control system that is able to guide a P3DX robot to traverse a maze with arbitrary obstacles. The model was trained by using *Matlab*. The inputs are generated by 8 sonar sensors, while the outputs indicate the angle of the robot's movement.

Another technique widely used in such case is FL. In (LI; CHOI, 2013) is presented a control system based on FL for mobile robots that aims to treat the uncertainties of an ultrasonic sensor and calculate the angular velocity of the motors.

However, in the last years, many researches focused on the development of Hybrid Intelligent Systems (HISs). These systems combine two or more different techniques, but at least one of them is from Artificial Intelligence (AI). This approach is reasonable because a particular technique might not be sufficient to solve the whole problem, but it could be used to solve at least part of it. Therefore, it is possible to combine the advantages of various techniques for solving a given problem.

Thus, some works related to robots navigation have been applying a fusion of FL, to handle the uncertainties of the environment, along with other techniques such as ANN to decide the robot's movement.

Considering this approach, Jeffril and Sariff (JEFFRIL; SARIFF, 2013) introduced a HIS based on FL and ANN. The FL algorithm treats the data of 8 infrared sensors and feeds an ANN which moves the robot in a certain direction. This model was trained on *Matlab* and tested in a virtual *E-Puck* robot. The problem is that the membership functions and the fuzzy rules must be created by a human specialist.

Moreover, a system is called neuro-fuzzy when it merges FL with ANN in order to create a homogeneous model. In this case, the components of FL are distributed within the ANN architecture, making it an indivisible system. But if it is necessary to adjust the model for a different environment, the entire system must be retrained because the FL elements cannot be separated from the ANN and vice versa.

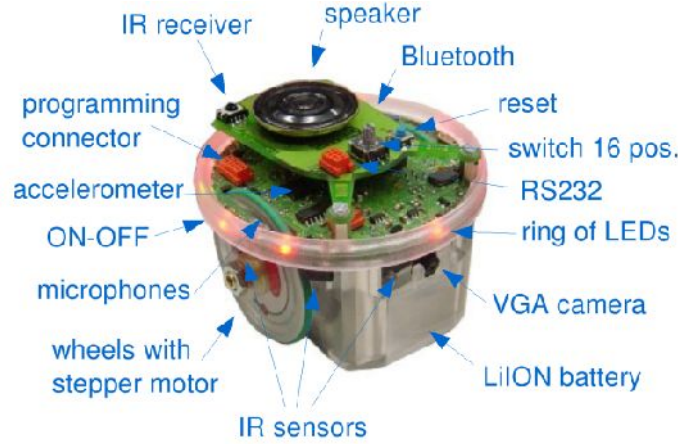
In (ALGABRI; MATHKOUR; RAMDANE, 2014) and (DUTTA, 2010) were addressed the application of *Adaptive Neuro-Fuzzy Inference System* (ANFIS) to control mobile robots. Algabri et. al (ALGABRI; MATHKOUR; RAMDANE, 2014) developed a multilayer ANFIS where 2 layers are dedicated to control the movements and the other 2 to detect obstacles. Also, Dutta (DUTTA, 2010) used a PSO algorithm to evolve the membership functions of the ANFIS network.

4.1.2 HIS controller

In our approach, we propose a HIS architecture that uses two separated modules, a fuzzy control based on the Fuzzy Lattice Reasoning (FLR) algorithm (KABURLASOS; ATHANASIADIS; MITKAS, 2007) along with a Multiple Layer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1988) network, to ensure a collision-free navigation for mobile robots. In this system, the sensors data pass through the fuzzy rules to generate linguistic/symbolic information about possible obstacles in the environment, for example, if the path is free or if there is an obstacle on the left side of the robot. These symbolic information are used by the neural network to determine which movement the robot must do to navigate without colliding, such as moving forward or turning right.

This system was compared to another approach that uses only a MLP. In such case, each sensor is an input neuron and the output neurons represent the robot movements. Both approaches were tested on an *E-Puck* robot within a 3D simulation environment using *Webots* (MICHEL, 2004) platform. Also, samples were created on *Webots* and used to generate the fuzzy rules and train the neural network on *Weka* (UNIVERSITY OF WAIKATO,).

The *E-Puck* is a small robot composed by: 2 wheels with stepper motors, 9 leds, 1 speaker, 1 VGA camera, 8 infrared sensors, 1 3D accelerometer, 3 microphones, and other devices, as depicted by the Figure 49.

Figure 49 – *E-Puck* devices

So far, we have used a virtual model of the *E-Puck* available on *Webots* platform. *Webots* is a software used to model, program, simulate and then transfer the code to a real robot. Samples were generated on *Webots* to create the fuzzy rules and also to create and train the neural networks. Then, both approaches were implemented and tested again on *Webots*.

Another software used in this work was the *Waikato Environment for Knowledge Analysis (Weka)*. It provides tools for preprocessing data, clustering, classification, regression, visualization and data association rules. In order to automatically create the fuzzy rules and the neural networks from the samples generated on *Webots*, two *Weka* packages were used, the *FLR* and the *MultilayerPerceptron*.

The *E-Puck* has 8 infrared sensors that measure the distance between the robot and a given object. Their values range from 0 to 2000, i.e., a value equals to 0 means that the object is distant and 2000 means that it is near. The stepper motors that control the speed of the wheels can receive values from -1000 to 1000. The speed is higher when the value is close to these limits. In addition, negative values make the wheel go backward, positive values make it go forward, and when the value is equal to 0 there is no movement on the wheel.

Reactive models, like static rules made by a specialist, could be applied to control the robots. However, some unpredictable aspects, such as the light variation or the interference from other devices, can lead the robot to have a wrong evaluation about the environment causing inappropriate movements. Moreover, the robot doesn't demonstrate autonomy over its actions.

The proposed HIS aims to guarantee a collision-free navigation to any kind of obstacle for an *E-Puck* robot. Thus, the robot have to sense if there is an object blocking its path and make a safe deviation. This system is composed by an architecture with two independent modules: Fuzzy and MLP (Figure 50).

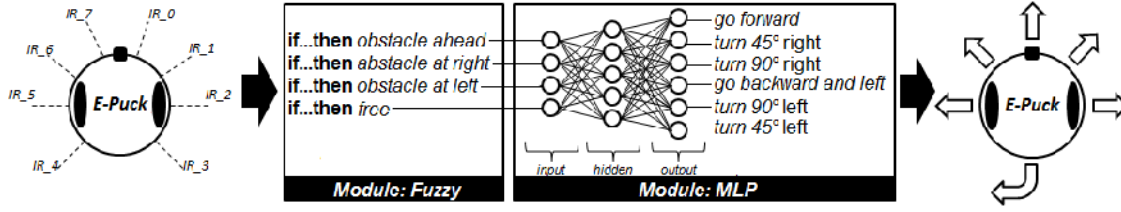


Figure 50 – HIS controller

The Fuzzy Module is responsible for treating the data of infrared (IR) sensors. The fuzzy rules of this module use the sensors values to determine if there is an obstacle around the robot. The outputs are symbolic variables that show if the robot is free or if there is an obstacle ahead, at left and/or at right. These variables represent the input neurons of the MLP. The goal of the MLP Module is to select the best movement for the robot considering the given input values. The possible output movements are: go forward, turn 45° right, turn 90° right, go backward and turn left, turn 90° left and turn 45° left.

4.1.2.1 Fuzzy Module

Fuzzy set theory differs from conventional set theory as it allows each element of a given set to belong to that set to some degree. Thus, the main idea behind fuzzy systems is that truth values or membership values are indicated by a value in the range 0-1 with 0 for absolute falsity and 1 for absolute truth (KLIR; YUAN, 1995).

Usually, a fuzzy system consists of the following steps:

- ❑ Fuzzification: crisp inputs are fuzzified to be associated to linguistic variables using membership functions.
- ❑ Inference: after the fuzzification, the inference engine refers to the fuzzy rule base containing fuzzy IF-THEN rules to derive the linguistic values for the intermediate and output linguistic variables.
- ❑ Defuzzification: once the output linguistic values are available, the defuzzification produces the final crisp values from the output linguistic values.

The main problem of this process is that it is necessary the knowledge of a specialist to define the fuzzy rules and the regions of the membership functions.

The *Fuzzy Lattice Reasoning* (FLR) (KABURLASOS; ATHANASIADIS; MITKAS, 2007) is a classifier for inducing decision-making knowledge (rules) based on hyperboxes. A hyperbox may be assigned a class label thus corresponding to the following rule: If a point p is inside hyperbox h (labeled by c) then p is in class c . The advantages of hyperbox-based rule induction include fast computation as well as straightforward interpretation.

Therefore, FLR is a technique that automatically generates fuzzy rules from samples. These samples are composed by numeric attributes and can be separated in classes labeled by linguistic/symbolic variables.

The FLR was applied in this work for two reasons. First, the MLP Module have symbolic inputs, thus, it is not necessary to execute the entire fuzzy process, specially the defuzzification phase. Second, it can generate fuzzy rules without human intervention.

In order to generate fuzzy rules to describe whether there is an obstacle around the robot, it is necessary to create samples on the environment in which the robot will navigate. These samples were created on *Webots* with an *E-puck* robot and a spherical obstacle.

The Figure 51 illustrates the sample generation to detect obstacles on the right side of the robot. The samples are structured in a vector composed by the values of each sensor, from IR0 to IR7, and their classes, in this case *class 1* (obstacle at right). On the right side of the figure, an obstacle is placed near to the robot. On the left side of the figure, the value of each infrared sensor is displayed on the screen. The IR1 and IR2 have higher values, which indicate an obstacle at right. The robot stays in a loop, spinning a little on its own axis, which creates more diversified samples. The samples are printed in the console, as shown at the bottom of the figure.

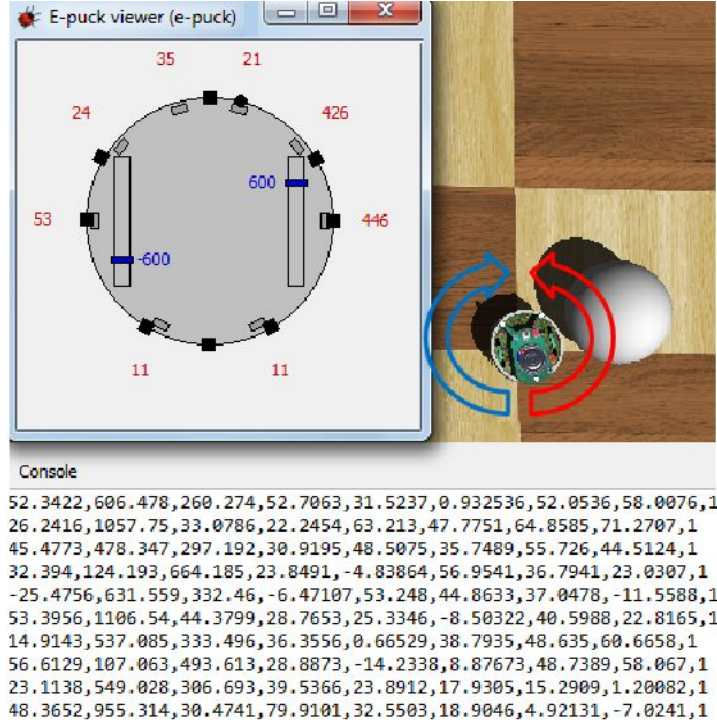


Figure 51 – Example of sample generation for the Fuzzy Module

To generate even more diversified samples for each class, the obstacle was placed as follows:

- ❑ class 0 (obstacle ahead): the obstacle was placed near to IR7, then near to IR0 and finally between IR7 and IR0.
- ❑ class 1 (obstacle at right): the obstacle was placed near to IR1, then near to IR2 and finally between IR1 and IR2.
- ❑ class 2 (obstacle at left): the obstacle was placed near to IR5, then near to IR6 and finally between IR5 and IR6.
- ❑ class 3 (free): no obstacle was placed near to the robot.

More than 7000 samples were created. These samples were converted to a *.arff* file and imported on *Weka*. The training method used was the cross-validation with 10 folds. The FLR obtained an accuracy equal to 76,37%. After the FLR execution, the following rules were generated:

- ❑ **IF** $(-18 < IR0 < 3107) \wedge (-64 < IR1 < 1921) \wedge (-74 < IR2 < 125) \wedge (-46 < IR3 < 115) \wedge (-49 < IR4 < 107) \wedge (-60 < IR5 < 108) \wedge (-40 < IR6 < 1250) \wedge (-33 < IR7 < 3009)$ **THEN** class 0
- ❑ **IF** $(-50 < IR0 < 647) \wedge (-36 < IR1 < 3005) \wedge (-25 < IR2 < 2465) \wedge (-38 < IR3 < 107) \wedge (-53 < IR4 < 121) \wedge (-43 < IR5 < 103) \wedge (-69 < IR6 < 106) \wedge (-34 < IR7 < 110)$ **THEN** class 1
- ❑ **IF** $(-40 < IR0 < 115) \wedge (-46 < IR1 < 129) \wedge (-39 < IR2 < 111) \wedge (-48 < IR3 < 108) \wedge (-44 < IR4 < 130) \wedge (-41 < IR5 < 2009) \wedge (-14 < IR6 < 2935) \wedge (-62 < IR7 < 2206)$ **THEN** class 2
- ❑ **IF** $(-43 < IR0 < 105) \wedge (-44 < IR1 < 111) \wedge (-38 < IR2 < 111) \wedge (-37 < IR3 < 114) \wedge (-51 < IR4 < 103) \wedge (-63 < IR5 < 114) \wedge (-45 < IR6 < 113) \wedge (-53 < IR7 < 118)$ **THEN** class 3

4.1.2.2 MLP Module

Artificial Neural Networks (ANNs) belong to a field of AI known as connectionist. They are inspired by the physical structure of biological neurons and nervous system. An ANN is composed by processing units (artificial neurons) distributed in layers and linked by weighted connections. The learning process is made by the adjustment of these connection weights which aims to reduce the classification error.

ANNs have a great power of generalization because they are able to classify samples not submitted to them during the training phase.

Multiple Layer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1988) maybe is the most popular architecture of ANN. It is characterized by the presence of at least one hidden layer of neurons. Hidden layers are those located between the input layer and the respective output layer. Therefore, a MLP network must have at least two layers of processing neurons, which allows to solve non-linear problems.

The MLP implemented in this module have to choose an appropriate movement using the symbolic inputs generated by the Fuzzy Module. In order to create and train this

neural network on *Weka*, it is necessary to generate other samples on *Webots*. This time, the samples must refer to classes that represent the movements, which are:

- ☐ class 0: go forward
- ☐ class 1: turn 45° right
- ☐ class 2: turn 90° right
- ☐ class 3: go backward and turn left
- ☐ class 4: turn 90° left
- ☐ class 5: turn 45° left

The samples are created using the rules of the Fuzzy Module. They are structured in a vector as follows:

- ☐ position 1 - obstacle ahead: 0 for false and 1 for true
- ☐ position 2 - obstacle at right: 0 for false and 1 for true
- ☐ position 3 - obstacle at left: 0 for false and 1 for true
- ☐ position 4 - obstacle free: 0 for false and 1 for true
- ☐ position 5 - movement class: from 0 to 5

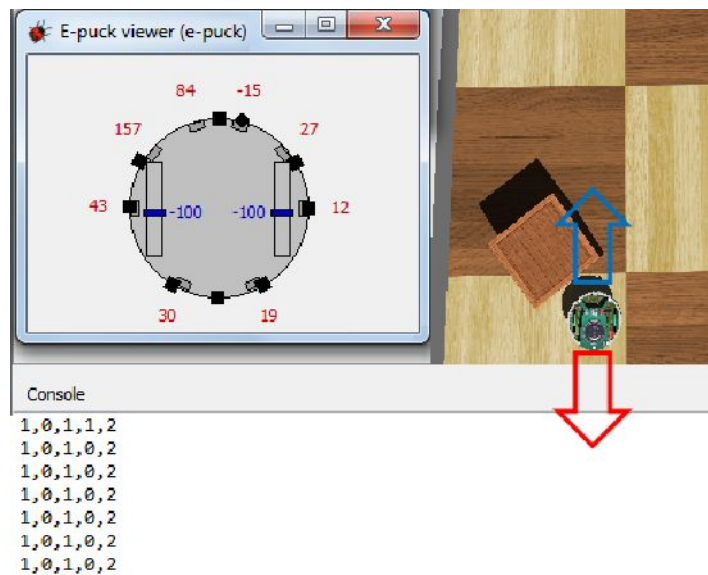


Figure 52 – Example of sample generation for the MLP Module

More than one fuzzy rule could be triggered, which means that an obstacle could be blocking the robot at left and ahead at the same time. In such case, it is required a movement like “turn 90° right”, described by the sample [1, 0, 1, 0, 2].

Moreover, the rule that says the robot is free can be triggered together with other rules. This means that, even if there is a possible obstacle around, the robot is sure to not deviate.

Different obstacles were used to create the samples for the MLP. Figure 52 depicts the sample generation to “turn 90° right”.

On the right side of the figure, an obstacle is blocking the front and the left side of the robot. On the left side of the figure, the value of each infrared sensor is displayed on the screen. The robot stays in a loop going forward and backward, which creates more diversified samples. The samples are printed in the console, as shown at the bottom of the figure.

More than 3500 samples were created. These samples were converted to a *.arff* file and imported on *Weka* to automatically create the MLP. This MLP has 4 neurons on the input layer, 5 neurons on the hidden layer, and 6 neurons that represent the movement classes on the output layer. The training parameters were: number of epochs = 500, learning rate = 0.3 and momentum = 0.2. Once again, the training method used was the cross-validation with 10 folds. The MLP obtained an accuracy equal to 89,48%.

4.1.3 MLP controller

Another obstacle avoidance controller, that uses only a MLP network, was developed and compared to the HIS controller. In this approach, the MLP must select one of the possible movements like the HIS controller. However, the values of the infrared sensors are sent directly to the neural network without any treatment, as illustrated by the Figure 53.

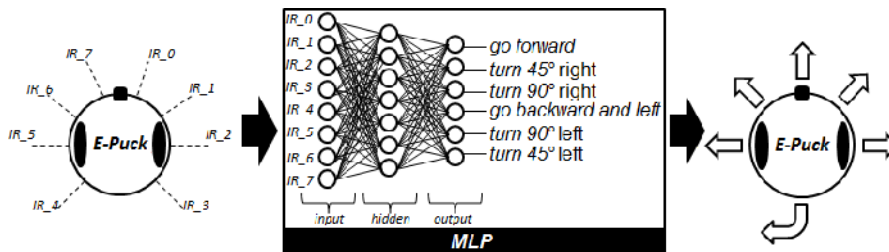


Figure 53 – MLP controller

Samples were generated in the same way of the MLP Module (Section 4.1.2.2), but they were structured in a vector composed by the values of each sensor, from IR0 to IR7, and their movement classes.

More than 7500 samples were created. These samples were converted to a *.arff* file and imported on *Weka* to automatically create the MLP. This MLP has 8 neurons that represent the infrared sensors on the input layer, 7 neurons on the hidden layer, and 6 neurons that represent the movement classes on the output layer. The training parameters were: number of epochs = 500, learning rate = 0.3 and momentum = 0.2. The training method used was also the cross-validation with 10 folds. This MLP obtained only 47,74% of accuracy.

4.1.4 Experiments

In order to make a comparative analysis between HIS controller and MLP controller, both approaches were set into a virtual *E-Puck* robot using *Webots*. Basically, the robot stays in a loop sending the sensors data to the controller and executing the selected movement.

The experiments were carried out by running the controllers in 4 different environments composed by 5, 10, 15 and 20 obstacles, as illustrated by the Figure 54. The robot must deviate from all obstacles and walls of the environments. For each approach, 10 executions of 5 minutes were realized in each environment.

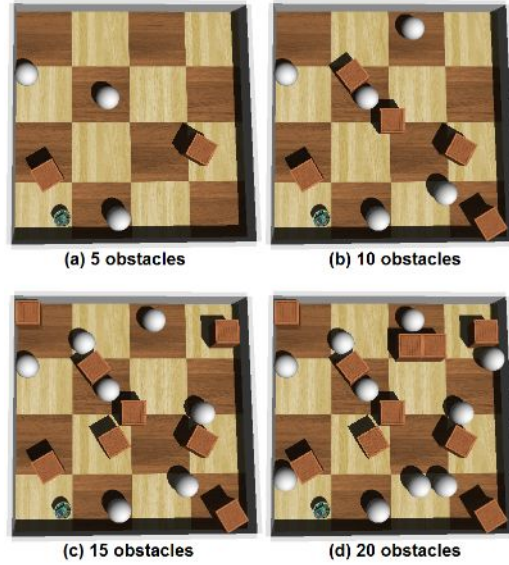


Figure 54 – Test environments.



Figure 55 depicts the results of both approaches. The horizontal axis of the chart displays the number of obstacles, and the vertical axis shows the average collision number.

Considering all the environments, the MLP controller had 60% more collisions than the HIS controller. This could be explained by the low accuracy obtained by the MLP network of this controller during the training phase. The high variation of the sensors values generates very different samples for the same class, which makes the classification

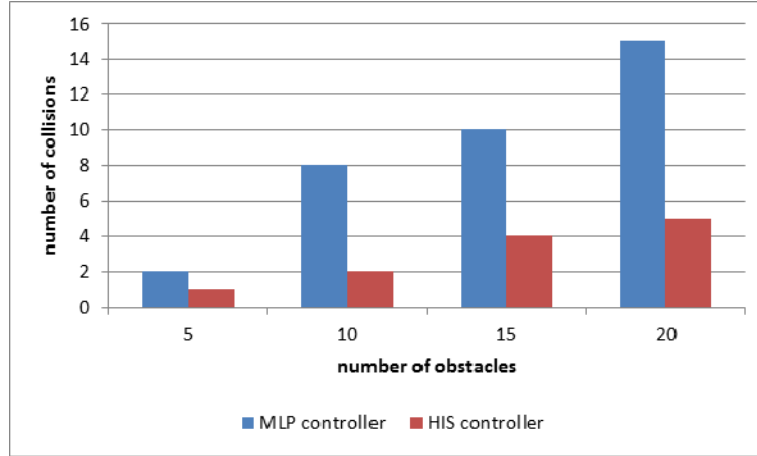


Figure 55 – Average collision number

too difficult. On the other hand, the FL Module of HIS controller works like a filter which helps the classification made by the MLP Module.

Therefore, the HIS controller avoids more collisions. Also, it produces smoother sequences of movements during the navigation, for example, the robot never stops going forward if the path is free. But the classification error of the MLP controller makes the robot turn a little sometimes, even if there is no obstacle around.

4.2 Path Planning

Path Planning for mobile robots consists of finding a sequence of state transitions that leads a robot from its initial state to some desired goal state. Typically, the states are robot locations and the transitions represent actions the robot can take, each of which has an associated cost. A path is said to be optimal if the sum of its transition costs (arc costs) is minimal across all possible paths leading from the initial position (start state) to the goal position (goal state). Such paths can be efficiently generated from a map of the environment using search algorithms such as A^* (HART; RAPHAEL, 1968) and *Dijkstra* (DIJKSTRA, 1959).

But when operating in real environments, a mobile robot usually does not have complete map information. As a result, any path generated using its initial map may turn out to be invalid or suboptimal as it receives updated map information through, for example, an onboard sensor. It is thus important that the robot is able to update its map and replan optimal paths when new information arrives. To overcome this issue, a new path can be generated from scratch regarding the current location of the robot and the changes found in the map. Another solution is to replan or fix the path more efficiently to go through the environment avoiding the objects detected.

D^* (STENTZ, 1994) is a dynamic version of A^* which prioritizes computational ef-

efficiency rather than optimality in changing environments. It uses heuristics to focus its search and reuses information from previous searches to find solutions to series of similar searches much faster than is possible by solving each search from scratch. D^* and its variants, like its simpler version called D^*Lite (KOENIG; LIKHACHEV, 2002), have been widely used for mobile robot and autonomous vehicle navigation including the Mars rovers (WOODEN, 2006).

Depending on the changes in the environment, the replanned path may be much longer than the original or in the worst case the target location cannot be reached anymore. In this sense, the concept of “Navigation Among Movable Obstacles” (NAMO) (STILMAN; KUFFNER, 2005) proposes that a robot must be able to move objects in its surroundings to reach a previously obstructed location. But this is a very complex task and involves several areas of robotics, like motion, vision, and grasping. Moreover, many popular robotic platforms today still lack arms and hands. Therefore, we introduce an adaptation to the D^*Lite algorithm which enables the robot to ask for human assistance to move objects on the way based on the replanned path. This can be seen as a symbiotic behavior since robots are performing tasks for humans, and humans are helping robots to complete their tasks.

The main idea is to compare the original path against the replanned one and determine whether the robot needs to ask for human assistance to move objects and how long it can wait for help. While the robot is waiting, it keeps scanning the environment to check if there are changes that allow the replanning of shorter paths.

Although this work is related to the interaction between robots and humans, the goal is to reduce the navigation time by removing obstacles and replanning paths efficiently with search algorithms rather than proposing new Human-Robot Interaction techniques.

4.2.1 Related Work

A number of A^* -like algorithms exist for performing path replanning. *Lifelong Planning A^** (LPA^*) (KOENIG; LIKHACHEV, 2001) is incremental version of A^* . It also uses heuristics to focus the search and always finds a shortest path for the current edge costs. The first search of LPA^* is the same as that of A^* but all subsequent searches are much faster. LPA^* produces at least the search tree that A^* builds. However, it achieves a substantial speedup over A^* because it reuses those parts of the previous search tree that are identical to the new search tree.

The D^* algorithm, a dynamic version of A^* , has been shown to be one to two orders of magnitude more efficient than planning from scratch with A^* , and it has been incorporated into a lot of real robotic systems (STENTZ; HEBERT, 1995), (HEBERT; MACLACHLAN; CHANG, 1999), (MATTHIES et al., 2002).

D^*Lite (KOENIG; LIKHACHEV, 2002) is a hybrid algorithm, taking aspects from both LPA^* and D^* . D^*Lite has been found to be slightly more efficient by some measures,

and it has been used to guide *Segbots* and *ATR*V vehicles in urban terrain (FERGUSON; LIKHACHEV; STENTZ, 2005). Both D^* and D^*Lite algorithms guarantee optimal paths over grid-based representations of a robot’s environment. Like LPA^* , D^*Lite replans when it finds divergent values at some arc costs during the navigation, which are classified as overconsistents or underconsistents. There are some differences between D^*Lite and LPA^* . First, D^*Lite switches the search direction of LPA^* , i.e., the direction of all edges within the environment are reversed. Another change that D^*Lite makes to LPA^* is to dynamically move the agent while updating the keys of vertices in its priority queue.

*Delayed D^** (FERGUSON; STENTZ, 2005) is a modified version of D^*Lite that delays the propagation of cost increases as long as possible. By delaying the processing of underconsistent states, *Delayed D^** holds two advantages over D^*Lite . Firstly, it is able to ignore some underconsistent states entirely, reducing the overall computation. Secondly, even when it has to process underconsistent states, if it has delayed processing them long enough then perhaps the effects of various underconsistent states can be incorporated at the same time, in a single propagation.

*Memory-Bounded D^*Lite (MD^*Lite)* (COCKBURN; KOBTI, 2008) is a variant of D^*Lite that operates in dynamic environments where agents have restricted memory. It uses less memory while maintaining optimality and completeness. MD^*Lite removes nodes from the priority queue after their expansion. When the algorithm runs out of space and needs to account for additional nodes, it recursively computes parent nodes, adding suitable ones to the queue until all nodes are again accounted for.

*DD *Lite* (MILLS-TETTEY; STENTZ; DIAS, 2006) extends D^*Lite to support reasoning about state dominance in a domain independent manner. The basic idea underlying search with state dominance is to identify and prune dominated states before they are expanded. It maintains the algorithmic simplicity and incremental search capability of D^*Lite , whilst enabling orders of magnitude improvements in search efficiency in large state spaces with dominance. In addition, *DD *Lite* is sound, complete, optimal, and efficient.

When planning on a 2D grid, it is common to plan from the center of each grid cell and only allow transitions to the centers of adjacent grid cells. This restricts the agent’s heading to increments of $\frac{\pi}{4}$, which results in paths that are suboptimal in length and difficult to traverse in practice. *Field D^** (FERGUSON; STENTZ, 2007) and E^* (PHILIPPSEN, 2006) apply interpolation to efficiently produce globally-smooth paths.

Our work is not directly related to search improvements of D^* . Instead, we aim to provide an adaptation that allows robots to ask for human assistance to easily move objects away and find better paths.

This symbiotic behavior can be seen in service robots such as *CoBots* (VELOSO et al., 2015b). These robots work in multiple floors of buildings occupied by the School of Computer Science at Carnegie Mellon University (CMU). The *CoBots* can perform

different types of tasks, but they have no hands. To overcome this actuation limitation, the robots proactively ask for help from humans, such as asking a human to push the elevator's buttons.

4.2.2 D*Lite with Human Assistance

Despite several advances in robotics, robots still have limitations at the perception, cognition, and execution levels. For example, the robot we used in our experiments lacks arms so it could never move blocking objects in its way. However, some of the robots limitations are strengths for humans who coexist with robots in the environment. In this sense, humans can clean robots' paths by lifting and moving objects, while robots are navigating in the environment in order to execute tasks for humans.

Therefore, we introduce an extension to the *D*Lite* algorithm that enables robots to ask for human assistance in case replanned paths, computed to deviate from obstacles, are too long. The algorithm is based on *D*Lite*, using the same notation. The changes required to *D*Lite with Human Assistance* (Algorithm 7) are presented below. Procedures that are the same as presented in *D*Lite* are simply mentioned.

The `Main()` procedure starts the execution of the algorithm. The variable s_{last} keeps the last initial state used to plan a path to the goal state s_{goal} . Initially, it is set with s_{start} which is the current initial state of the problem (line 27). Then, `Main()` calls `Initialize()` procedure to initialize the search problem (line 28).

After that, `Main()` executes `ComputeShortestPath()`, which contains the main loop of the algorithm (line 29). This procedure retrieves the shortest path as it is done in *D*Lite*.

Then, the robot starts executing the path until it reaches s_{goal} (lines 30-56). At each iteration of the execution, the robot selects the best successor of the current state (line 32) and moves to that state (line 33). Meanwhile, changes in the environment might occur. Therefore, onboard sensors scan the environment to detect modifications (line 34). If it is detected any change on edge costs of the map (line 35), the algorithm updates the parameter k_m (line 36), and sets again s_{last} as been the current state s_{start} (line 37). The parameter k_m is used to calculate states keys and prioritize the *OPEN* list U .

For all edges with changed costs, the procedure `UpdateVertex()` is called to update states potentially affected by the changes as well as their membership in the priority queue U (lines 38-40). The variable p_{size} receives the size of the current path (line 41), calculated by the procedure `CalculatePathSize()` (lines 21-26) which computes the path from the current state to the goal state and returns its size.

Once again, the algorithm calls `ComputeShortestPath()` to replan the path given the changes detected (line 42). The variable p'_{size} receives the size of the path after replanning (line 43). The algorithm proceeds by comparing the size of both paths, original and replanned (line 44).

Algorithm 7 D*Lite with Human Assistance

```

procedure CalculateKey( $s$ )
1: return  $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))];$ 

procedure Initialize()
2:  $U = \emptyset;$ 
3:  $k_m = 0;$ 
4: for all  $s \in S$   $rhs(s) = g(s) = \infty;$ 
5:  $rhs(s_{goal}) = 0;$ 
6:  $U.insert(s_{goal}, CalculateKey(s_{goal}));$ 

procedure UpdateVertex( $u$ )
7: if ( $u \neq s_{goal}$ )  $rhs(u) = \min_{s' \in Succ(u)} (c(u, s') + g(s'));$ 
8: if ( $u \in U$ )  $U.remove(u);$ 
9: if ( $g(u) \neq rhs(u)$ )  $U.insert(u, CalculateKey(u));$ 

procedure ComputeShortestPath()
10: while ( $U.TopKey() < CalculateKey(s_{start})$  OR  $rhs(s_{start}) \neq g(s_{start})$ )
11:    $k_{old} = U.TopKey();$ 
12:    $u = U.Pop();$ 
13:   if ( $k_{old} < CalculateKey(u)$ )
14:      $U.insert(u, CalculateKey(u));$ 
15:   else if ( $g(u) > rhs(u)$ )
16:      $g(u) = rhs(u);$ 
17:     for all  $s \in Pred(u)$  UpdateVertex( $s$ );
18:   else
19:      $g(u) = \infty;$ 
20:     for all  $s \in Pred(u) \cup \{u\}$  UpdateVertex( $s$ );

procedure CalculatePathSize()
21:  $P = \emptyset;$ 
22:  $s = s_{start};$ 
23: while ( $s \neq s_{goal}$ )
24:    $P.insert(s);$ 
25:    $s = \min_{s' \in Succ(s)} (c(s, s') + g(s'));$ 
26: return  $|P| * cell\_size$  /* in centimeters */;

procedure Main()
27:  $s_{last} = s_{start};$ 
28: Initialize();
29: ComputeShortestPath();
30: while ( $s_{start} \neq s_{goal}$ )
31:   /* if ( $g(s_{start}) = \infty$ ) then there is no known path */
32:    $s_{start} = \arg \min_{s' \in Succ(s_{start})} (c(s_{start}, s') + g(s'));$ 
33:   Move to  $s_{start};$ 
34:   Scan graph for changed edge costs;
35:   if any edge costs changed
36:      $k_m = k_m + h(s_{last}, s_{start});$ 
37:      $s_{last} = s_{start};$ 
38:     for all directed edges  $(u, v)$  with changed edge costs
39:       Update the edge cost  $c(u, v);$ 
40:       UpdateVertex( $u$ );
41:      $p_{size} = CalculatePathSize();$ 
42:     ComputeShortestPath();
43:      $p'_{size} = CalculatePathSize();$ 
44:     if ( $|p'_{size} - p_{size}| > \min\_deviation$ )
45:       Ask for Help;
46:        $timeout = |p'_{size} - p_{size}| * average\_speed;$ 
47:       while ( $timeout > 0$ )
48:         Scan graph for changed edge costs;
49:         if any edge costs changed
50:            $k_m = k_m + h(s_{last}, s_{start});$ 
51:            $s_{last} = s_{start};$ 
52:           for all directed edges  $(u, v)$  with changed edge costs
53:             Update the edge cost  $c(u, v);$ 
54:             UpdateVertex( $u$ );
55:             ComputeShortestPath();
56:             break;

```

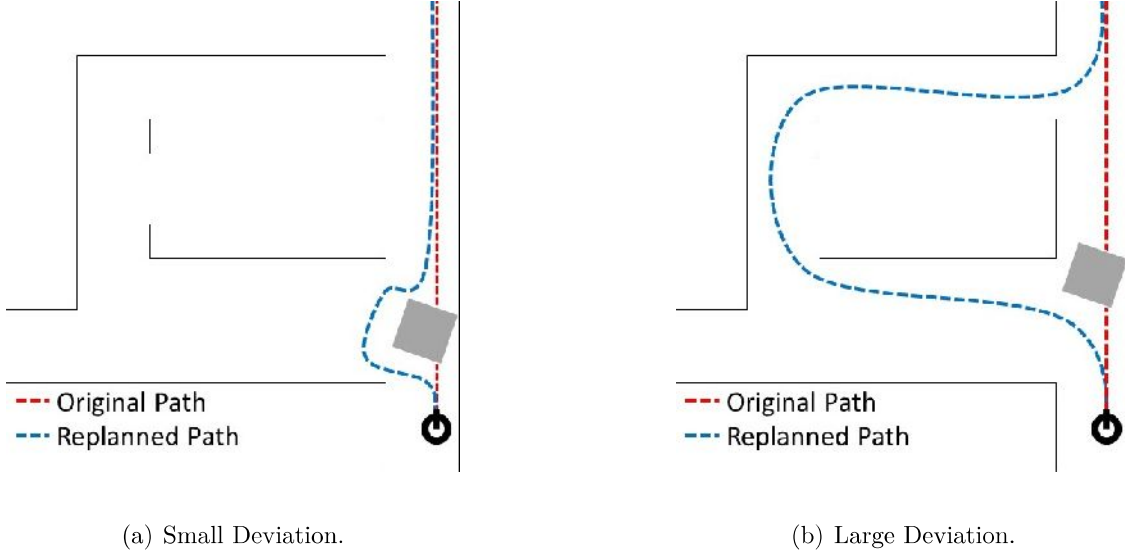


Figure 56 – Replanning Deviations in Obstructed Paths.

If the difference is less than the predefined threshold *min_deviation*, the robot executes the replanned path as the deviation can be done quickly. This scenario is depicted by Figure 56 (a). There, the robot is positioned at the bottom right of the image. The original path, dashed in red, is blocked by a gray box. The replanned path, dashed in blue, shows the deviation necessary to avoid the object. As this is a small deviation, the robot decides to perform the replanned path rather than waiting for human assistance.

Otherwise the robot sends messages, by playing an audio file and also printing a text on its console, to ask for assistance in case the replanned path is much longer the original (line 45), as shown in Figure 56 (b). The gray box is blocking the corridor where the robot goes, which requires a deviation to the other side. However, it cannot wait indefinitely because maybe there is nobody around to help. Thus, we compute a timeout based on the difference between both paths and the average speed of the robot (line 46). During this time, if anyone comes to clean at least part of the original path, the robot can reach the destination faster than if it takes the replanned path.

While it waits (lines 47-56), the robot keeps scanning the environment. If any change is detected (line 49) the robot replans again, which means that someone probably has cleaned the way. Thus, the algorithm proceeds as before (lines 50-55) and breaks the timeout loop (line 56).

4.2.3 Experiments

The navigation experiments were carried out using virtual and real environments. These experiments aim to ensure that we kept the properties of *D*Lite* for *D*Lite with Human Assistance*, and to evaluate the impact of human assistance during the navigation in indoor environments.

The first experiment aims to analyze the efficiency of *D*Lite with Human Assistance* for replanning when the original path is blocked. Therefore, we developed a system with a Graphical User Interface (GUI) as depicted by Figure 57. In this system, the user can generate a grid with sizes of 10x10, 20x20, or 30x30. There, Long Term Features (LTFs) are static blocked cells, colored in black, where the robot cannot traverse, like walls in a real indoor environment. The system randomly creates these cells depending on the parameter %LTF.

The user can set the start (green cell) and goal (red cell) states. At any time, the user can stop the system and add Short Term Features (STFs) and Dynamic Features (DFs). STFs are represented by purple cells which also block the path of the robot, but they are not part of the static map as LTFs. They can be seen as furniture or any other movable object. DFs are orange cells which dynamically change their positions over time, like people and other moving objects. After selecting one of the available algorithms, the system computes a path and starts the execution.

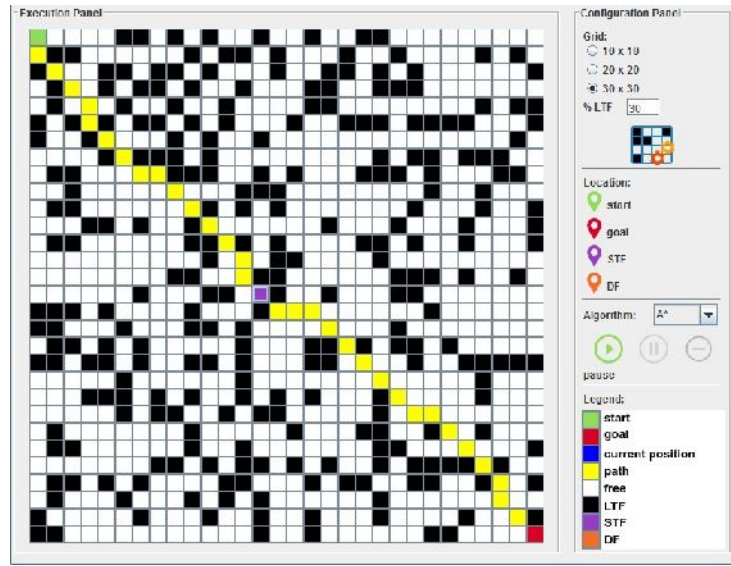


Figure 57 – Graphical User Interface (GUI)



We generated some scenarios using grids of 10x10, 20x20, and 30x30 cells. Two search algorithms were tested, *D*Lite with Human Assistance* and A^* . The tests were carried out by computing paths that cross the grid. After planning, the execution of the path begins. Then, some objects were added on the middle of the path to force the replanning. *D*Lite with Human Assistance* tries to replan the path efficiently while A^* plans another path from scratch.

Usually, search algorithms have an queue, called *OPEN* list, that maintains states to be explored during the search. Thus, to analyse the performance of both algorithms, we tracked how many states were stored into the *OPEN* lists while the algorithms were

replanning.

Figure 58 depicts the results of both approaches. The horizontal axis of the charts displays the number of states in the *OPEN* list and the vertical axis shows the iteration number during the search. As can be seen, the number of states explored using *D*Lite with Human Assistance* is much lower than the number of states used by *A**. Also, it requires fewer iterations to perform the search.

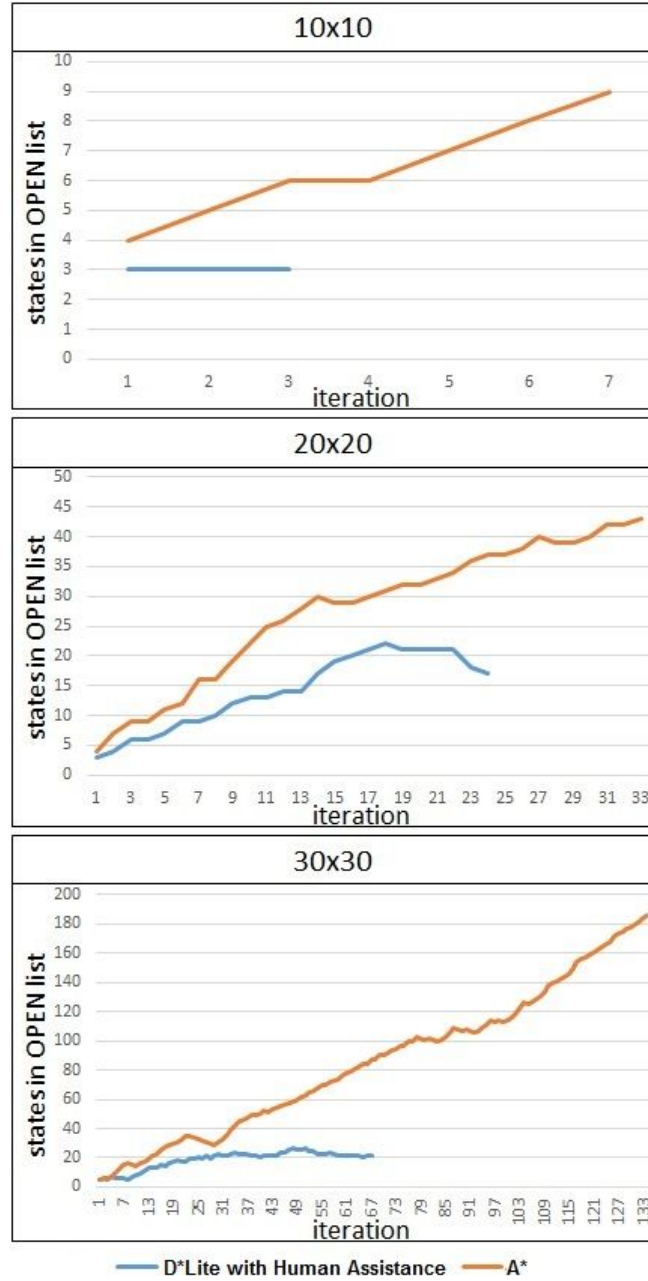


Figure 58 – Replanning: OPEN list

*A** algorithm needs to explore as many states as necessary to find the optimal path, which requires more CPU and memory resources to evaluate the states stored in the

OPEN list. This can be a problem for big and complex environments. Therefore, in the next experiment, we evaluate the performance of a relaxed version of A^* againsts D^*Lite with *Human Assistance*. This *Relaxed A^** does not track all possible paths to the goal state but only the best, working almost as a *Best First Search*.

We implemented both algorithms on *Robot Operating System* (ROS) as been Global Path Planners. ROS is an open-source, meta-operating system for robots. It provides services expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple robots and computers. With ROS, we can run our algorithms on real and virtual robots in the same way.

For this test, we used a virtual *Turtlebot* on *Gazebo*. *Gazebo* is a 3D simulator that has an interface with ROS. This simulator allows the creation of 3D scenarios with robots, obstacles and many other objects. It also uses a physical engine for illumination, gravity, inertia, etc. To track the internal state of the robot we used ROS visualization (RViz) which is a 3D visualizer for displaying sensor data and state information from the robot. In this experiment we evaluated the quality of the paths produced by D^*Lite with *Human Assistance* and *Relaxed A^** . We generated four different paths on the indoor virtual environment depicted by Figure 59.

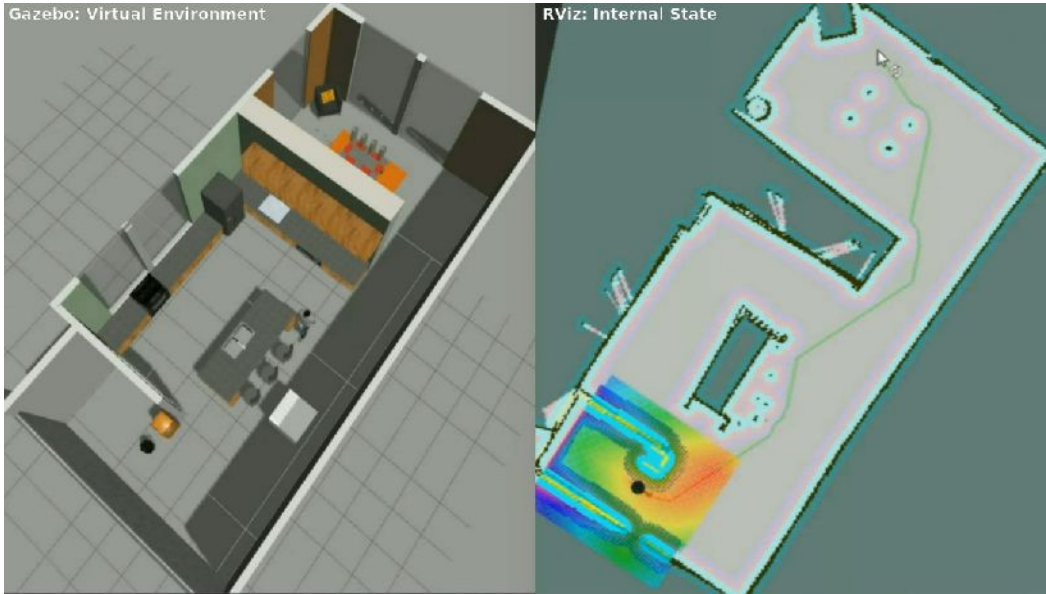


Figure 59 – Simulation with Turtlebot, ROS, Gazebo, and RViz



The charts in Figure 60 show that our approach yields shorter paths for longer distances or at least paths of the same length for smaller distances.

These two experiments show that we kept the properties of D^*Lite regarding the efficiency of building paths in dynamic environments. For the last experiment, we in-

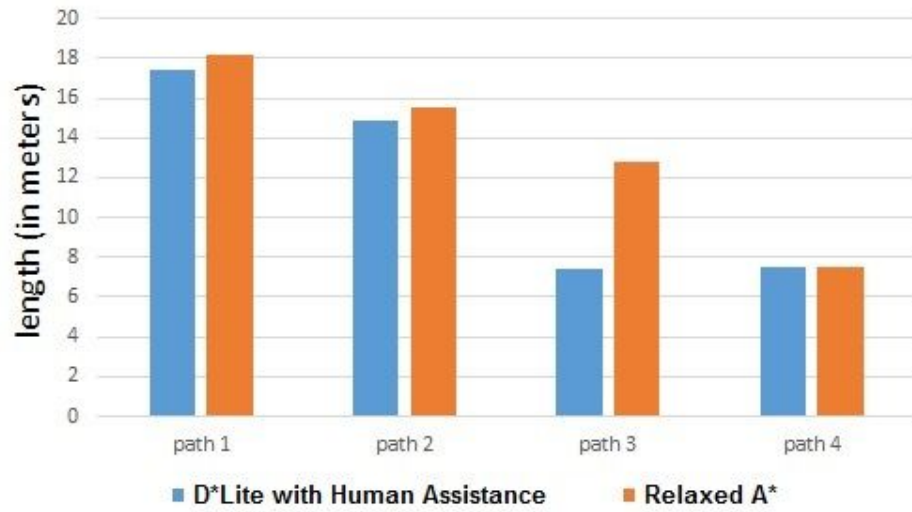


Figure 60 – Path length

investigated the impact of human assistance during the navigation in simple and complex environments using virtual and real scenarios, like those illustrated by Figures 61 62.

For the virtual scenarios we used a virtual *Turtlebot* on *Gazebo*, and for the real ones we used our physical *Turtlebot* with the same ROS codes that we developed.

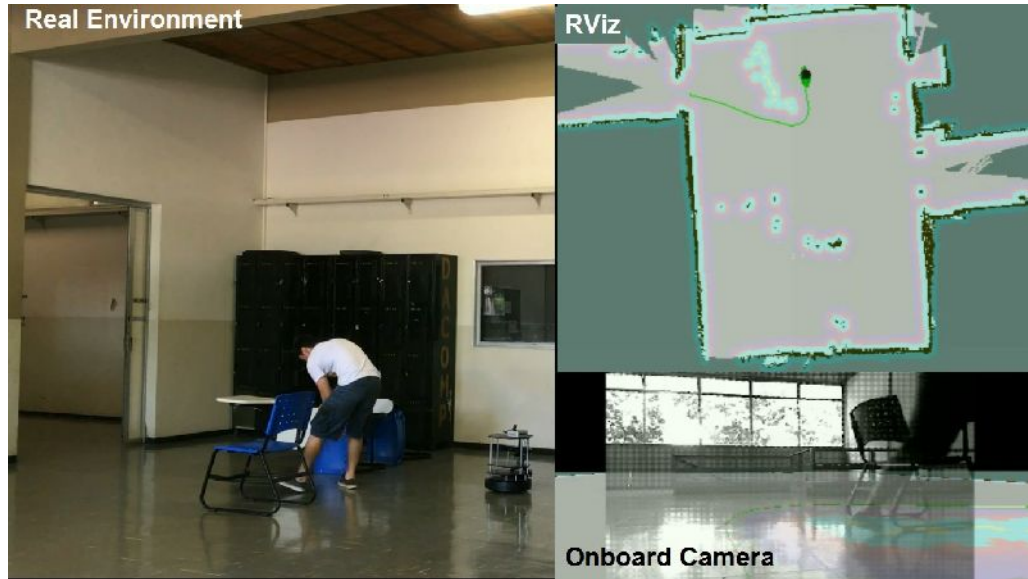


Figure 61 – Corridor: 2nd floor of Building-B at UFU



In this experiment we also added some obstacles further away in the middle of the the robot's path. When the robot detects theses objects it replans its path. Then we calculated the difference between the original and the replanned path as been the deviation necessary to the robot reaches the destination.

Figure 62 – Willow Garage: virtual environment $\approx 53\text{m} \times 42\text{m}$

Table 5 – Navigation Time

Environment	Deviation	D*Lite		D*Lite with H. A.	
		Waiting	Navigation	Waiting	Navigation
Simple	3.97m	0s	1:00	5s	0:58
Complex	43.82m	0s	6:00	10s	2:50

Table 5 presents the navigation time for a simple scenario and a complex scenario using *D*Lite* with and without human assistance. The first column shows the type of the environment. The second column shows the deviation length. Then we show the results for *D*Lite*. The column *Waiting* shows the time that the robot waited for help, in this case is 0 as the robot always decide to take the replanned path. The column *Navigation* shows the total navigation time to reach the destination since the beginning, which were 1:00 minute for the simple scenario and 6:00 minutes for the complex scenario. Finally, we present the results for *D*Lite with Human Assistance*. The robot waited 5 seconds in the simple scenario and 10 seconds in the complex scenario to a human move the blocking objects away. Thus, the total navigation time for the simple and complex scenarios were respectively 0:58 minutes and 2:50 minutes.

As we can see, for small deviations the navigation time is almost the same. However, for long deviations the human assistance can decrease the navigation time and eventually save robot's resources such as battery.

4.3 Summary

In this chapter, we presented our work on navigation for autonomous mobile robots in indoor human environments. We divided the navigation task into two problems, Obstacle

Avoidance and Path Planning.

For the Obstacle Avoidance problem, the objective is to explore an environment without hitting any of the obstacles detected by the robot. However, the uncertainties on sensors can affect its controllers. There are many techniques that could be applied to detect obstacles and move the robot at the same time. But the use of only one technique could not be efficient to solve the entire problem. Therefore, Hybrid Intelligent Systems (HISs) have been widely used since they are able to combine the advantages of several techniques to solve different parts of the problem.

Comparative experiments were made with two controllers using a virtual *E-Puck* robot on *Webots* platform. The MLP controller uses a Multiple Layer Perceptron (MLP) network to collect the data of infrared sensors and select the most appropriate movement to the robot. The other controller is a modular HIS based on Fuzzy Logic (FL) and Artificial Neural Network (ANN). The Fuzzy Module uses the Fuzzy Lattice Reasoning (FLR) algorithm to transform the sensors data into symbolic variables that show whether there is an obstacle around the robot. These variables are used by a MLP Module that chooses the robot's movement.

The controllers were created and trained using *Weka* toolbox, and tested in different environments on *Webots*. Both approaches produced good results. Whereas the MLP controller avoids obstacles in most of the time, the HIS controller produces a safer and smoother navigation. This occurs because the Fuzzy Module works like a filter that allows a better classification.

It was noticed that the samples generation is extremely important for building more accurate models. For instance, to generate samples for detecting an obstacle in a certain region of the robot, it is necessary to place the object in a position that strongly activates the sensors of that region, otherwise, the model can have a low accuracy.

Although it was created a system with neural network and fuzzy rules, it cannot be considered as a neuro-fuzzy system. But this could be an advantage for the problem addressed in this chapter. The values captured by infrared sensors on a real environment can diverge from the ones captured on the virtual environment. Thus, a neuro-fuzzy system must be trained from the scratch while in our approach only the Fuzzy Module must be calibrated or trained again. The proposed technique is intended to be used as a reactive controller so the robot can avoid any obstacle even if it is not moving to a specific place.

For the Path Planning problem, we introduced a dynamic path planner called *D*Lite with Human Assistance* based on *D*Lite* algorithm. *D*Lite* aims to replan efficiently by fixing inconsistencies on edge costs found when the search space changes. Our algorithm extends *D*Lite* to ask for human assistance to move blocking objects on the way during the navigation. When the robot's current path gets blocked, the algorithm replans an alternative path and compares it with the previous one. Then, based on the difference

between these two paths, the algorithm decides if it takes the new path or asks for human assistance. If it chooses to ask for help, the algorithm also computes how long it can wait. During this time, the robot asks for assistance and keeps scanning the environment in order to find better paths. After that, if no better path is found, it executes the alternative path.

We started our investigation by testing path finder algorithms on a system with a Graphical User Interface (GUI) which allows the user to generate grid maps, set start and goal cells, and add static and dynamic objects. Two algorithms were implemented, A^* that replans from scratch when the path is blocked during the navigation and our proposed algorithm, *D*Lite with Human Assistance*. The experiment shows that our approach explores less states to find alternative paths when it replans.

Then, we compared the quality of the paths generated by our algorithm and a relaxed version of A^* . These algorithms were implemented as Global Path Planners on ROS and executed on *Gazebo* simulator using a virtual *Turtlebot*. During the tests *D*Lite with Human Assistance* found shorter paths for almost all instances.

For the last experiment, it was analyzed the total navigation time by applying human assistance in real and virtual environments. When the deviation is too long, the human assistance can decrease a lot the navigation time.

CHAPTER 5

Coordination

A team of mobile robots that work in parallel has the potential to finish a given task faster than a single robot. Examples of such tasks include mapping, searching for objects or persons, and covering of environments in order to execute tasks at specific places. Furthermore, teams of robots introduce redundancy into the system that makes it more robust to failure. When multiple robots jointly solve a task, failing robots can be replaced by other team mates thus avoiding the single point of failure of systems in which only one robot is used. Moreover, multiple robots can join their efforts to accomplish tasks that go beyond the ability of each individual robot.

Like human society, there is collective behaviour in multi-robot environments, which could be either competitive or cooperative. Competitive refers to a situation whereby robots compete against each other to best fulfil their own self-interest, such as robot soccer leagues (RoboCup) (BRUCE et al., 2003). Cooperative refers to a situation whereby multiple robots need to interact together in order to complete a task while increasing the total utility of the system. There are several representative examples of multi-robot cooperation, such as multi-robot localization (NERURKAR; ZHOU; ROUMELIOTIS, 2011) (PROROK; BAHR; MARTINOLI, 2012), multi-robot exploration (PEI; MUTKA, 2012) (FAIGL; KULICH; PŘEUČIL, 2012), multi-robot search and rescue (BALAKIRSKY et al., 2007) (ZLOT; STENTZ, 2005), and multi-robot transportation (YAN; JOUANDEAU; ALI-CHÉRIF, 2012) (COLTIN; VELOSO, 2014).

Despite the type of behaviour, competitive or cooperative, there is a general taxonomy regarding multi-robot coordination approaches divided into centralized and distributed (Figure 63). In a centralized approach there is a central control agent that has the global information about the environment as well as all information about the robots, and which can communicate with all the robots to share them. The central control agent could be a computer or a robot. The advantage of the centralized architecture is that the central control agent has a global view of the world, whereby the globally optimal plans can be produced. On the other hand, distributed approaches do not have a central control agent, such that all the robots are equal with respect to control and are completely autonomous

in the decision-making process.

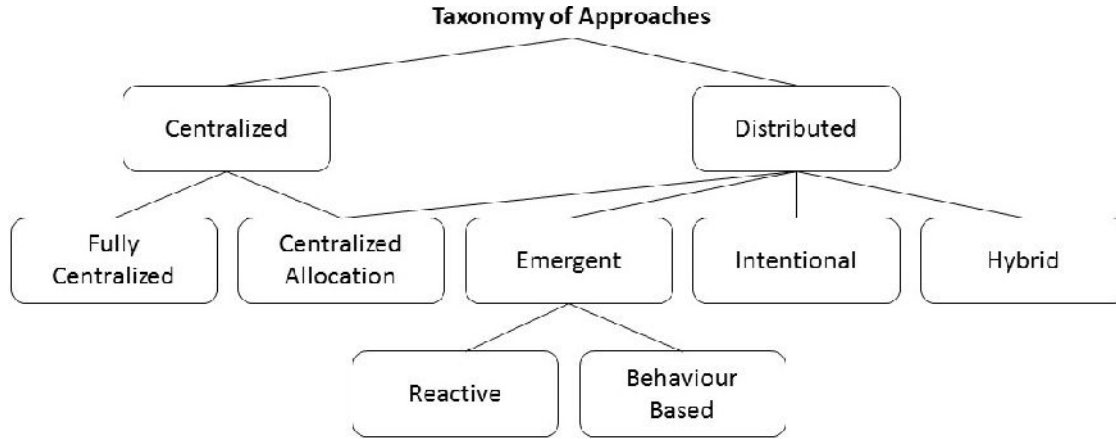


Figure 63 – Taxonomy of Approaches.

Table 6 shows the advantages and disadvantages of each approach presented in this taxonomy.

Table 6 – Comparison of main centralized and distributed approaches.

Approach	Description	Pros	Cons	Example of application
Fully Centralized	single agent plans for the entire team;	potential to be optimal; implicitly encodes coordination;	usually computationally intractable; single point of failure; slow to respond to changes	centralized construction (KHATIB et al., 1996)
Centralized Allocation	single agent assigns tasks to teammates; teammates complete tasks individually;	execution is distributed; allocation can be optimal;	still computationally expensive; still has single point of failure;	Generalized Mission Planner for Multiple Mobile Robots (BRUMITT; STENTZ, 1998)
Reactive	robots have a tight sense-act loop;	extremely fast; very simple;	cannot handle complex tasks;	GOFER project: control the operations of many mobile robots in an indoor environment (CALOUD et al., 1990)
Behaviour Based	use state information to choose actions;	fast and simple; robots can contribute to multiple tasks; more expressive than reactive;	still cannot plan;	legged robot soccer team (VELOSO et al., 2002)
Intentional	communication with the intent to coordinate;	facilitates planning and scheduling;	slow in time-critical situations; very dependent on communication;	dynamic task allocation among groups of heterogeneous agents (GERKEY; MATARIĆ, 2000)
Hybrid	emergent approach in larger intentional;	allows better planning/distribution of resources; can have tight coordination;	cannot have complex interactions;	TRESTLE project: building structures with multiple mobile robots (HEGER, 2010)

In this thesis, we address the problem of coordinating the visits of a team of robots to specific locations in an environment using the Centralized Allocation approach based on a variant of the Traveling Salesman Problem (TSP).

The problem of scheduling visits for a single robot can be seen as an instance of the Traveling Salesman Problem (TSP), which is a combinatorial optimization problem. Combinatorial optimization problems are problems that attempt to find an optimal solution from a finite set of solutions. In the TSP, given a set of n cities and the cost of traveling (or distance) between each possible pair, the goal is to find the best possible way of visiting all cities that minimizes the final traveling cost (DAVENDRA, 2010).

The TSP representation is useful to model practical problems in areas such as logistics, games, space exploration, robotics, etc. In these cases, a city could be a specific point of the problem where an agent must go, and the distance could be the time or the cost of sending this agent to that point. While the TSP is restricted to a single salesman, the multiple Traveling Salesmen Problem (mTSP) generalizes the problem for multiple salesmen, which is more common in real world applications, such as:

- *Print Scheduling*: print press scheduling (GORENSTEIN, 1970), pre-print advertisement scheduling (CARTER; RAGSDALE, 2002);
- *Workforce Planning*: bank crew scheduling (SVESTKA; HUCKFELDT, 1973), technical crew scheduling (LENSTRA; KAN, 1975), photographer team scheduling (ZHANG; GRUVER; SMITH, 1999), interview scheduling (GILBERT; HOFSTRA, 1992), workload balancing (OKONJO-ADIGWE, 1988), security service scheduling (CALVO; CORDONE, 2003);
- *Transportation Planning*: school bus routing (ANGEL et al., 1972) (ORLOFF, 1974), management of pickup and delivery (CHRISTOFIDES; EILON, 1969) (SAVELSBERGH; SOL, 1995), crane scheduling (KIM; PARK, 2004);
- *Mission Planning*: planning of autonomous mobile robots (BRUMITT; STENTZ, 1997) (YU et al., 2002) (BASU; ELNAGAR; AL-HAJJ, 2000), planning of unmanned air vehicles (RYAN et al., 1998);
- *Production Planning*: hot rolling scheduling (TANG et al., 2000);
- *Satellite Systems*: designing satellite surveying systems (SALEH; CHELOUAH, 2004).

Thus, the mTSP consists of finding routes for m salesmen, who start and finish their routes at a single spot, known as depot. Also, each city must be visited exactly once by any salesman, and the total distance of visiting all cities should be minimized.

There are three main strategies to solve mTSP. *Exact methods* which guarantee to give an optimum solution of the problem, *heuristic methods* that only attempt to yield

a good but not necessarily optimum solution, and *transformation techniques* that aim to transform the problem to a standard TSP thus being able to use any algorithm developed for the latter to be used to obtain a solution to the former. Table 7 summarizes the existing algorithms proposed for the mTSP.

Table 7 – Strategies and Algorithms for mTSP (BEKTAS, 2006).

Strategy	Algorithm
Exact Methods	Integer linear programming formulations (KULKARNI; BHAVE, 1985) (KARA; BEKTAS, 2006)
	Cutting plane (LAPORTE; NOBERT, 1980)
	Branch and Bound (ALI; KENNINGTON, 1986) (GROMICHO; PAIXÃO; BRONCO, 1992)
	Lagrangian relaxation + branch and bound (GAVISH; SRIKANTH, 1986)
Heuristic Methods	Simple heuristics (RUSSELL, 1977) (POTVIN; LAPALME; ROUSSEAU, 1989)
	Evolutionary algorithm (DB, 1990)
	Simulated annealing (SONG; LEE; LEE, 2003)
	Tabu search (RYAN et al., 1998)
	Genetic algorithms (ZHANG; GRUVER; SMITH, 1999) (YU et al., 2002) (TANG et al., 2000)
	Neural networks (MODARES; SOMHOM; ENKAWA, 1999) (TORKI; SOMHON; ENKAWA, 1997) (VAKHUTINSKY; GOLDEN, 1994)
Transformations	Asymmetric mTSP to asymmetric TSP (BELLMORE; HONG, 1974)
	Symmetric mTSP to symmetric TSP (HONG; PADBERG, 1977) (RAO, 1980) (JONKER; VOLGENANT, 1988)
	Multidepot mTSP to TSP (LAPORTE; NOBERT; TAILLEFER, 1988) (GUOXING, 1995)

TSP belongs to the class of NP-complete problems, because the time increases exponentially for every increase in n . But, mTSP is even more complex as it requires the evaluation and sorting of nodes in the route of each salesman, which makes it NP-hard (KIRALY; ABONYI, 2010). Therefore, approximation and heuristic methods, such as Genetic Algorithms (GAs), are more likely to be used for this kind of problem.

As we mentioned before, the main goal of mTSP is to minimize the total distance traveled by the salesmen team. However, taking into account only the total distance, the distance traveled individually by each salesman can become imbalanced. For example, when a salesman travels a long distance, the cost of using this salesman to visit all cities in that region is less than the cost of sending another salesman to visit some of them. As a result, some salesmen might travel for long distances, while others just stay around the depot. In this sense, in our work, we aim to optimize the overall distance and also the balance of individual distances, which are opposite objectives. Sometimes, when the total distance is minimized, the individual distances tend to get imbalanced; and when the individual distances are forced to be balanced, the total distance increases.

The purpose of this chapter is to describe the development of a centralized system to schedule routes for a team of robots in indoor environments by means of mTSP. In

this case, robots are salesmen and cities are locations to be visited. We modeled mTSP constraints into GAs components, so they can be used to generate routes for the robots in order to cover all desired locations, while the total distance and the balance of the routes are optimized simultaneously. Two types of GAs were developed. The first one is a multi-objective GA that uses the sum of the route distance of the salesmen to estimate the total distance, and the standard deviation of the routes to estimate the balance. The second is a mono-objective GA with a fitness function that combines these two objectives by the use of a parameter.

Several experiments were performed using different combinations of genetic operators for crossover and selection. The system was tested using indoor real maps acquired by robots and the generated routes were sent to multiple virtual robots on a simulated environment. Although the results show that both GAs can generate good solutions, the Pareto-optimal set of the multi-objective GA tends to keep more balanced solutions for any instance of the problem.

The remainder of the chapter is organized as follows. Section 5.1 describes some related works. The proposed approaches for solving the mTSP, considering multiple objectives, are described in Section 5.2. Section 5.3 presents the development of a system to schedule routes for robots using occupancy grid maps. The description of the experiments and their results are provided in Section 5.4. Finally, a summary is presented in Section 5.5.

5.1 Related Work

There are several real world problems based on the traditional TSP. They are also too complex, and for this reason methods such as GAs are often used to solve them. For instance, the Pickup and Delivery Problem combines vehicle routing and object distribution. The goal is to find the optimal visiting schedules for vehicles to convey passengers or commodities from pickup nodes to delivery nodes. This problem is addressed in (LIAO; CHIEN; TING, 2014), where the authors have developed two techniques to improve the performance of a GA. The Reversely Weighting technique, responsible for evaluate the solutions, accumulates the proportion of supplies and weights each edge in reverse order to build a table. Then, a crossover technique, called Edge Aggregate Crossover, uses that table to choose edges for offspring according to probable effects occurred in parents.

A GA is proposed in (YU; LU, 2014) for planning the routes of an automatic garment cutter machine. It aims to reduce the size of the cut path and improve the smoothness of movement. The authors have made some improvements regarding the operators of mutation and crossover to accelerate the convergence and avoid local optima.

(SAKURAI et al., 2010a) and (SAKURAI et al., 2010b) presented real-time GAs to schedule delivery routes using the TSP representation. This real-time scheduling is important as some constraints can appear during the execution, making the current schedule

infeasible. Thus, the GAs were modified in order to backtrack.

To solve the mTSP, a hybrid algorithm that combines GA with local search guided by the 2-opt heuristic is proposed by (SEDIGHPOUR; YOUSEFIKHOSHBAKHT; DARANI, 2011). Another approach for scheduling delivery routes with GA, using the mTSP representation, is presented by (SADIQ, 2012) which apply a clustering phase to allocate similar packages together before the generation of the routes.

To model an individual for the mTSP, (KIRALY; ABONYI, 2010) use one chromosome for each salesman. But in (ARYA; GOYAL; JAISWAL, 2014), the individual has a single chromosome of length $p + q$ where the nodes are represented by a permutation of the integers from 1 to p . This permutation is partitioned into q sub-tours by the insertion of q negative integers (from 1 to q) that represent the change from one salesman to the next.

These previous works have used mono-objective GAs which attempt to optimize just one objective of the problem. However, there are some problems that require the optimization of multiple objectives simultaneously. In such cases, multi-objective GAs can be applied. For instance, (GARCIA-NAJERA; GUTIERREZ-ANDRADE, 2013) introduced a multi-objective GA to solve the Pickup and Delivery Problem. In their approach three objectives are considered: the number of routes, the total distance; and the time traveled, as sometimes the distance is not proportional to the time spent on a route.

In the context of Multi-Robot Coordination, (HUSSAIN et al., 2002) presented an evolutionary and a hybrid approach to the problem of scheduling routes for a team of robotic agents to perform a resource distribution task in a static environment. The proposed GA breaks down the task of multiple route design into a single TSP as a centralized route planner. (LI; LI, 2016) proposed a GA to allocate tasks at specific locations and plan paths for a team of robots, while it minimizes possible collisions between the robots during the execution of their paths. To coordinate a multi-robot exploration for unknown environments (MA; ZHANG; LI, 2007) developed a GA to allocate target points to multiple robots appropriately, so that the multiple robots simultaneously explore different areas of the environment to guarantee minimum overall exploration time.

5.2 Using GAs to solve the mTSP

GA has a generic architecture. To solve a problem using this architecture, some of its components must be designed considering the constraints of the problem. As components of the architecture we have the structure of the individual, fitness function, and methods for crossover and mutation. This section describes how these components were designed and how the GAs were implemented in order to solve the mTSP.

5.2.1 Individual Structure

The mTSP extends the traditional TSP to a multi-agent model, where many salesmen can be used to visit the n cities described in the problem. Thus, each city must be visited only once by one of the m salesmen. To indicate the positions of the cities on a plan or map, they must have coordinates. The coordinates are important especially to estimate the distance between a pair of cities. Therefore, an individual can be modeled with one chromosome as shown in Figure 64.

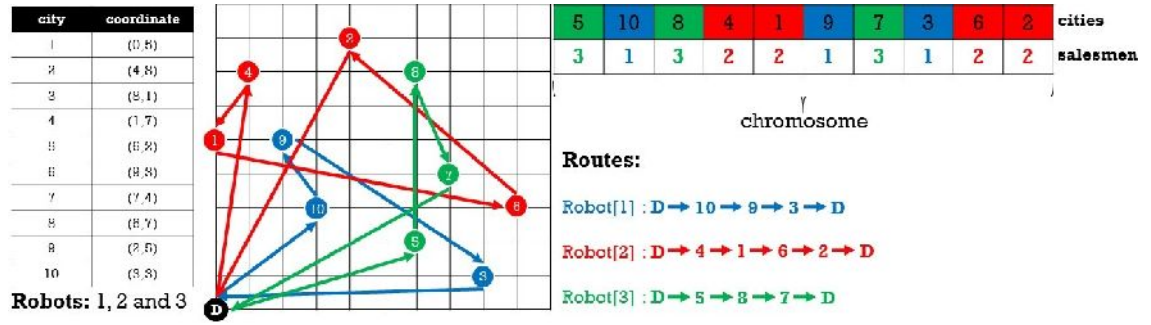


Figure 64 – Example of an individual.

The length of the chromosome is equal to the number of cities, as all of them must be visited only once. Each gene of the chromosome relates a salesman with a city that he must visit. The order of the cities on the chromosome is also important as it shows the sequence in which the salesmen will perform their visits, i.e., the route of each salesman.

5.2.2 Crossover & Mutation

The mTSP is a permutation problem as the cities should not be repeated within the chromosome. Thus, methods for crossover and mutation must consider this constraint. The methods Partially Matched Crossover (PMX) and Cycle Crossover (CX) are widely used in such cases.

In the PMX, a section of the chromosomes (through 2 cut points) is selected, where the genetic material of both parents will be fully exchanged. Then, some changes are made in the offspring to eliminate repeated genes. On CX, a sequence of exchanges between the parents is made to form a cycle. After that, only the genes that are within the cycle will be exchanged in the offspring. In this way, while CX keeps exactly the genetic structure of the parents, PMX makes some changes to avoid repetition, which can generate different genetic structures.

Regarding mutation, a simple permutation is done between two random genes to ensure that each city will appear only once in the chromosome.

5.2.3 Objectives

The main objective of mTSP is to minimize the total distance traveled by all salesmen. However, it may cause an imbalance in the distance traveled by them.

This may occur when a salesman travels far away to visit a city. Since mTSP aims to reduce the total distance, this salesman tends to cover all the cities around, as sending another salesman there, for visiting just a few cities, can increase the total distance. As a result, some salesmen might travel much more than others.

For some real problems, this behavior is not suitable. For example, in a delivery company, there may be some overloaded employees making many deliveries and driving long distances, while others stand nearby the company.

Therefore, we proposed the use of two objectives for the problem. The first is responsible for minimizing the total distance traveled by the set of salesmen, while the second aims to reduce the difference of the distance traveled individually by them.

In order to estimate the total distance traveled by all salesmen, it is calculated the sum of the Euclidean distances between the cities' locations according to the route of each salesman, as defined by the following formula:

$$\Delta = \sum_{i=1}^m \sum_{j=1}^n \sqrt{(x_j - x_{j+1})^2 + (y_j - y_{j+1})^2}, \forall j \in Cities_i$$

To measure how balanced the routes are, the standard deviation is calculated over the distance traveled by each salesman. The Euclidean distance is also used to estimate the distance d_i traveled by the salesman i and to calculate the average distance \bar{X} . A low value of standard deviation means that the routes are balanced.

$$\sigma = \sqrt{\frac{\sum_{i=1}^m (d_i - \bar{X})^2}{m - 1}}$$

5.2.4 mono-objective GA

The mono-objective GA described here is inspired by the traditional GA (GOLDBERG, 1989), as shown in Algorithm 8.

Algorithm 8 mono-objective GA

- 1: Create T_p random individuals for the initial population
 - 2: Evaluate the initial population
 - 3: **for** generation $t \leftarrow 1..N_{iter}$ **do**
 - 4: Select P_{rec} individuals
 - 5: Crossover
 - 6: Mutation - with probability P_{mut}
 - 7: Evaluate the new individuals
 - 8: Select the best T_p individuals
 - 9: **end for**
 - 10: Return the best individuals
-

First, an initial population is generated randomly with T_p individuals and submitted to the evaluation process. Each individual represents a route scheduling for m salesmen passing through n cities without repetition.

The fitness function F used in this GA combines both objectives, the total distance Δ and the standard deviation σ . The standard deviation is multiplied by a parameter α to weigh its relevance, as shown in the following formula:

$$F = \Delta + (\sigma * \alpha)$$

The main loop runs for N_{iter} iterations, where genetic operations for selection, crossover, and mutation are made. For the next step, $Prec$ individuals must be selected from the current population. We implemented three selection methods: Roulette Wheel, Tournament, and Stochastic Tournament.

In the Roulette Wheel, each individual of the current population is represented by a value proportional to its fitness. It means that, individuals with high values have more chance to be selected. In the Tournament, many subsets are created from the current population. For each one, the individual with the highest fitness value is selected. The Stochastic Tournament can be seen as a fusion of these two approaches. First, some individuals are selected through the Roulette Wheel to create subsets. Then, Tournaments are made on these subsets to select the best individuals.

After the selection, the crossover process begins. We implemented two crossover methods: CX and PMX. These methods prevent the generation of invalid individuals as they perform safe permutations between the chromosomes.

Next, some of the new individuals are submitted to the mutation process with probability $Pmut$. The mutation method applies a simple permutation between two random genes of the individual, which also avoids the generation of invalid individuals.

Finally, the new individuals are evaluated and inserted into the current population. This population is sorted according to the fitness values. The best T_p individuals, in each iteration, are selected to form the new population that will perform again the processes of selection, crossover, and mutation.

5.2.5 multi-objective GA

So far, we address a problem where two objectives must be optimized simultaneously. Multi-objective optimization problems sometimes require specific techniques, such as multi-objective GA.

In many problems, the objectives can conflict with each other, and optimizing a particular solution with respect to an objective can result in unacceptable results with respect to the other objectives.

This behavior can be seen in the mTSP. If we choose to reduce only the total distance, as we cited before, the standard deviation of the route distances increases, as some salesmen tend to travel more than others. On the other hand, if we try to minimize the standard deviation, a salesman may not care about traveling more than necessary just to have his route distance similar to the others, which increases the total distance.

A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution. A solution is said to be dominated if there is another solution that is at least equal to it in all objectives, but better in at least one of them. A solution is said to be Pareto-optimal if it is not dominated by any other solution in the solution space.

The multi-objective GA implemented in this thesis is based on the Strength Pareto Evolutionary Algorithm II (SPEA2) (ZITZLER; LAUMANN; THIELE, 2001) which is an elitist multi-objective evolutionary algorithm. GAs multi-objective that use elitist strategies tend to outperform their non-elitist counterparts. This algorithm has two populations, Q to store the Pareto-optimal solutions, and P to store solutions generated during an iteration. The Algorithm 9 shows its pseudo-code.

Algorithm 9 multi-objective GA

```

1: Create  $N_{ind}$  random individuals for  $P_1$ 
2:  $Q_1 \leftarrow \emptyset$ 
3: for generation  $t \leftarrow 1..N_{iter}$  do
4:   for individual  $i \in R_t = P_t \cup Q_t$  do
5:     Calculate  $strength_i$ 
6:     Calculate  $raw_i$ 
7:     Calculate  $dens_i$ 
8:     Calculate  $F_i$ 
9:   end for
10:  Move individual  $i \in R_t$  to  $Q_{t+1}$  since  $F_i < 1$ 
11:  if  $|Q_{t+1}| < N_{ext}$  then
12:    Sort  $R_t$  in ascending order by the values of  $F_i$ 
13:    Move the first  $N_{ext} - |Q_{t+1}|$  individuals of  $R_t$  to  $Q_{t+1}$  since  $F_i \geq 1$ 
14:  end if
15:  if  $|Q_{t+1}| > N_{ext}$  then
16:    Reduce  $Q_{t+1}$ 
17:    Move the first  $N_{ext} - |Q_{t+1}|$  individuals of  $R_t$  to  $Q_{t+1}$  since  $F_i \geq 1$ 
18:  end if
19:  Generate new population  $P_{t+1}$  by applying the genetic operators on  $Q_{t+1}$ 
20: end for
21:  $Q_{final} \leftarrow Q_{t+1}$ 

```

First, the population P_1 is generated randomly with N_{ind} individuals, like the initial population on the mono-objective GA, while the population Q_1 stays empty.

Then, the evolutionary process runs for N_{iter} iterations. During this process, the populations P and Q are put together in R . From now, for each individual in R , the algorithm calculates the values of $strength$, raw , $dens$ and F , used to evaluate the individual.

The $strength$ counts how many individuals are dominated by the current individual. The raw sums the strengths of the individuals that dominate the current individual. The $dens$ value is used to better spread the individuals, especially in the Pareto-optimal set. Finally, F represents the evaluation of the current individual, where $F = raw + dens$.

All individuals whose $F < 1$ must belong to the Pareto front, thus they are moved to Q . However, Q has a fixed size of N_{ext} individuals. Therefore, if the size of Q exceeds N_{ext} then Q must be reduced by the means of a truncation operator, otherwise, if the size of Q is less than N_{ext} then Q must be filled with the best dominated individuals that remain in R .

Finally, the processes of selection, crossover, and mutation are made on Q to generate the new population P , as it is done on the mono-objective GA.

5.3 Route Scheduling for a Team of Robots in Indoor Environments

Using the Genetic Algorithms described in Section 5.2, we built a system to schedule routes for a team of robots on indoor maps. Such maps are occupancy grid maps acquired by robots using laser mapping techniques presented in Section 3.1.2.

By the means of a Graphical User Interface (GUI) available in the system, a user can choose the image file of a map, set the depot location, and also select locations that must be visited by the robots. However, since these maps are not open plans, we cannot use straight lines as paths between locations. Instead, real paths must be computed for each pair of locations considering the structure of the map, creating a path matrix that will be used in the scheduling process.

Furthermore, planning paths on raw images of occupancy grid maps cannot guarantee the generation of feasible or even safe paths, as they do not take into account the shape of the robots. Therefore, an inflation technique is applied on the raw images based on the size of the robots so the scheduling can generate paths that fit the robots.

5.3.1 Indoor Robot Maps

In our experiments, we used two laser maps, one we acquired in our Laboratory (see Section 3.1.2) and another from an entire floor at Willow Garage Inc. Before using these occupancy grid maps on our system, we applied an inflation technique, available on ROS Inflation Costmap Plugin, based on the size of our Turtlebot to ensure the generation of feasible paths along the map.

The inflation process inserts a buffer zone around each lethal obstacle. Locations where the robot would definitely be in collision are marked with a lethal cost, and the immediately surrounding areas have a small non-lethal cost. These values ensure the robot does not collide with lethal obstacles, and prefers not to get too close. This process involves the propagation of cost values out from occupied cells that decrease with distance. For this purpose, it is defined five specific symbols for costmap values as they relate to a robot:

- ❑ *Lethal* cost means that there is an actual (workspace) obstacle in a cell. So, if the robot's center were in that cell, the robot would obviously be in collision.
- ❑ *Inscribed* cost means that a cell is less than the robot's inscribed radius away from an actual obstacle. So, the robot is certainly in collision with some obstacle if the

robot's center is in a cell that is at or above the inscribed cost.

- ❑ *Possibly circumscribed* cost is similar to *inscribed*, but using the robot's circumscribed radius as cutoff distance. Thus, if the robot center lies in a cell at or above this value, then it depends on the orientation of the robot whether it collides with an obstacle or not. The term *possibly* is used because it might be that it is not really an obstacle cell, but some user-preference, that put that particular cost value into the map. For example, if a user wants to express that a robot should attempt to avoid a particular area of a building, they may inset their own costs into the costmap for that region independent of any obstacles.
- ❑ *Freespace* cost is assumed to be zero, and it means that there is nothing that should keep the robot from going there.
- ❑ *Unknown* cost means there is no information about a given cell. The user of the costmap can interpret this as they see fit.
- ❑ All other costs are assigned a value between *Freespace* and *Possibly circumscribed* depending on their distance from a *Lethal* cell and the decay function provided by the user.

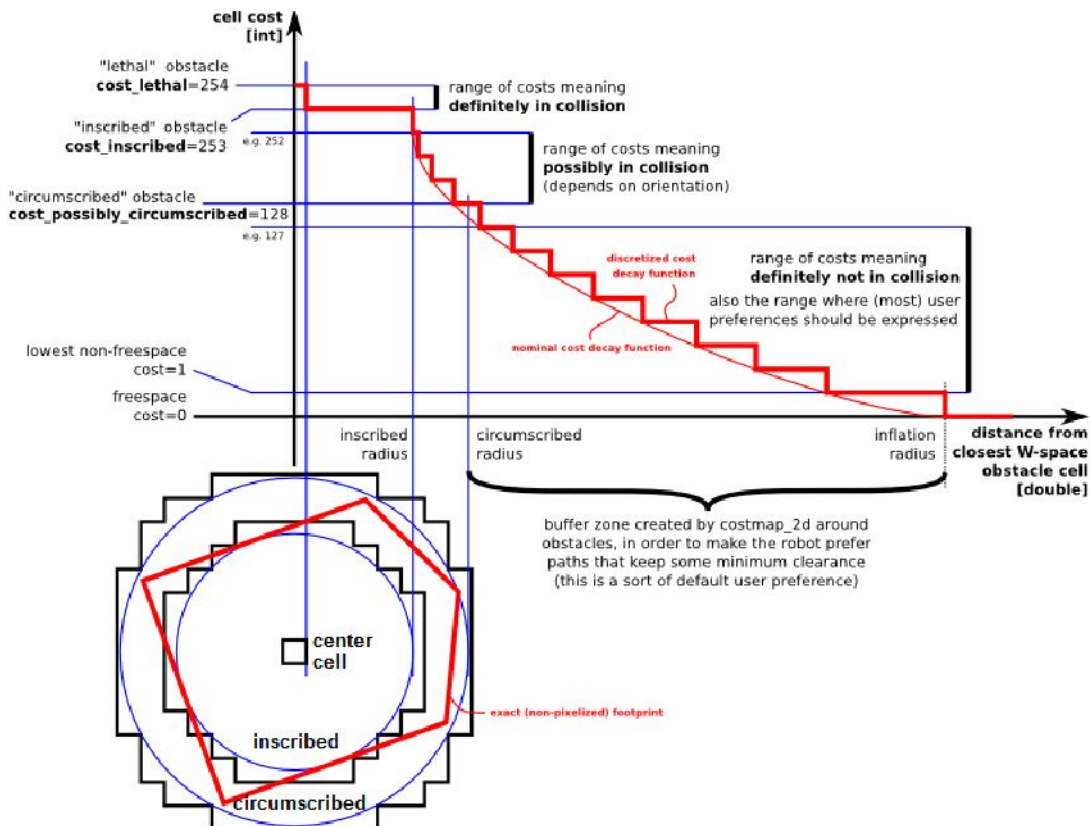
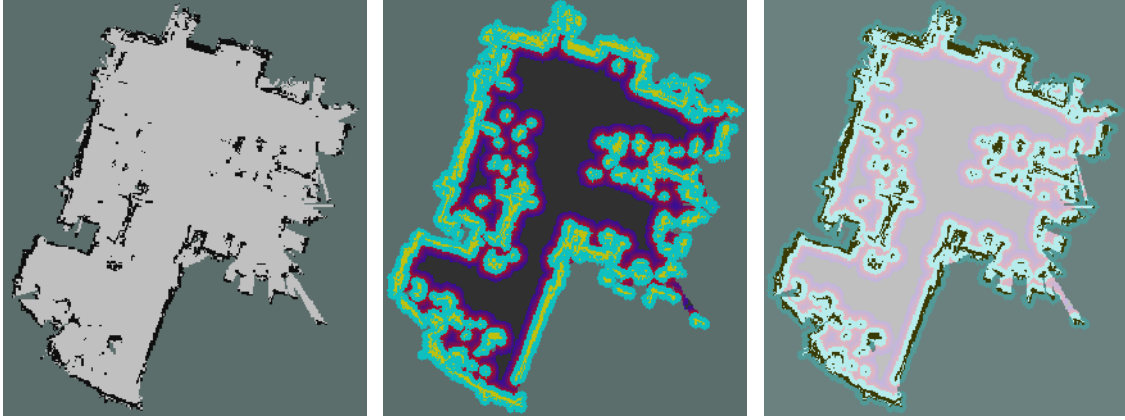


Figure 65 – Inflation. (available online on ROS Costmap: Inflation)

Figures 66 and 67 show both occupancy grid maps. The first images (Figures 66 (a) and 67 (a)) are the occupancy grids as acquired by robot laser mapping. Figures 66 (b) and 67 (b) highlight the inflation on occupied cells, where *lethal* cost is shown in yellow, *inscribed* cost in green, *possibly circumscribed* cost in different shades of purple, and *freespace* cost in dark gray. The last images (Figures 66 (c) and 67 (c)) are those used in our system, they are still inflated but less colored.

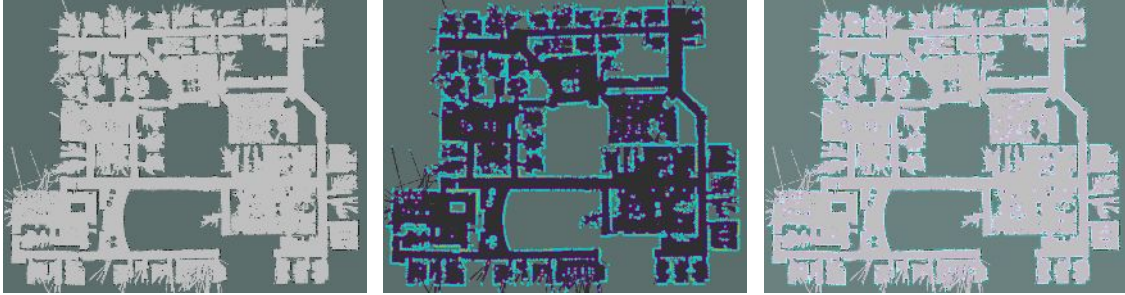


(a) Raw Map.

(b) Inflated Map.

(c) Final Map.

Figure 66 – Indoor Robot Map: Laboratory.



(a) Raw Map.

(b) Inflated Map.

(c) Final Map.

Figure 67 – Indoor Robot Map: Willow Garage Inc.

Using such maps as input, our system computes paths for route scheduling considering only *freespace* cells.

5.3.2 Path Matrix

So far, we used straight lines to represent paths between locations in the multiple Traveling Salesmen Problem. Therefore, our Genetic Algorithms (GAs) minimize the total distance and balance the routes based on Euclidian distance of these locations.

However, in order to schedule routes for robots in real indoor maps, we must compute the actual path between all locations.

Therefore, before scheduling the routes, our system computes a path matrix P using A^* for all pairs of locations, including the depot location, as illustrated by Figure 68. Thus, the dimension of P is $|L| + 1 \times |L| + 1$ where L is the list of locations to be visited. To optimize the computation, the elements of the main diagonal in P are set as being empty, which correspond to paths where the source and destination are the same location. Also, when the element P_{ij} is computed, which corresponds to the path from i to j , the inverse path is automatically stored at P_{ji} .

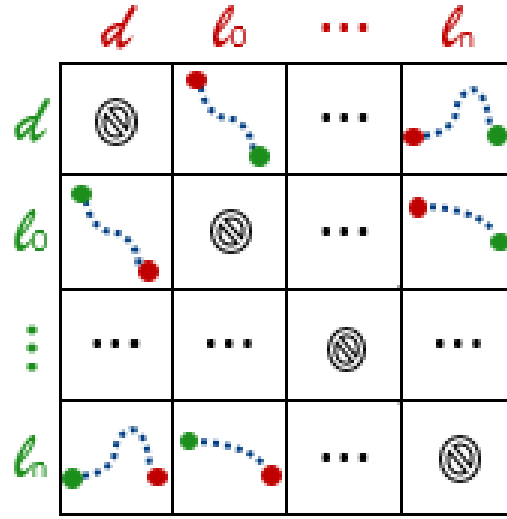


Figure 68 – Path Matrix.

The paths stored in P are used to plot the routes in the GUI of our system, and also to calculate the objectives of the GAs. The route distance traveled by a salesman i is calculated by the sum of path lengths within its route R_i :

$$R_i = \sum_{j=1}^n P_{j,j+1}, \forall j \in Cities_i$$

Therefore, the total distance traveled by all salesmen in a given solution is:

$$\Delta = \sum_{i=1}^m R_i$$

The balance of the routes is measured by the standard deviation of route lengths, where \bar{R} is the average route length.

$$\sigma = \sqrt{\frac{\sum_{i=1}^m (R_i - \bar{R})^2}{m - 1}}$$

5.3.3 Route Scheduling System

Our system allows the user to control and follow all the process of route scheduling. When the user launches the system, the following control panel appears on the screen (Figure 69). The process of route scheduling is divided in four steps.



- 1 Select a Map**
- 2 Add or Remove Locations and Depot**
- 3 Build Path Matrix**
- 4 Select GA, set number of salesmen, and run route scheduling**

Figure 69 – Control Panel.

First, the user looks for the image file of the inflated occupancy grid map by clicking on the *search* button (Figure 70).

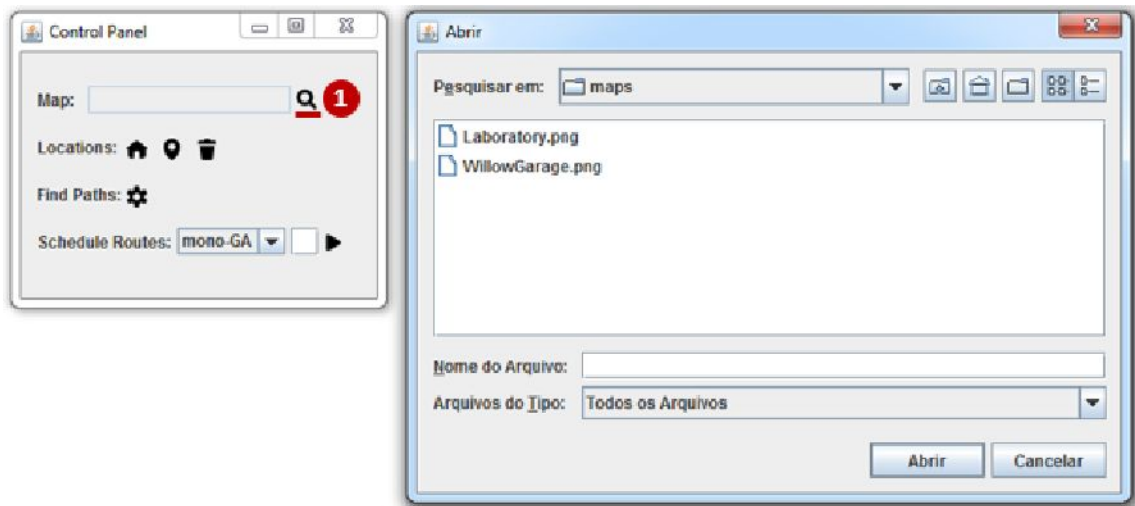


Figure 70 – Select a Map.

Then, the execution panel pops up with the desired map. By selecting the *depot* option, the user can set its location by clicking on the map, as shown in the Figure 71.

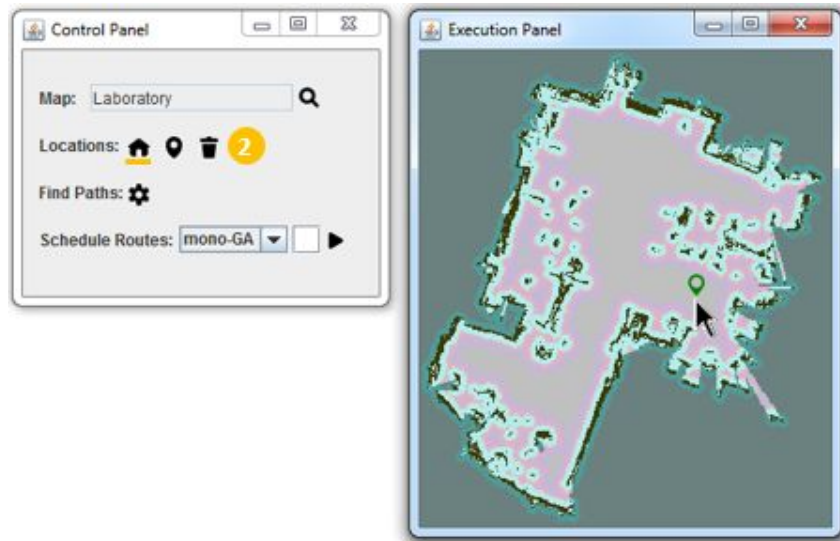


Figure 71 – Set Depot.

To set a location to be visited, the user can select the *location* option and set it by clicking on the map (Figure 72). He can repeat this step as much as necessary to set all desired locations. By clicking on the *delete* option (trash icon), the system removes all locations from the map, including the depot.

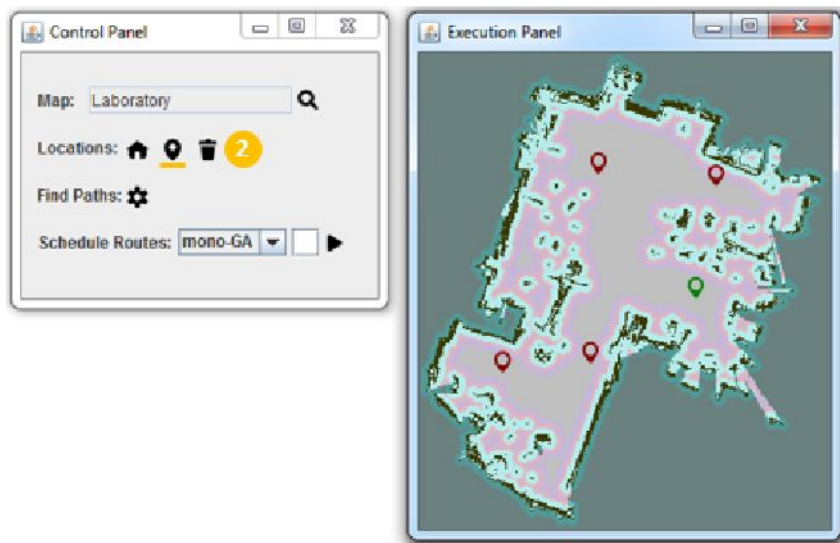


Figure 72 – Set Locations.

After that, it is necessary to create the path matrix used for route scheduling. When the user clicks on the *find paths* option, the system starts computing paths for all locations.

The system plots each path on the map as soon as it is computed by the A* algorithm. A message appears to the user when the process is done (Figure 73).

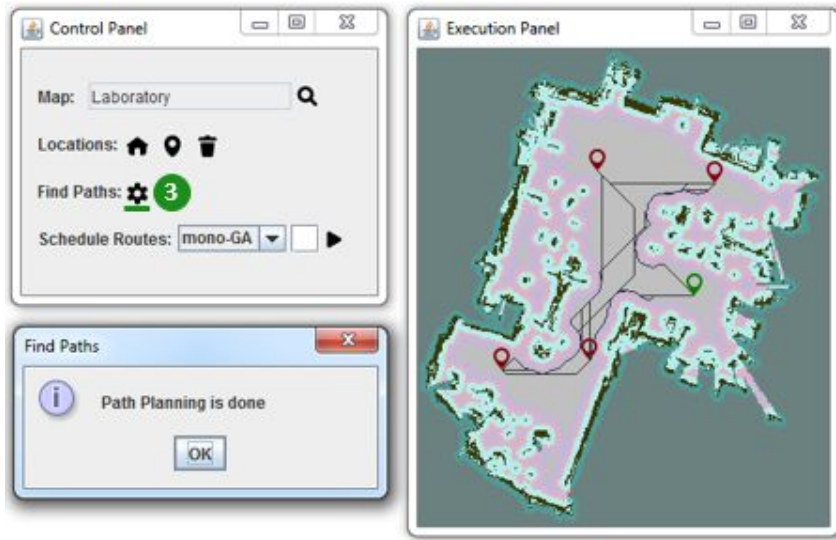


Figure 73 – Find Paths.

To schedule the routes, the user must select between mono-GA or multi-GA, fill the number of salesmen (robots) to be used, and click on *run*. The user can follow the progress from the execution panel as the system plots the best solution found after each iteration of the algorithm. Different colors are used to represent the route of each salesman. A message appears to the user when the route scheduling is done (Figure 74).

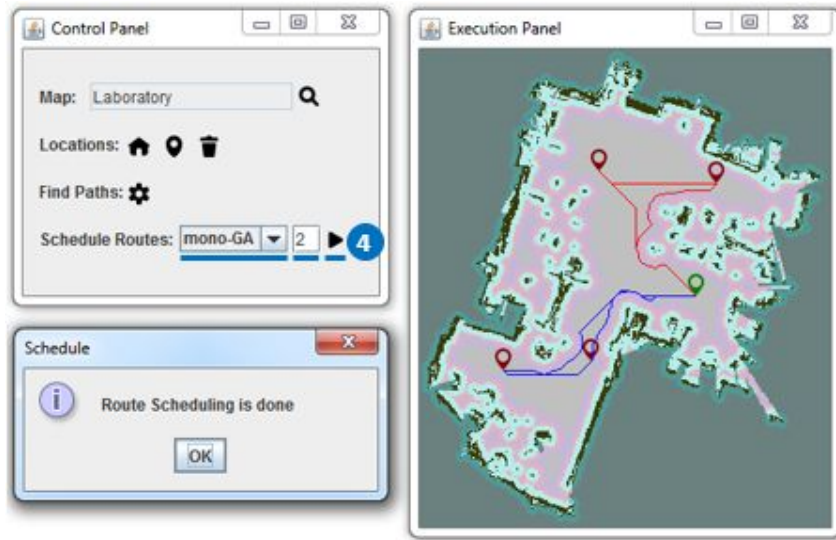


Figure 74 – Route Scheduling.

Finally, the lists of coordinates to be visited in the environment can be sent to virtual and/or real robots via ROS messages.

5.4 Experiments

We started our experiments by running the GAs on open plans using the objectives described in Section 5.2.3, without any specific map. Figure 75 illustrates the execution of multi-objective GA using 3 salesmen and 20 cities. First, the locations are plotted, as depicted by Figure 75 (a), where the black dots represent the cities and the gray dot at the corner represents the depot. Figure 75 (b) presents the best solution found among the population on the first iteration of the algorithm. The routes of the salesmen are displayed in different colors. Finally, the best solution of the last iteration is shown in Figure 75 (c).

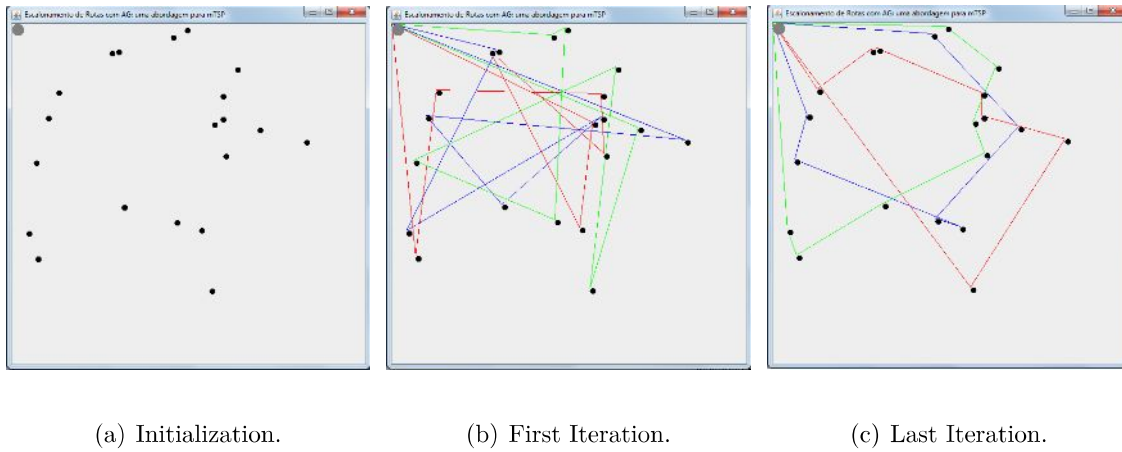


Figure 75 – Route scheduling for the mTSP with multi-objective GA.

The GAs parameters were set to values that have proved to work well on preliminary tests: population size = 160, number of generations = 200, crossover rate = 60%, mutation rate = 30%, and tour = 3. Experiments were carried out by running the multi-objective GA and the mono-objective GA with $\alpha = 1, 10, 20$ and 30.

We created 5 problems for the mTSP:

- ☐ Problem 01: 2 salesmen and 10 cities.
- ☐ Problem 02: 3 salesmen and 20 cities.
- ☐ Problem 03: 5 salesmen and 30 cities.
- ☐ Problem 04: 8 salesmen and 50 cities.
- ☐ Problem 05: 13 salesmen and 80 cities.

Each problem is composed by 100 different instances in relation to cities locations, whose x and y (coordinate) vary from 0 to 500. The coordinate (0,0) represents the depot location, where the salesmen must leave and return after the execution of their routes.

The methods were performed using the following combinations: a) Tournament & PMX, b) Tournament & CX, c) Roulette Wheel & PMX, d) Roulette Wheel & CX, e) Stochastic Tournament & PMX, and f) Stochastic Tournament & CX.

For each problem and combination, a set of 20 solutions was generated from average values given by the best 20 solutions of each instance. These sets are used to build the comparative charts and to calculate the hypervolumes.

The hypervolume is a metric widely used to evaluate multi-objective algorithms. In order to compute the hypervolume, it is required an appropriate reference point. This point is given by the worst values of total distance and standard deviation found. Therefore, regarding the best combination of each approach, we took the non-dominated solutions of each set and computed the hypervolume covered by them.

The following tables show the results of each problem considering the average values of standard deviation and total distance, where the best combination of each approach is presented in bold and plotted on the charts.

Table 8 – Problem 01 - 2 salesmen and 10 cities

Method	Selection	Crossover	$\bar{\Delta}$	$\bar{\sigma}$
mono GA $\alpha = 1$	Tournament	PMX	2257,32	41,53
		CX	2263,77	47,34
	Roulette Wheel	PMX	2256,83	33,47
		CX	2254,69	40,82
	Stochastic Tournament	PMX	2260,86	25,48
		CX	2259,62	27,56
mono GA $\alpha = 10$	Tournament	PMX	2346,11	0,80
		CX	2387,57	1,19
	Roulette Wheel	PMX	2324,45	0,67
		CX	2345,93	0,91
	Stochastic Tournament	PMX	2329,81	1,53
		CX	2324,66	1,06
mono GA $\alpha = 20$	Tournament	PMX	2378,36	0,76
		CX	2457,55	1,08
	Roulette Wheel	PMX	2344,11	0,51
		CX	2384,98	0,99
	Stochastic Tournament	PMX	2348,37	1,28
		CX	2345,61	1,24
mono GA $\alpha = 30$	Tournament	PMX	2399,69	0,81
		CX	2468,35	1,20
	Roulette Wheel	PMX	2361,00	0,60
		CX	2415,22	0,83
	Stochastic Tournament	PMX	2384,73	1,61
		CX	2358,03	1,04
multi GA	Tournament	PMX	2061,34	460,18
		CX	2129,64	405,60
	Roulette Wheel	PMX	2120,34	422,56
		CX	2268,90	330,34
	Stochastic Tournament	PMX	2062,61	492,79
		CX	2201,73	344,04

Table 8 shows the results of the proposed GAs for Problem 01. To choose the best combination of selection and crossover for each method, it was analyzed the dominance of all solutions.

The following list shows the methods ordered by their hypervolumes:

- ❑ mono GA $\alpha = 1$ (RouletteWheel&PMX): hypervolume = 45248
- ❑ mono GA $\alpha = 10$ (Roulette Wheel&PMX): hypervolume = 17464
- ❑ multi GA (Tournament&PMX): hypervolume = 9365
- ❑ mono GA $\alpha = 20$ (RouletteWheel&PMX): hypervolume = 8496
- ❑ mono GA $\alpha = 30$ (RouletteWheel&PMX): hypervolume = 472

As presented by the chart of Figure 76, the solutions found with the multi-objective GA are better distributed than the solutions found with the mono-objective GA, however, the hypervolumes of the mono-objective GA with $\alpha = 1$ and 10 were greater.

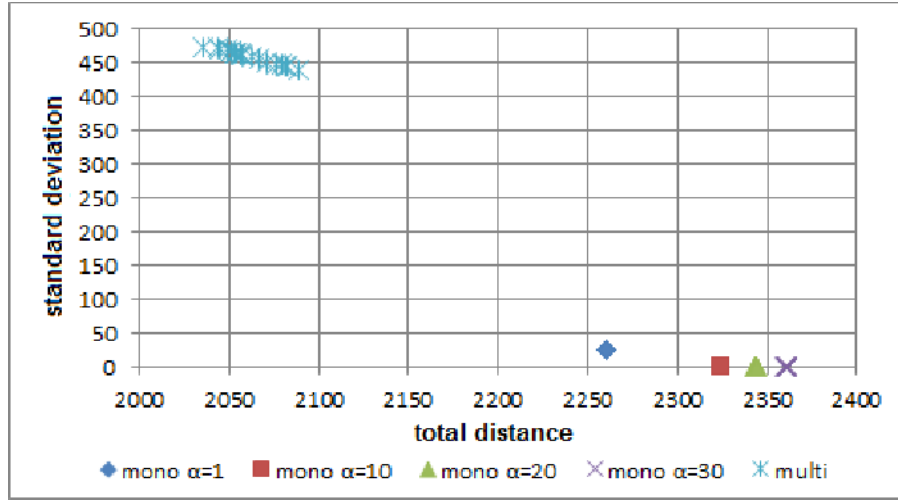


Figure 76 – Problem 01: solution sets given by the best combination of each method.

According to Table 9, which shows the results of Problem 02, the best combination for the multi-objective GA is Stochastic Tournament with PMX. The combination of Roulette Wheel with PMX keeps generating the best solutions for the mono-objective GA.

Table 9 – Problem 02 - 3 salesmen and 20 cities

Method	Selection	Crossover	Δ	$\bar{\sigma}$
mono GA $\alpha = 1$	Tournament	PMX	3401,12	184,65
		CX	3578,57	170,87
	Roulette Wheel	PMX	3236,11	207,34
		CX	3457,49	154,82
	Stochastic Tournament	PMX	3279,52	203,00
		CX	3268,73	227,45
mono GA $\alpha = 10$	Tournament	PMX	4561,55	3,95
		CX	4825,14	3,81
	Roulette Wheel	PMX	4242,20	3,98
		CX	4580,22	4,00
	Stochastic Tournament	PMX	4557,09	27,09
		CX	4658,80	25,08
mono GA $\alpha = 20$	Tournament	PMX	4989,96	3,05
		CX	5165,81	3,31
	Roulette Wheel	PMX	4721,71	3,11
		CX	4997,40	3,09
	Stochastic Tournament	PMX	5003,77	16,91
		CX	5013,01	13,88
mono GA $\alpha = 30$	Tournament	PMX	5142,25	2,27
		CX	5349,96	3,08
	Roulette Wheel	PMX	4903,68	3,20
		CX	5143,52	2,46
	Stochastic Tournament	PMX	5234,93	12,45
		CX	5191,77	9,38
multi GA	Tournament	PMX	3991,75	119,94
		CX	4089,77	116,53
	Roulette Wheel	PMX	4077,41	104,63
		CX	4325,79	103,37
	Stochastic Tournament	PMX	3562,56	117,66
		CX	3924,21	79,63

Figure 77 presents a small increase in the diversity of the solutions found by the mono-objective GA. On the other hand, the hypervolume of multi-objective GA was only 2% lower than the best value found, as shown in the list below:

- mono GA $\alpha = 10$ (RouletteWheel&PMX): hypervolume = 138507
- multi GA (StochasticTournament&PMX): hypervolume = 136434
- mono GA $\alpha = 20$ (RouletteWheel&PMX): hypervolume = 45478
- mono GA $\alpha = 30$ (RouletteWheel&PMX): hypervolume = 7140

□ mono GA $\alpha = 1$ (RouletteWheel&PMX): hypervolume = 0

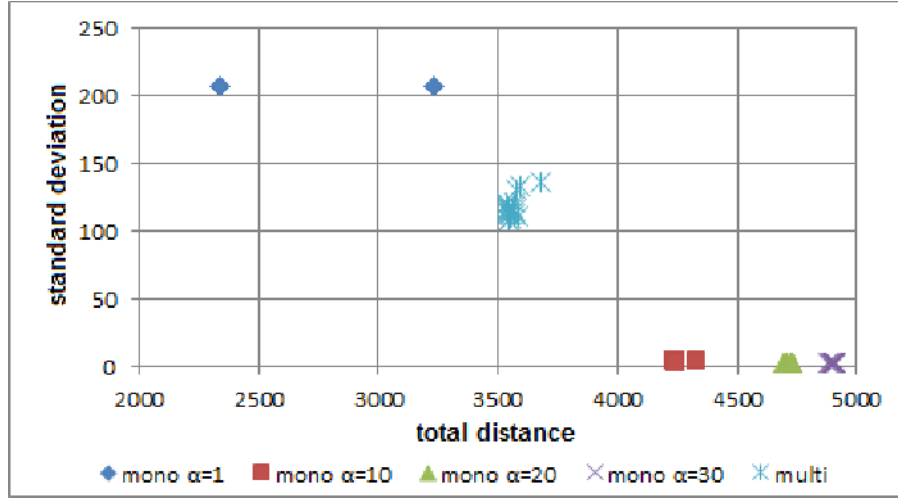


Figure 77 – Problem 02: solution sets given by the best combination of each method.

The combinations that generate the best results for Problem 3 are the same of Problem 2. As we can see in Table 10, the mono-objective GA prioritizes only one objective at a time. For example, when the total distance has a low value the standard deviation has a high one.

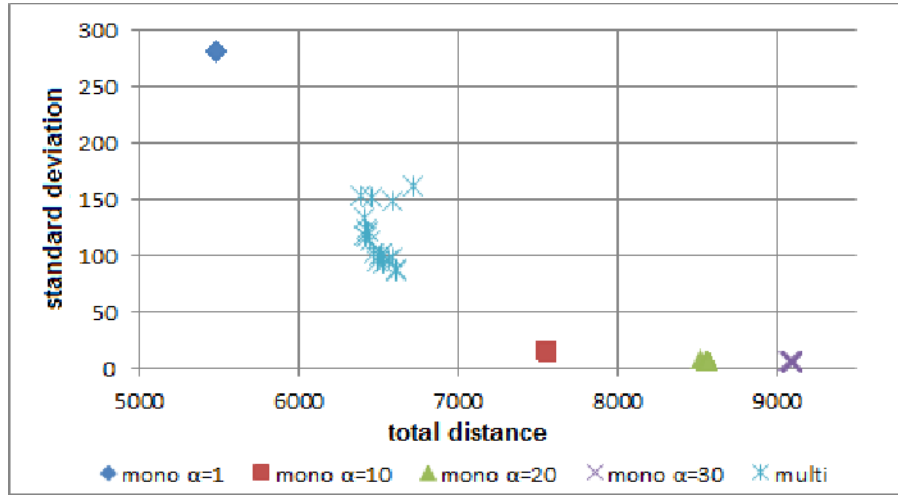


Figure 78 – Problem 03: solution sets given by the best combination of each method.

The chart of Figure 78 depicts this behavior. For $\alpha = 1$, the mono-objective GA finds the lowest total distance, but also the highest standard deviation. The opposite happens with $\alpha = 10, 20$ and 30 . However, the multi-objective GA produces well spread solutions at the center, which reflects on its hypervolume:

□ multi GA (StochasticTournament&PMX): hypervolume = 523521

- mono GA $\alpha = 10$ (RouletteWheel&PMX): hypervolume = 416395
- mono GA $\alpha = 20$ (RouletteWheel&PMX): hypervolume = 159177
- mono GA $\alpha = 30$ (RouletteWheel&PMX): hypervolume = 8280
- mono GA $\alpha = 1$ (RouletteWheel&PMX): hypervolume = 3623

Table 10 – Problem 03 - 5 salesmen and 30 cities

Method	Selection	Crossover	Δ	$\bar{\sigma}$
mono GA $\alpha = 1$	Tournament	PMX	6058,42	219,13
		CX	6517,61	227,73
	Roulette Wheel	PMX	5477,15	282,12
		CX	6022,91	241,12
	Stochastic Tournament	PMX	5639,42	305,56
		CX	5714,77	306,47
mono GA $\alpha = 10$	Tournament	PMX	7979,20	15,64
		CX	8370,05	17,33
	Roulette Wheel	PMX	7565,77	14,09
		CX	7890,01	17,10
	Stochastic Tournament	PMX	7787,26	68,69
		CX	7894,75	61,47
mono GA $\alpha = 20$	Tournament	PMX	8947,02	7,86
		CX	9150,92	9,29
	Roulette Wheel	PMX	8566,14	6,90
		CX	8831,87	8,45
	Stochastic Tournament	PMX	8841,21	42,82
		CX	8930,63	34,93
mono GA $\alpha = 30$	Tournament	PMX	9434,88	7,36
		CX	9519,38	7,40
	Roulette Wheel	PMX	9097,59	6,01
		CX	9174,33	6,92
	Stochastic Tournament	PMX	9221,53	36,20
		CX	9255,65	28,81
multi GA	Tournament	PMX	6910,88	146,06
		CX	7180,56	145,51
	Roulette Wheel	PMX	6887,11	157,53
		CX	7193,64	168,11
	Stochastic Tournament	PMX	6507,32	115,69
		CX	6807,95	139,64

Table 11 lists the comparative results for Problem 4. The best combination for all methods is Roulette Wheel with PMX. Despite having generated more balanced solutions between standard deviation and total distance, the multi-objective GA did not produce a good distribution of them, as illustrated by Figure 79.

Table 11 – Problem 04 - 8 salesmen and 50 cities

Method	Selection	Crossover	$\bar{\Delta}$	$\bar{\sigma}$
mono GA $\alpha = 1$	Tournament	PMX	10872,37	298,32
		CX	11374,33	295,15
	Roulette Wheel	PMX	10063,48	318,67
		CX	10837,75	314,75
	Stochastic Tournament	PMX	11446,54	389,58
		CX	11053,64	399,53
mono GA $\alpha = 10$	Tournament	PMX	12844,32	54,04
		CX	13076,44	57,20
	Roulette Wheel	PMX	12573,19	45,18
		CX	12824,12	53,64
	Stochastic Tournament	PMX	13766,45	143,55
		CX	13636,32	128,82
mono GA $\alpha = 20$	Tournament	PMX	15219,06	13,29
		CX	15210,54	15,80
	Roulette Wheel	PMX	14397,30	15,82
		CX	14741,78	17,66
	Stochastic Tournament	PMX	15071,89	101,77
		CX	15000,90	93,27
mono GA $\alpha = 30$	Tournament	PMX	15902,89	11,69
		CX	15976,41	14,53
	Roulette Wheel	PMX	15612,05	12,09
		CX	15761,27	12,99
	Stochastic Tournament	PMX	15945,39	86,70
		CX	15873,15	75,25
multi GA	Tournament	PMX	12428,03	149,42
		CX	12861,60	153,17
	Roulette Wheel	PMX	12477,55	141,79
		CX	12890,38	154,09
	Stochastic Tournament	PMX	12563,90	140,38
		CX	12795,71	149,92

The following list shows the hypervolumes given by the best combination of each method:

- ❑ mono GA $\alpha = 10$ (RouletteWheel&PMX): hypervolume = 838472
- ❑ multi GA (RouletteWheel&PMX): hypervolume = 591448
- ❑ mono GA $\alpha = 20$ (RouletteWheel&PMX): hypervolume = 371488
- ❑ mono GA $\alpha = 1$ (RouletteWheel&PMX): hypervolume = 5556
- ❑ mono GA $\alpha = 30$ (Roulette Wheel&PMX): hypervolume = 616

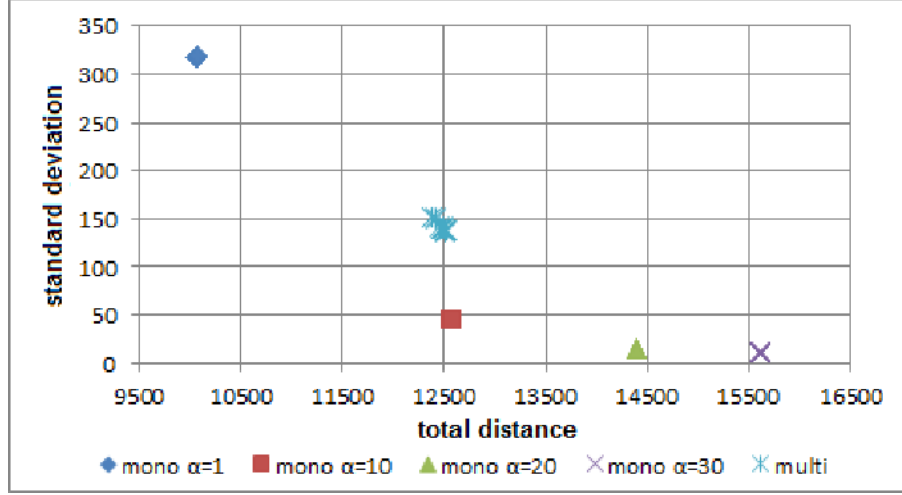


Figure 79 – Problem 04: solution sets given by the best combination of each method.

In Table 12, which represents the results of Problem 5, the proposed GAs still reach the best solutions using Roulette Wheel with PMX. According to Figure 80, the solutions found by the mono-objective GA with $\alpha = 10$ clearly dominate the solutions of the multi-objective GA.

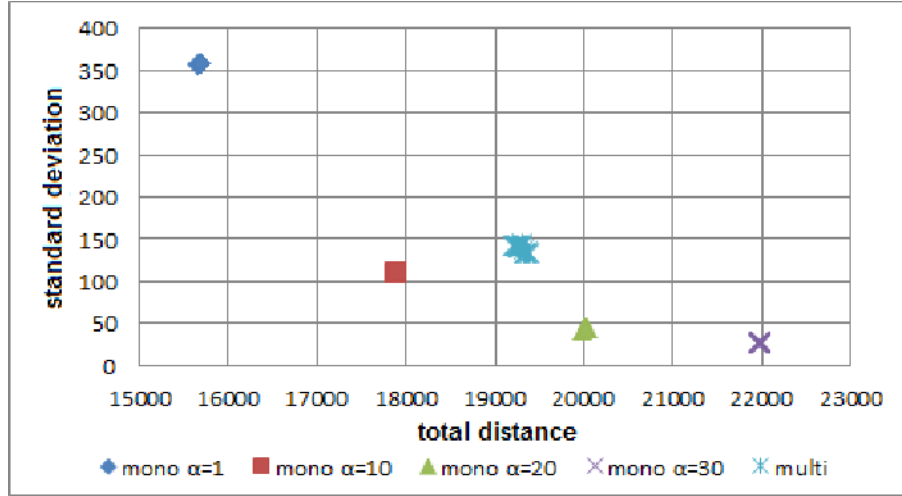


Figure 80 – Problem 05: solution sets given by the best combination of each method.

But, the hypervolumes of multi-objective GA and mono-objective GA with $\alpha = 20$ are almost the same, as listed below:

- mono GA $\alpha = 10$ (RouletteWheel&PMX): hypervolume = 1029250
- mono GA $\alpha = 20$ (RouletteWheel&PMX): hypervolume = 636536
- multi GA (RouletteWheel&PMX0: hypervolume = 629379

□ mono GA $\alpha = 1$ (RouletteWheel&PMX): hypervolume = 12692

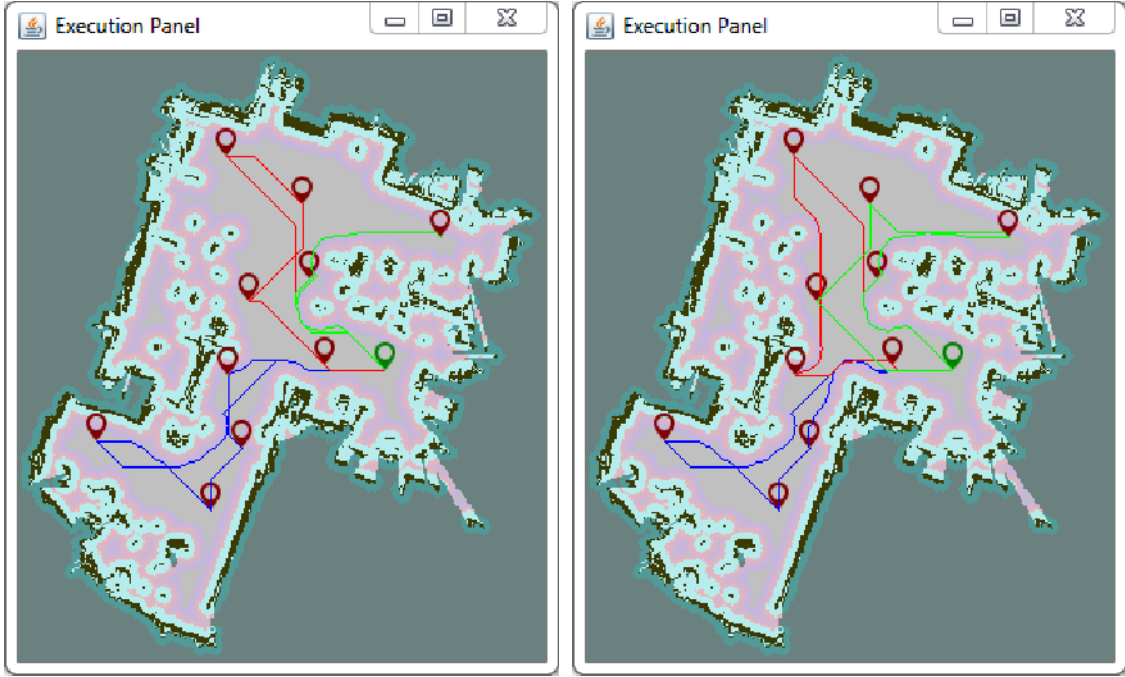
□ mono GA $\alpha = 30$ (RouletteWheel&PMX): hypervolume = 9296

Table 12 – Problem 05 - 13 salesmen and 80 cities

Method	Selection	Crossover	$\bar{\Delta}$	$\bar{\sigma}$
mono GA $\alpha = 1$	Tournament	PMX	16506,84	326,61
		CX	17155,08	309,63
	Roulette Wheel	PMX	15682,80	359,16
		CX	16831,37	354,38
	Stochastic Tournament	PMX	18588,49	471,21
		CX	17688,35	446,17
mono GA $\alpha = 10$	Tournament	PMX	18014,19	121,32
		CX	18388,82	138,23
	Roulette Wheel	PMX	17898,60	110,43
		CX	18596,47	122,30
	Stochastic Tournament	PMX	20718,78	230,90
		CX	20190,39	214,05
mono GA $\alpha = 20$	Tournament	PMX	20254,93	42,63
		CX	20335,32	51,53
	Roulette Wheel	PMX	20018,85	43,33
		CX	20589,01	57,29
	Stochastic Tournament	PMX	22186,17	178,19
		CX	22219,49	169,80
mono GA $\alpha = 30$	Tournament	PMX	22228,94	24,65
		CX	21978,21	34,36
	Roulette Wheel	PMX	21980,78	27,07
		CX	22269,02	38,19
	Stochastic Tournament	PMX	23181,18	152,17
		CX	23304,73	145,39
multi GA	Tournament	PMX	19279,15	140,58
		CX	20015,39	163,50
	Roulette Wheel	PMX	19291,86	139,63
		CX	20010,16	178,54
	Stochastic Tournament	PMX	20099,01	174,95
		CX	19942,37	175,64

The next experiment was performed on our Route Scheduling system. The parameters were almost the same: population size = 160, number of generations = 200, crossover rate = 60%, and mutation rate = 30%. The selection method was Roulette Wheel for both GAs, and the mono-objective GA was executed with $\alpha = 1$.

Figure 81 shows the execution of both GAs on the map of the Laboratory. In these trials, 10 locations and 3 salesmen were used.



(a) mono-GA.

(b) multi-GA.

Figure 81 – Route Scheduling System: Laboratory.

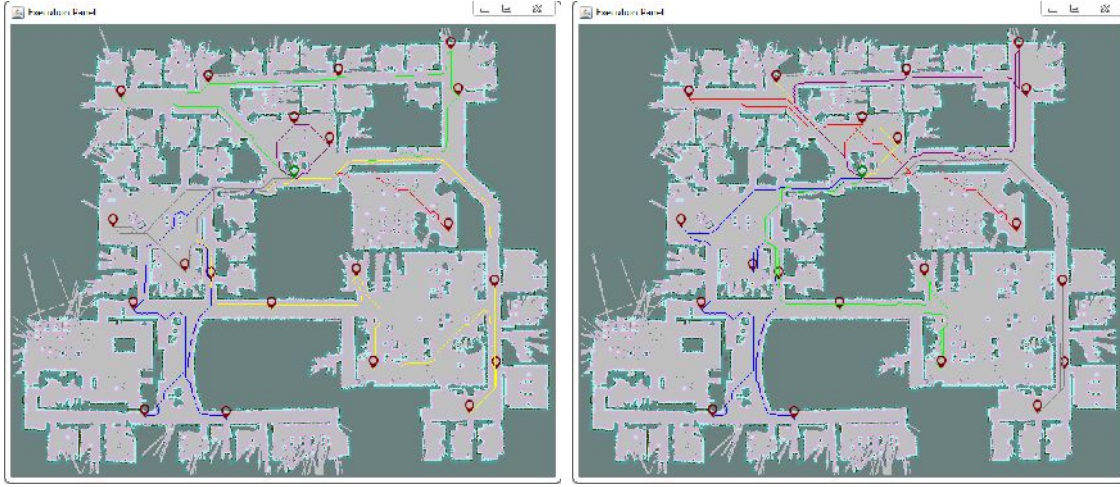
Table 13 presents the results of this test. The table shows the distance traveled by each robot (salesman), the total distance, and the standard deviation; considering both approaches. These distances are given in *pixels*.

Usually, using the transformation function of Turblebot, a *pixel* in an occupancy grid map corresponds to a cell of 5 cm^2 in the real world.

Table 13 – Route Scheduling System: Laboratory.

Method	Robot 1	Robot 2	Robot 3	Δ	σ
mono GA	417	324	332	1073	51.54
multi GA	398	368	381	1147	15.04

Figure 82 illustrates the route scheduling using the map of Willow Garage Inc. This time, 20 locations and 6 salesmen were used.



(a) mono-GA.

(b) multi-GA.

Figure 82 – Route Scheduling System: Willow Garage Inc.



Table 14 shows the results of Willow Garage Inc., which is much bigger than our Laboratory.

Table 14 – Route Scheduling System: Willow Garage Inc.

Method	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5	Robot 6	Δ	σ
mono GA	987	398	959	1343	519	167	4373	439.54
multi GA	1133	899	1083	331	998	778	5256	290.63

As we can see, for both scenarios, the mono-GA minimizes the total distance more than the multi-GA, on the other hand, their routes are more imbalanced.

Imagine that the battery of a robot can be sufficient to travel for a distance of 1200 (in pixel). Considering the schedule generated by mono-GA for Willow Garage Inc., Robot 4 could not be able to finish its route without recharging.

Although we are using A* to compute paths, it is only applied to estimate the routes for the coordination process. During the execution of the visits, the robots can use other path finder algorithms like Dijkstra, or dynamic algorithms, such as D*, that adapts to changes on the map in real-time.

5.5 Summary

In this chapter, we introduced the problem of scheduling routes for a team of agents, which can be seen as an instance of the multiple Traveling Salesmen Problem (mTSP).

There are several applications that could be modeled through the representation of mTSP, including the coordination of robots that need to visit specific locations in an environment to perform some task. This problem is classified as NP-hard, thus approximation and heuristic methods, such as Genetic Algorithms (GAs), are widely used as they can find good solutions very quickly.

The mTSP is an extension of the well-known Traveling Salesman Problem (TSP), in which more than one salesman can be used in the solution. Although its main objective is to minimize the total distance traveled by all salesmen, an imbalance between the distance traveled by each of them might occur, depending on the distribution of the cities.

Therefore, comparative experiments were made with GAs, using different combinations of selection and crossover, to solve the mTSP. The proposed GAs consider two objectives to be minimized, the total distance and the standard deviation of the distance traveled by the salesmen.

We developed two types of GAs, a multi-objective GA which attempts to optimize both objectives simultaneously, and a mono-objective GA whose fitness function is composed by both objectives along with a parameter that can be used to adjust their relevance. We also built a centralized planner system to schedule routes for a team of robots using the mTSP framework. In this case, cities are locations to be visited in the environment and salesmen are robots. The system uses the proposed GAs to generate routes for the robots on inflated indoor laser maps. In order to provide feasible routes, the system computes paths for all possible pairs of locations and stores them into a matrix so they can be used in the scheduling.

Both GAs produced good results during the experiments. Whereas the combination of Roulette Wheel with PMX was predominant for the mono-objective GA, the multi-objective GA found the best solutions using different combinations for each problem.

Although the nature of the multi-objective GA is to work with the optimization of more than one objective at a time, as addressed in this chapter, the mono-objective GA also shows competitive results. Even generating more diversified solutions in the Pareto-optimal set, for some instances the solutions of the multi-objective GA were dominated by the solutions produced by the mono-objective GA.

However, in order to produce good results, the mono-objective GA should be parameterized differently for each problem. Furthermore, the multi-objective GA tends to generate more balanced solutions to all problem instances.

Conclusion and Future Work

This thesis introduced several algorithms for *localization*, *navigation*, and *coordination* for autonomous mobile robots. Our objective is to enable robots to assist humans in their everyday lives. Therefore, we focused on the problem where a team of robots needs to perform tasks in specific locations of an indoor human environment.

We started by addressing the problem of robot *localization* in indoor human environments. Some localization techniques are not suitable for this type of environment. For instance, GPS are not applied as signals cannot come across most of buildings, StarGazer systems need marks all over the environment, depth cameras and laser range finders approaches are affected by moving and movable objects that change the map frequently, and WiFi techniques have a minimum error greater than that of other approaches. Therefore, we introduced a novel approach that combines the benefits of two localization techniques, the Kinect localization and the WiFi localization. The WiFi localization is responsible for tracking the global location of the robot in the environment, and the Kinect localization estimates the robot's position locally. First, we map the environment using a Kinect device by the means of the GMapping algorithm. Then, we introduced a WiFi mapping based on the concept of irregular grids. During the navigation, the robot use these maps to localize itself in the environment. Two Particle Filters (PFs) are used separately, one for each device. A PF, called Adaptive Monte Carlo Localization (AMCL), estimates the robot's position based on Kinect observations, and our WiFi localization algorithm uses a PF along with different interpolation techniques to estimate the location of the robot from WiFi signal strengths. Our approach also enables the robot to automatically recover its position in case it is lost.

We separated the robot *navigation* problem into two topics, Obstacle Avoidance and Path Planning. To create a reactive controller for the Obstacle Avoidance task, we introduced a Hybrid Intelligent System (HIS) based on Fuzzy Logic (FL) and Artificial Neural Networks (ANN) to ensure a collision-free navigation for mobile robots. In this system, infrared sensors positioned around the robot send data to fuzzy rules, generating linguistic/symbolic information about possible obstacles in the environment, for example,

“obstacle on the left” or “path is free”. This symbolic information is used by an ANN to determine which movement the robot must do to navigate without colliding, such as “turn right” or “move forward”. For the Path Planning task, we introduce an adaptation to the D*Lite algorithm, which is able to dynamically replan the robot’s path efficiently in case it is blocked, and also to ask for human assistance to remove objects on the way if the cost of the new path is high.

Finally, we tackled the problem of robots *coordination*. We are interested in the case where a team of robots needs to visit some specific locations in an indoor environment. Each location must be visited only once by one of the robots, while the total distance traveled by them is reduced. To solve this problem, it is necessary to find routes for the robots that cover all locations. Actually, this problem is an instance of the multiple Traveling Salesmen Problem (mTSP). The regular TSP belongs to the class of NP-complete problems, because the time increases exponentially for every increase in the number of locations to be visited. But mTSP is even more complex, as it requires the evaluation and sorting of nodes in the route of each robot, which makes it NP-hard. Thus, approximation and heuristic methods, like Genetic Algorithms (GAs), are often used to solve this kind of problem. However, the distances traveled individually by the robots can become imbalanced when the main objective is to reduce the total distance. Therefore, we introduced a multi-robot route scheduling system for indoor environments that uses multi-objective GAs able to minimize the total distance and also to balance the routes for the robots.

Some directions for future work on *localization*, *navigation* and *coordination* are listed below.

Regarding *localization*, during the tests we noticed a positioning error of 50cm-60cm, which is great considering the cost of our devices (Microsoft Kinect v1 and USB WiFi dongle). This result could be even better depending on the environment. For example, the indoor positioning within small and narrow areas, such as corridors and rooms, has a better precision, as walls and corners yield more featured observations to the Kinect localization, different from large indoor open areas, like a lobby or a laboratory. We have not found any related work using low cost devices in indoor open areas like we did. Either way, more experiments can be carried out in different types of environment in order to show more about the contributions of our work.

We believe that other devices could be tested, such as LIDAR laser scanner or Kinect v2, in order to reduce the error. Furthermore, new methods for WiFi localization using Angle of Arrival and Time of Arrival should be evaluated, as they seem to perform better than the traditional Received Signal Strength approach.

We had some problems with WiFi localization regarding the network management. Usually, the settings of some APs are switched for security and performance purposes, which changes the WiFi map of the environment and increases the error. Also, furniture that are moved frequently in indoor areas might change the laser map used by the Kinect

localization. Thus, researches on how to fix maps will be extremely important to the indoor localization field.

In our work, the *navigation* problem was divided into Obstacle Avoidance and Path Planning. We presented a Hybrid Intelligent System as a reactive controller to detect obstacles and avoid collisions, and a Dynamic Path Planner with Human Assistance to plan the robot's path towards the target location on the map. However, we did not implement the integration between these two approaches, so they can work together in a single navigation system. As a future work, this integration could be done by creating an algorithm that prioritizes the execution of the obstacle avoidance controller in case an object is coming towards the robot. Once the robot is safe, the algorithm can call again the Path Planner to resume its execution or re-plan the path if the robot ended up in a different location.

The architecture of our Hybrid Intelligent System enables it to use different algorithms for obstacle detection and motion selection. Therefore, other Machine Learning techniques could be tested. Furthermore, our experiments on obstacle avoidance were carried out using only an E-puck robot on simulation. It would be interesting to create a ring of InfraRed sensors on a physical robot, like a Turtlebot, and test our approach in real environments.

Although our Dynamic Path Planner with Human Assistance was developed base on D*Lite algorithm, this concept can be applied to other search algorithms. For instance, Rapidly-Exploring Random Trees (RRT) (LAVALLE; KUFFNER; JR., 2000) has been widely applied to robot navigation, and instead of fixing the current path as D*Lite does, some RRT variants are able to plan new paths from scratch very fast.

Another aspect about the Dynamic Path Planner with Human Assistance that was not explored in this work is the Human Robot Interaction (HRI). HRI techniques could be applied in this case to make the robot more friendly and easy to understand when it needs human assistance.

In *coordination*, our centralized route scheduling system enables the generation feasible routes to be executed by robots in indoor environments, while it minimizes the total distance and balances the routes. We have not found any work in the literature performing such task in robotics considering these two objectives. The bottleneck of this system is the computation of the path matrix, as we use A* to compute the optimal path between all pairs of locations. Although the path matrix can be saved and used later for other executions on the same map, this process still takes too much time. For instance, the system required almost 4 hours to compute the path matrix for the experiment illustrated by Figure 82, using a HP laptop (CPU: Intel i5, MEM:8GB, OS: Windows7-64Bits). Thus, other search algorithms could be tested to replace A* in order to reduce the time.

As we have only one physical robot (Turtlebot v2) to perform our experiments, most of the tests on coordination were carried out sending the routes generated by our system

to virtual robots on Gazebo simulator. To do that, we had to adapt some virtual models of Turtlebot, so they can work in parallel on Gazebo. It would be important to perform experiments on a team of real robots, and measure their batteries to ensure that the system is providing balanced routes to all robots.

Furthermore, there are some other interesting aspects that should be investigated. In particular, it is necessary to investigate mTSP with multiple depots (charging stations). It is also necessary the development of on-line route scheduling algorithms to fix the routes in case the set of locations has changed during the execution.

In addition to these directions for futures work, another important issue regarding this thesis is to test the integration of all proposed algorithms. For example, in case the robot is kidnapped while it is navigating to some location, how the localization algorithm can call the path planner algorithm to fix the path after it recovers the robot's position.

The problems of localization, navigation and coordination are difficult to solve, and put their solutions working together in a single platform is not an easy task. Besides the technical challenge, it will be necessary many real robots to test all algorithms together, as some natural aspects are not trivial to be implemented on simulation, like WiFi and Lighting noises. Even if it is possible, they may not reflect the actual result in real environments.

Bibliography

- ACKERMAN, E. **iRobot Brings Visual Mapping and Navigation to the Roomba 980**. 2015. [Online; posted 16-September-2015]. Disponível em: <<http://spectrum.ieee.org/automaton/robotics/home-robots/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980>>.
- ALGABRI, M.; MATHKOUR, H.; RAMDANE, H. Mobile robot navigation and obstacle-avoidance using anfis in unknown environment. **International Journal of Computer Applications**, v. 91, n. 14, p. 36–41, April 2014.
- ALI, A. I.; KENNINGTON, J. L. The asymmetric m-travelling salesmen problem: A duality based branch-and-bound algorithm. **Discrete Applied Mathematics**, Elsevier, v. 13, n. 2-3, p. 259–276, 1986.
- ALVES, R. M. F.; LOPES, C. R. Using genetic algorithms to minimize the distance and balance the routes for the multiple traveling salesman problem. In: **2015 IEEE Congress on Evolutionary Computation (CEC)**. Institute of Electrical and Electronics Engineers (IEEE), 2015. Disponível em: <<http://dx.doi.org/10.1109/CEC.2015.7257285>>.
- _____. Obstacle avoidance for mobile robots: a hybrid intelligent system based on fuzzy logic and artificial neural network. In: **IEEE World Congress on Computational Intelligence (WCCI)**. [S.l.: s.n.], 2016.
- ALVES, R. M. F.; LOPES, C. R.; VELOSO, M. Checkpoint coverage with wifi localization for indoor service robots. In: **Workshop on Autonomous Mobile Service Robots: 25th International Joint Conference on Artificial Intelligence (IJCAI)**. [S.l.: s.n.], 2016.
- ANGEL, R. D. et al. Computer-assisted school bus scheduling. **Management Science**, v. 18, n. 6, p. B-279–B-288, 1972.
- ARYA, V.; GOYAL, A.; JAISWAL, V. An optimal solution to multiple traveling salesperson problem using modified genetic algorithm. **International Journal of Application or Innovation in Engineering & Management (IJAIEEM)**, 2014.
- BALAKIRSKY, S. et al. Results and performance metrics from robocup rescue 2007. **Journal of Field Robotics**, 24(11-12):943-967, Citeseer, 2007.

BASU, A.; ELNAGAR, A.; AL-HAJJ, R. Efficient coordinated motion. **Mathematical and computer modelling**, Elsevier, v. 31, n. 2-3, p. 39–53, 2000.

BEKTAS, T. The multiple traveling salesman problem: an overview of formulations and solution procedures. **Omega**, Elsevier, v. 34, n. 3, p. 209–219, 2006.

BELLMORE, M.; HONG, S. Transformation of multisalesman problem to the standard traveling salesman problem. **Journal of the ACM (JACM)**, ACM, v. 21, n. 3, p. 500–504, 1974.

BI, Z.; YIMIN, Y.; YISAN, X. Mobile robot navigation in unknown dynamic environment based on ant colony algorithm. In: IEEE. **Intelligent Systems, 2009. GCIS'09. WRI Global Congress on**. [S.l.], 2009. v. 3, p. 98–102.

BISWAS, J.; Veloso, M. Depth camera based indoor mobile robot localization and navigation. In: **Proceedings of IEEE International Conference on Robotics and Automation**. [S.l.]: IEEE, 2012. p. 1697–1702.

_____. Episodic non-markov localization: Reasoning about short-term and long-term features. In: **International Conference on Robotics and Automation (ICRA) 2014**. [S.l.: s.n.], 2014.

BISWAS, J.; VELOSO, M. Wifi localization and navigation for autonomous indoor mobile robots. In: **2010 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2010. p. 4379–4384. ISSN 1050-4729.

_____. Depth camera based indoor mobile robot localization and navigation. In: IEEE. **Robotics and Automation (ICRA), 2012 IEEE International Conference on**. [S.l.], 2012. p. 1697–1702.

_____. Multi-sensor mobile robot localization for diverse environments. In: SPRINGER. **Robot Soccer World Cup**. [S.l.], 2013. p. 468–479.

BORENSTEIN, J.; KOREN, Y. The vector field histogram-fast obstacle avoidance for mobile robots. **IEEE transactions on robotics and automation**, IEEE, v. 7, n. 3, p. 278–288, 1991.

BRENA, R. F. et al. Evolution of indoor positioning technologies: A survey. **Journal of Sensors**, Hindawi Publishing Corporation, v. 2017, 2017.

BRUCE, J. et al. Multi-robot team response to a multi-robot opponent team. In: IEEE. **Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on**. [S.l.], 2003. v. 2, p. 2281–2286.

BRUMITT, B.; STENTZ, A. Dynamic mission planning for multiple mobile robots. In: **Intelligent Unmanned Ground Vehicles**. [S.l.]: Springer, 1997. p. 221–234.

BRUMITT, B. L.; STENTZ, A. Grammps: A generalized mission planner for multiple mobile robots in unstructured environments. In: IEEE. **Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on**. [S.l.], 1998. v. 2, p. 1564–1571.

BUHMANN, J. et al. The mobile robot rhino. In: _____. **Neural Networks: Artificial Intelligence and Industrial Applications: Proceedings of the Third Annual SNN Symposium on Neural Networks, Nijmegen, The Netherlands, 14–15 September 1995**. London: Springer London, 1995. p. 129–139. ISBN 978-1-4471-3087-1. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-3087-1_26>.

_____. The mobile robot rhino. **Ai Magazine**, v. 16, n. 2, p. 31, 1995.

CALLOUD, P. et al. Indoor automation with many mobile robots. In: IEEE. **Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on**. [S.l.], 1990. p. 67–72.

CALVO, R. W.; CORDONE, R. A heuristic approach to the overnight security service problem. **Computers & Operations Research**, Elsevier, v. 30, n. 9, p. 1269–1287, 2003.

CANEDO-RODRÍGUEZ, A. et al. Particle filter robot localisation through robust fusion of laser, wifi, compass, and a network of external cameras. **Inf. Fusion**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 27, n. C, p. 170–188, jan. 2016. ISSN 1566-2535. Disponível em: <<http://dx.doi.org/10.1016/j.inffus.2015.03.006>>.

CARTER, A. E.; RAGSDALE, C. T. Scheduling pre-printed newspaper advertising inserts using genetic algorithms. **Omega**, Elsevier, v. 30, n. 6, p. 415–421, 2002.

CASTILLO, O.; TRUJILLO, L.; MELIN, P. Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots. **Soft Computing-A Fusion of Foundations, Methodologies and Applications**, Springer, v. 11, n. 3, p. 269–279, 2007.

CHAITIN, L. J. et al. **Research and Applications - Artificial Intelligence**. [S.l.], 1970.

CHENG, H.; CHEN, H.; LIU, Y. Topological indoor localization and navigation for autonomous mobile robot. **IEEE Transactions on Automation Science and Engineering**, IEEE, v. 12, n. 2, p. 729–738, 2015.

CHI, K.-H.; LEE, M.-F. R. Obstacle avoidance in mobile robot using neural network. In: **International Conference on Consumer Electronics, Communications and Networks**. [S.l.: s.n.], 2011.

CHIA, S.-H. et al. Ant colony system based mobile robot path planning. In: IEEE. **Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on**. [S.l.], 2010. p. 210–213.

CHRISTOFIDES, N.; EILON, S. An algorithm for the vehicle-dispatching problem. **OR**, JSTOR, p. 309–318, 1969.

COCKBURN, D.; KOBTI, Z. Memory-bounded d* lite. In: WILSON, D.; LANE, H. C. (Ed.). **FLAIRS Conference**. AAAI Press, 2008. p. 376–380. ISBN 978-1-57735-365-2. Disponível em: <<http://dblp.uni-trier.de/db/conf/flairs/flairs2008.html#CockburnK08>>.

COLTIN, B.; VELOSO, M. Online pickup and delivery planning with transfers for mobile robots. In: IEEE. **Robotics and Automation (ICRA), 2014 IEEE International Conference on**. [S.l.], 2014. p. 5786–5791.

DAVENDRA, D. (Ed.). **Traveling Salesman Problem, Theory and Applications**. InTech, 2010. Disponível em: <<http://dx.doi.org/10.5772/547>>.

DB, F. A parallel processing approach to a multiple traveling salesman problem using evolutionary programming. In: **Proceedings of the fourth annual symposium on parallel processing**. Fullerton, CA.: [s.n.], 1990. p. 318–26.

DELLAERT, F. et al. Monte carlo localization for mobile robots. In: IEEE. **Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on**. [S.l.], 1999. v. 2, p. 1322–1328.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **NUMERISCHE MATHEMATIK**, v. 1, n. 1, p. 269–271, 1959.

DOUC, R. Comparison of resampling schemes for particle filtering. In: **In 4th International Symposium on Image and Signal Processing and Analysis (ISPA**. [S.l.: s.n.], 2005. p. 64–69.

DURRANT-WHYTE, H. F. Sensor models and multisensor integration. **The international journal of robotics research**, Sage Publications Sage CA: Thousand Oaks, CA, v. 7, n. 6, p. 97–113, 1988.

DUTTA, S. Obstacle avoidance of mobile robot using pso-based neuro fuzzy technique. **International Journal of Computer Science and Engineering**, v. 2, n. 2, p. 301–304, 2010.

ENGELSON, S. P.; MCDERMOTT, D. V. Error correction in mobile robot map learning. In: IEEE. **Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on**. [S.l.], 1992. p. 2555–2560.

FAIGL, J.; KULICH, M.; PŘEUCIL, L. Goal assignment using distance cost in multi-robot exploration. In: IEEE. **Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on**. [S.l.], 2012. p. 3741–3746.

FARID, Z.; NORDIN, R.; ISMAIL, M. Recent advances in wireless indoor localization techniques and system. **Journal of Computer Networks and Communications**, Hindawi Publishing Corporation, v. 2013, 2013.

FERGUSON, D.; LIKHACHEV, M.; STENTZ, A. A guide to heuristic-based path planning. In: **Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)**. [S.l.: s.n.], 2005. p. 9–18.

FERGUSON, D.; STENTZ, A. The delayed d* algorithm for efficient path replanning. In: IEEE. **Proceedings of the 2005 IEEE international conference on robotics and automation**. [S.l.], 2005. p. 2045–2050.

_____. Field d*: An interpolation-based path planner and replanner. In: **Robotics research**. [S.l.]: Springer, 2007. p. 239–253.

FOX, D. Kld-sampling: Adaptive particle filters. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2001. p. 713–720.

FOX, D.; BURGARD, W.; THRUN, S. The dynamic window approach to collision avoidance. **Robotics Automation Magazine, IEEE**, v. 4, n. 1, p. 23–33, mar. 1997. ISSN 1070-9932.

GANGANATH, N.; CHENG, C. T.; TSE, C. K. An aco-based off-line path planner for nonholonomic mobile robots. In: **2014 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2014. p. 1038–1041. ISSN 0271-4302.

GARCIA-NAJERA, A.; GUTIERREZ-ANDRADE, M. A. An evolutionary approach to the multi-objective pickup and delivery problem with time windows. In: **2013 IEEE Congress on Evolutionary Computation**. IEEE, 2013. Disponível em: <<http://dx.doi.org/10.1109/CEC.2013.6557676>>.

GAVISH, B.; SRIKANTH, K. An optimal solution method for large-scale multiple traveling salesmen problems. **Operations Research, INFORMS**, v. 34, n. 5, p. 698–717, 1986.

GAVRILOV, A. V.; LEE, S. An architecture of hybrid neural network based navigation system for mobile robot. In: **Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)**. [S.l.: s.n.], 2007. p. 587–590. ISSN 2164-7143.

GERKEY, B. P.; MATARIĆ, M. J. Murdoch: Publish/subscribe task allocation for heterogeneous agents. In: **ACM. Proceedings of the fourth international conference on Autonomous agents**. [S.l.], 2000. p. 203–204.

GILBERT, K. C.; HOFSTRA, R. B. A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem. **Decision Sciences, Wiley Online Library**, v. 23, n. 1, p. 250–259, 1992.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

GORENSTEIN, S. Printing press scheduling for multi-edition periodicals. **Management Science, INFORMS**, v. 16, n. 6, p. B–373, 1970.

GRISSETTI, G.; STACHNISS, C.; BURGARD, W. Improved techniques for grid mapping with rao-blackwellized particle filters. **IEEE transactions on Robotics, IEEE**, v. 23, n. 1, p. 34–46, 2007.

GROMICHO, J.; PAIXÃO, J.; BRONCO, I. Exact solution of multiple traveling salesman problems. In: **Combinatorial Optimization**. [S.l.]: Springer, 1992. p. 291–292.

GUOXING, Y. Transformation of multidepot multisalesmen problem to the standard travelling salesman problem. **European Journal of Operational Research, Elsevier**, v. 81, n. 3, p. 557–560, 1995.

GUTMANN, J.-S.; SCHLEGEL, C. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In: IEEE. **Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on**. [S.l.], 1996. p. 61–67.

HART, N. J. N. P. E.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems, Science, and Cybernetics**, SSC-4, n. 2, p. 100–107, 1968.

HEBERT, M.; MACLACHLAN, R.; CHANG, P. Experiments with driving modes for urban robots. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Photonics East'99**. [S.l.], 1999. p. 140–149.

HEGER, F. W. **Assembly planning in constrained environments: Building structures with multiple mobile robots**. [S.l.]: Carnegie Mellon University, 2010.

HENRY, P. et al. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In: **12th International Symposium on Experimental Robotics (ISER**. [S.l.: s.n.], 2010.

HONG, S.; PADBERG, M. W. A note on the symmetric multiple traveling salesman problem with fixed charges. **Operations Research**, INFORMS, v. 25, n. 5, p. 871–874, 1977.

HUANG, H.-C. Fpga-based parallel metaheuristic pso algorithm and its application to global path planning for autonomous robot navigation. **Journal of Intelligent & Robotic Systems**, Springer Science & Business Media, v. 76, n. 3-4, p. 475, 2014.

HUSSAIN, M. et al. Multi-robot scheduling using evolutionary algorithms. In: IEEE. **Automation Congress, 2002 Proceedings of the 5th Biannual World**. [S.l.], 2002. v. 13, p. 233–238.

JALOBEANU, M. et al. Reliable kinect-based navigation in large indoor environments. In: IEEE. **2015 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.], 2015. p. 495–502.

JEFFRIL, M. A.; SARIFF, N. The integration of fuzzy logic and artificial neural network methods for mobile robot obstacle avoidance in a static environment. In: **2013 IEEE 3rd International Conference on System Engineering and Technology**. IEEE, 2013. Disponível em: <<http://dx.doi.org/10.1109/ICSEngT.2013.6650193>>.

JONKER, R.; VOLGENANT, T. An improved transformation of the symmetric multiple traveling salesman problem. **Operations Research**, INFORMS, v. 36, n. 1, p. 163–167, 1988.

KABURLASOS, V. G.; ATHANASIADIS, I. N.; MITKAS, P. A. Fuzzy lattice reasoning (flr) classifier and its application for ambient ozone estimation. **International Journal of Approximate Reasoning**, v. 45, n. 1, p. 152 – 188, 2007. ISSN 0888-613X. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0888613X06000909>>.

KARA, I.; BEKTAS, T. Integer linear programming formulations of multiple salesman problems and its variations. **European Journal of Operational Research**, Elsevier, v. 174, n. 3, p. 1449–1458, 2006.

- KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. **The international journal of robotics research**, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 1, p. 90–98, 1986.
- KHATIB, O. et al. Force strategies for cooperative tasks in multiple mobile manipulation systems. In: MIT PRESS. **ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-**. [S.l.], 1996. v. 7, p. 333–342.
- KIM, K. H.; PARK, Y.-M. A crane scheduling method for port container terminals. **European Journal of operational research**, Elsevier, v. 156, n. 3, p. 752–768, 2004.
- KIRALY, A.; ABONYI, J. A novel approach to solve multiple traveling salesmen problem by genetic algorithm. In: **Computational Intelligence in Engineering**. [S.l.]: Springer, 2010. p. 141–151.
- KLIR, G.; YUAN, B. **Fuzzy sets and fuzzy logic**. [S.l.]: Prentice Hall New Jersey, 1995. v. 4.
- KOENIG, S.; LIKHACHEV, M. Incremental a*. In: **NIPS**. [S.l.: s.n.], 2001. p. 1539–1546.
- _____. D*lite. In: **Eighteenth National Conference on Artificial Intelligence**. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002. p. 476–483. ISBN 0-262-51129-0. Disponível em: <<http://dl.acm.org/citation.cfm?id=777092.777167>>.
- KOENIG, S.; SIMMONS, R. Xavier: A robot navigation architecture based on partially observable markov decision process models. **Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems**, Citeseer, p. 91–122, 1998.
- KORTENKAMP, D.; WEYMOUTH, T. Topological mapping for mobile robots using a combination of sonar and vision sensing. In: **AAAI**. [S.l.: s.n.], 1994. v. 94, p. 979–984.
- KUIPERS, B.; BYUN, Y.-T. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. **Robotics and autonomous systems**, Elsevier, v. 8, n. 1, p. 47–63, 1991.
- KULKARNI, R.; BHAVE, P. R. Integer programming formulations of vehicle routing problems. **European Journal of Operational Research**, Elsevier, v. 20, n. 1, p. 58–67, 1985.
- KUNCHEV, V. et al. Path planning and obstacle avoidance for autonomous mobile robots: A review. In: _____. **Knowledge-Based Intelligent Information and Engineering Systems: 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006. Proceedings, Part II**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 537–544. ISBN 978-3-540-46539-3. Disponível em: <https://doi.org/10.1007/11893004_70>.
- LAPORTE, G.; NOBERT, Y. A cutting planes algorithm for the m-salesmen problem. **Journal of the Operational Research Society**, JSTOR, p. 1017–1023, 1980.
- LAPORTE, G.; NOBERT, Y.; TAILLEFER, S. Solving a family of multi-depot vehicle routing and location-routing problems. **Transportation science**, INFORMS, v. 22, n. 3, p. 161–172, 1988.

LAVALLE, S. M.; KUFFNER, J. J.; JR. Rapidly-exploring random trees: Progress and prospects. In: **Algorithmic and Computational Robotics: New Directions**. [S.l.: s.n.], 2000. p. 293–308.

LEE, S.; SONG, J.-B. Mobile robot localization using infrared light reflecting landmarks. In: **IEEE: Control, Automation and Systems**. [S.l.: s.n.], 2007. p. 674–677.

LENSTRA, J. K.; KAN, A. R. Some simple applications of the travelling salesman problem. **Journal of the Operational Research Society**, Springer, v. 26, n. 4, p. 717–733, 1975.

LEONARD, J.; DURRANT-WHYTE, H.; COX, I. J. Dynamic map building for autonomous mobile robot. In: IEEE. **Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on**. [S.l.], 1990. p. 89–96.

LI, X.; CHOI, B. jae. **Design of Obstacle Avoidance System for Mobile Robot using Fuzzy Logic Systems**. 2013.

LI, Y.; CHEN, X. Mobile robot navigation using particle swarm optimization and adaptive nn. **Advances in Natural Computation**, Springer, p. 439–439, 2005.

LI, Z.; LI, X. Genetic algorithm for task allocation and path planning of multi-robot system. **Journal of Mathematical Sciences and Applications**, Science and Education Publishing, v. 4, n. 1, p. 34–38, 2016.

LIAO, X.-L.; CHIEN, C.-H.; TING, C.-K. A genetic algorithm for the minimum latency pickup and delivery problem. In: IEEE. **Evolutionary Computation (CEC), 2014 IEEE Congress on**. [S.l.], 2014. p. 3272–3279.

LUMELSKY, V. J.; SKEWIS, T. Incorporating range sensing in the robot navigation function. **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE, v. 20, n. 5, p. 1058–1069, 1990.

LUMELSKY, V. J.; STEPANOV, A. A. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. **Autonomous robot vehicles**, New York, NY, USA: Springer-Verlag New York, Inc, p. 363–390, 1990.

MA, X.; ZHANG, Q.; LI, Y. Genetic algorithm-based multi-robot cooperative exploration. In: **IEEE: Control and Automation**. [S.l.: s.n.], 2007. p. 1018–1023.

MAHONEY, R. **Service Robotics Case Studies in Silicon Valley**. 2015. Last access: 2017-09-07. Disponível em: <<https://svrobo.org/wp-content/uploads/2015/05/Service-Robotics-Case-Studies.pdf>>.

MATTHIES, L. et al. A portable, autonomous, urban reconnaissance robot. **Robotics and Autonomous Systems**, Elsevier, v. 40, n. 2, p. 163–172, 2002.

MAUTZ, R. **Indoor positioning technologies**. Tese (Doutorado) — ETH Zurich, 2012.

MAYBECK, P. S. **Stochastic models, estimation, and control**. [S.l.]: Academic press, 1982. v. 3.

- MICHEL, O. Cyberbotics ltd. webotsTM: professional mobile robot simulation. **International Journal of Advanced Robotic Systems**, SAGE Publications Sage UK: London, England, v. 1, n. 1, p. 5, 2004.
- MILLS-TETTEY, G. A.; STENTZ, A.; DIAS, M. B. Dd* lite: Efficient incremental search with state dominance. In: MENLO PARK, CA; CAMBRIDGE, MA; LONDON; AAAI PRESS; MIT PRESS; 1999. **Proceedings of the National Conference on Artificial Intelligence**. [S.l.], 2006. v. 21, n. 2, p. 1032.
- MIROWSKI, P.; HO, T. K.; WHITING, P. Building optimal radio-frequency signal maps. In: **2014 22nd International Conference on Pattern Recognition**. [S.l.: s.n.], 2014. p. 978–983. ISSN 1051-4651.
- MODARES, A.; SOMHOM, S.; ENKAWA, T. A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. **International Transactions in Operational Research**, Wiley Online Library, v. 6, n. 6, p. 591–606, 1999.
- MORAVEC, H. P. Sensor fusion in certainty grids for mobile robots. **AI magazine**, v. 9, n. 2, p. 61, 1988.
- MOTLAGH, O. et al. Automatic navigation of mobile robots in unknown environments. **Neural Computing and Applications**, v. 24, n. 7, p. 1569–1581, Jun 2014. ISSN 1433-3058. Disponível em: <<https://doi.org/10.1007/s00521-013-1393-z>>.
- NERURKAR, E. D.; ZHOU, K. X.; ROUMELIOTIS, S. I. A hybrid estimation framework for cooperative localization under communication constraints. In: **IEEE Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on**. [S.l.], 2011. p. 502–509.
- NICHOLS, E.; MCDAID, L. J.; SIDDIQUE, N. Biologically inspired snn for robot control. **IEEE transactions on cybernetics**, v. 43, n. 1, p. 115–128, 2013.
- OKONJO-ADIGWE, C. An effective method of balancing the workload amongst salesmen. **Omega**, Elsevier, v. 16, n. 2, p. 159–163, 1988.
- ORLOFF, C. S. Routing a fleet of m vehicles to/from a central facility. **Networks**, v. 4, n. 2, p. 147–162, 1974.
- OSHBOT. **OSHbot**. 2017. Last access: 2017-09-07. Disponível em: <<http://www.lowesinnovationlabs.com/innovation-robots/>>.
- OYAMA, A. et al. Come on in, our community is wide open for robotics research! In: **RSJ**. [s.n.], 2009. Disponível em: <<http://www.rsj.or.jp/rsj2009/>>.
- PALANIAPPAN, R. et al. Autonomous rf surveying robot for indoor localization and tracking. In: **International conference on indoor positioning and indoor navigation**. [S.l.: s.n.], 2011.
- PANDEY, D. A. Mobile robot navigation and obstacle avoidance techniques: A review. v. 2, p. 1–12, 05 2017.
- PARHI, D. R. Navigation of mobile robots using a fuzzy logic controller. **Journal of Intelligent & Robotic Systems**, Springer, v. 42, n. 3, p. 253–273, 2005.

- PEI, Y.; MUTKA, M. W. Steiner traveler: Relay deployment for remote sensing in heterogeneous multi-robot exploration. In: IEEE. **Robotics and Automation (ICRA), 2012 IEEE International Conference on**. [S.l.], 2012. p. 1551–1556.
- PHILIPPSEN, R. **A Light Formulation of the E Interpolated Path Replanner**. [S.l.], 2006.
- POTVIN, J.-Y.; LAPALME, G.; ROUSSEAU, J.-M. A generalized k-opt exchange procedure for the mtsp. **INFOR: Information Systems and Operational Research**, Taylor & Francis, v. 27, n. 4, p. 474–481, 1989.
- PRADHAN, S. K.; PARHI, D. R.; PANDA, A. K. Fuzzy logic techniques for navigation of several mobile robots. **Applied soft computing**, Elsevier, v. 9, n. 1, p. 290–304, 2009.
- PRANSKY, J. The pransky interview: Dr steve cousins, ceo, savioka, entrepreneur and innovator. **Industrial Robot: An International Journal**, Emerald Group Publishing Limited, v. 43, n. 1, p. 1–5, 2016.
- PRECUP, R.-E. et al. Simulated annealing approach to fuzzy modeling of servo systems. In: IEEE. **Cybernetics (CYBCONF), 2013 IEEE International Conference on**. [S.l.], 2013. p. 267–272.
- PROROK, A.; BAHR, A.; MARTINOLI, A. Low-cost collaborative localization for large-scale multi-robot systems. In: IEEE. **Robotics and Automation (ICRA), 2012 IEEE International Conference on**. [S.l.], 2012. p. 4236–4241.
- QIAN, K. et al. Mobile robot navigation in unknown corridors using line and dense features of point clouds. In: IEEE. **Industrial Electronics Society, IECON 2015–41st Annual Conference of the IEEE**. [S.l.], 2015. p. 001831–001836.
- QUINLAN, S.; KHATIB, O. Elastic bands: Connecting path planning and control. In: IEEE. **Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on**. [S.l.], 1993. p. 802–807.
- RAO, M. A note on the multiple traveling salesmen problem. **Operations Research**, INFORMS, v. 28, n. 3-part-i, p. 628–632, 1980.
- REISER, U. et al. Care-o-bot® 3-creating a product vision for service robot applications by integrating design and technology. In: **IROS**. [S.l.: s.n.], 2009. v. 9, p. 1992–1998.
- ROSENBLATT, F. **Principles of neurodynamics. perceptrons and the theory of brain mechanisms**. [S.l.], 1961.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, C. P. R. (Ed.). Cambridge, MA, USA: MIT Press, 1986. cap. Learning Internal Representations by Error Propagation, p. 318–362. ISBN 0-262-68053-X. Disponível em: <<http://dl.acm.org/citation.cfm?id=104279.104293>>.
- _____. Neurocomputing: Foundations of research. In: ANDERSON, J. A.; ROSENFELD, E. (Ed.). Cambridge, MA, USA: MIT Press, 1988. cap. Learning Internal Representations by Error Propagation, p. 673–695. ISBN 0-262-01097-6. Disponível em: <<http://dl.acm.org/citation.cfm?id=65669.104449>>.

- RUSSELL, R. A. An effective heuristic for the m-tour traveling salesman problem with some side conditions. **Operations Research**, INFORMS, v. 25, n. 3, p. 517–524, 1977.
- RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Third. Upper Saddle River, NJ: Prentice Hall, 2010. (Series in Artificial Intelligence). Disponível em: <<http://aima.cs.berkeley.edu/>>.
- RYAN, J. L. et al. Reactive tabu search in unmanned aerial reconnaissance simulations. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of the 30th conference on Winter simulation**. [S.l.], 1998. p. 873–880.
- SADIQ, S. The traveling salesman problem: Optimizing delivery routes using genetic algorithms. **SAS Global Forum 2012**, 2012.
- SAFFIOTTI, A.; RUSPINI, E.; KONOLIGE, K. **A Fuzzy Controller For Flakey, An Autonomous Mobile Robot**. 333 Ravenswood Ave., Menlo Park, CA 94025, 1993.
- SAKURAI, Y. et al. Backtrack and restart genetic algorithm to optimize delivery schedule. In: YÉTONGNON, K.; DIPANDA, A.; CHBEIR, R. (Ed.). **SITIS**. [S.l.]: IEEE Computer Society, 2010. p. 85–92. ISBN 978-0-7695-4319-2.
- _____. Inner random restart genetic algorithm to optimize delivery schedule. In: **SMC**. [S.l.]: IEEE, 2010. p. 263–270.
- SALEH, H. A.; CHELOUAH, R. The design of the global navigation satellite system surveying networks using genetic algorithms. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 17, n. 1, p. 111–122, 2004.
- SAMSUDIN, K.; AHMAD, F. A.; MASHOHOR, S. A highly interpretable fuzzy rule base using ordinal structure for obstacle avoidance of mobile robot. **Applied Soft Computing**, Elsevier, v. 11, n. 2, p. 1631–1637, 2011.
- SAVELSBERGH, M. W. P.; SOL, M. The general pickup and delivery problem. **TRANSPORTATION SCIENCE**, v. 29, p. 17–29, 1995.
- SEDIGHPOUR, M.; YOUSEFIKHOSHBAKHT, M.; DARANI, N. M. An effective genetic algorithm for solving the multiple traveling salesman problem. **Journal of Optimization in Industrial Engineering**, v. 8, p. 73–79, 2011.
- SEZER, V.; GOKASAN, M. A novel obstacle avoidance algorithm: “follow the gap method”. **Robotics and Autonomous Systems**, Elsevier, v. 60, n. 9, p. 1123–1134, 2012.
- SONG, C.-H.; LEE, K.; LEE, W. D. Extended simulated annealing for augmented tsp and multi-salesmen tsp. In: IEEE. **Neural Networks, 2003. Proceedings of the International Joint Conference on**. [S.l.], 2003. v. 3, p. 2340–2343.
- STACHNISS, C.; BURGARD, W. Particle filters for robot navigation. **Foundations and Trends in Robotics**, Now Publishers Inc., v. 3, n. 4, p. 211–282, 2014.
- STENTZ, A. Optimal and efficient path planning for partially-known environments. In: IEEE. **Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on**. [S.l.], 1994. p. 3310–3317.

- STENTZ, A.; HEBERT, M. A complete navigation system for goal acquisition in unknown environments. **Autonomous Robots**, v. 2, p. 127–145, 1995.
- STILMAN, M.; KUFFNER, J. J. Navigation among movable obstacles: Real-time reasoning in complex environments. **International Journal of Humanoid Robotics**, World Scientific, v. 2, n. 04, p. 479–503, 2005.
- SVESTKA, J. A.; HUCKFELDT, V. E. Computational experience with an m-salesman traveling salesman algorithm. **Management Science**, INFORMS, v. 19, n. 7, p. 790–799, 1973.
- TAMILSELVI, D. et al. Optimal path selection for mobile robot navigation using genetic algorithm in an indoor environment. In: _____. **Advanced Computing, Networking and Security: International Conference, ADCONS 2011, Surathkal, India, December 16-18, 2011, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 263–269. ISBN 978-3-642-29280-4. Disponível em: <https://doi.org/10.1007/978-3-642-29280-4_31>.
- TANG, L. et al. A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. **European Journal of Operational Research**, Elsevier, v. 124, n. 2, p. 267–282, 2000.
- TANG, Q.; EBERHARD, P. Cooperative motion of swarm mobile robots based on particle swarm optimization and multibody system dynamics. **Mechanics based design of structures and machines**, Taylor & Francis, v. 39, n. 2, p. 179–193, 2011.
- TECHI. **Techi**. 2017. Last access: 2017-09-07. Disponível em: <<http://techmetrics-group.com/>>.
- TERESA, Z. History of service robots. In: **Concepts, Methodologies, Tools, and Applications**. IGI Global, 2014. p. 1–14. Disponível em: <<http://dx.doi.org/10.4018/978-1-4666-4607-0.ch001>>.
- THRUN, S. et al. Minerva: A second-generation museum tour-guide robot. In: **In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'99)**. [S.l.: s.n.], 1999.
- THRUN, S.; BÜCKEN, A. **Learning Maps for Indoor Mobile Robot Navigation**. [S.l.], 1996.
- THRUN, S. et al. Robotic mapping: A survey. **Exploring artificial intelligence in the new millennium**, v. 1, p. 1–35, 2002.
- TORKI, A.; SOMHON, S.; ENKAWA, T. A competitive neural network algorithm for solving vehicle routing problem. **Computers & industrial engineering**, Elsevier, v. 33, n. 3, p. 473–476, 1997.
- UNIVERSITY OF WAIKATO. **WEKA 3 - Data Mining with Open Source Machine Learning Software in Java**. Disponível em: <<http://www.cs.waikato.ac.nz/ml/weka/>>.

VAKHUTINSKY, A. I.; GOLDEN, B. Solving vehicle routing problems using elastic nets. In: IEEE. **Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on**. [S.l.], 1994. v. 7, p. 4535–4540.

VELOSO, M. et al. Cmpack-02: Cmu's legged robot soccer team. In: **Proceedings of the RoboCup Symposium**. [S.l.: s.n.], 2002.

VELOSO, M. M. et al. Cobots: Robust symbiotic autonomous mobile service robots. In: **Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015**. [s.n.], 2015. p. 4423. Disponível em: <<http://ijcai.org/Abstract/15/656>>.

_____. Cobots: Robust symbiotic autonomous mobile service robots. In: YANG, Q.; WOOLDRIDGE, M. (Ed.). **IJCAI**. AAAI Press, 2015. p. 4423–. ISBN 978-1-57735-738-4. Disponível em: <<http://dblp.uni-trier.de/db/conf/ijcai/ijcai2015.html#VelosoBCR15>>.

WANG, M.; LIU, J. N. Fuzzy logic-based real-time robot navigation in unknown environment with dead ends. **Robotics and Autonomous Systems**, Elsevier, v. 56, n. 7, p. 625–643, 2008.

WEISS, G.; PUTTKAMER, E. v. A map based on laserscans without geometric interpretation. In: IOS PRESS. **Intelligent Autonomous Systems**. [S.l.], 1995. v. 4, n. 2, p. 403–407.

WISE, M. et al. Fetch and freight: Standard platforms for service robot applications. In: **Workshop on Autonomous Mobile Service Robots: 25th International Joint Conference on Artificial Intelligence (IJCAI)**. [S.l.: s.n.], 2016.

WOODEN, D. T. Graph-based path planning for mobile robots. Georgia Institute of Technology, 2006.

WU, B.-F.; JEN, C.-L. Particle-filter-based radio localization for mobile robots in the environments with low-density wlan aps. **IEEE Transactions on Industrial Electronics**, v. 61, n. 12, p. 6860–6870, 2014.

WURMAN, P. R.; D'ANDREA, R.; MOUNTZ, M. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In: **Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence - Volume 2**. AAAI Press, 2007. (IAAI'07), p. 1752–1759. ISBN 978-1-57735-323-2. Disponível em: <<http://dl.acm.org/citation.cfm?id=1620113.1620125>>.

YAN, Z.; JOUANDEAU, N.; ALI-CHÉRIF, A. Multi-robot heuristic goods transportation. In: IEEE. **Intelligent Systems (IS), 2012 6th IEEE International Conference**. [S.l.], 2012. p. 409–414.

YANAR, T. A.; AKYÜREK, Z. Fuzzy model tuning using simulated annealing. **Expert Syst. Appl.**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 38, n. 7, p. 8159–8169, jul. 2011. ISSN 0957-4174. Disponível em: <<http://dx.doi.org/10.1016/j.eswa.2010.12.159>>.

YU, W.; LU, L. A route planning strategy for the automatic garment cutter based on genetic algorithm. In: **2014 IEEE Congress on Evolutionary Computation (CEC)**. IEEE, 2014. Disponível em: <<http://dx.doi.org/10.1109/CEC.2014.6900425>>.

- YU, Z. et al. An implementation of evolutionary computation for path planning of cooperative mobile robots. In: IEEE. **Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on**. [S.l.], 2002. v. 3, p. 1798–1802.
- YUN, S. C.; GANAPATHY, V.; CHONG, L. O. Improved genetic algorithms based optimum path planning for mobile robot. In: IEEE. **Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on**. [S.l.], 2010. p. 1565–1570.
- ZADEH, L. A. Fuzzy sets. **Information and control**, Elsevier, v. 8, n. 3, p. 338–353, 1965.
- ZHANG, T.; GRUVER, W.; SMITH, M. H. Team scheduling by genetic search. In: IEEE. **Intelligent Processing and Manufacturing of Materials, 1999. IPMM'99. Proceedings of the Second International Conference on**. [S.l.], 1999. v. 2, p. 839–844.
- ZHANG, Y.; GONG, D.-w.; ZHANG, J.-h. Robot path planning in uncertain environment using multi-objective particle swarm optimization. **Neurocomputing**, Elsevier, v. 103, p. 172–185, 2013.
- ZHU, Q.; YAN, Y.; XING, Z. Robot path planning based on artificial potential field approach with simulated annealing. In: IEEE. **Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on**. [S.l.], 2006. v. 2, p. 622–627.
- ZITZLER, E.; LAUMANN, M.; THIELE, L. **SPEA2: Improving the Strength Pareto Evolutionary Algorithm**. [S.l.], 2001.
- ZLOT, R.; STENTZ, A. Complex task allocation for multiple robots. In: IEEE. **Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on**. [S.l.], 2005. p. 1515–1522.
- ZOHAIB, M. et al. An improved algorithm for collision avoidance in environments having u and h shaped obstacles. **Studies in Informatics and Control**, v. 23, n. 1, p. 97–106, 2014.