Gonçalo Nuno dos Santos Augusto

RobotTeamSim – 3D Visualization of Cooperative Mobile Robot Missions in Gazebo Virtual Environment

Universidade de Coimbra

Department of Electrical and Computer Engineering

Faculty of Sciences and Technology

University of Coimbra

Master of Science in Electrical and Computer Engineering

RobotTeamSim - 3D Visualization of Cooperative Mobile Robots Missions in Gazebo Virtual Environment

Supervisor:

Prof. Doutor Rui P. Rocha

Co-Supervisor:

Eng. Micael Couceiro

Jury:

Prof. Doutor Rui Cortesão

Prof. Doutor Paulo Menezes

Prof. Doutor Rui P. Rocha

Coimbra, September 2013

Agradecimentos

Queria em primeiro lugar agradecer aos meus pais, sem os quais nada teria sido possível e também por compreenderem que o curso não é fácil e me terem apoiado sempre que possível. De seguida queria agradecer ao Prof. Doutor Rui Rocha por esta oportunidade, pela atenção em responder sempre o mais rápido possível a qualquer dúvida e também pela compreensão dos problemas alheios à minha pessoa que dispenderam uma quantia elevada de tempo a resolver. Agradeço ao Instituto de Sistemas e Robótica da Universidade de Coimbra pelo espaço, material e condições ideais para a realização da dissertação. Um grande abraço e agradecimento ao Eng. Micael Couceiro e ao Eng. David Portugal pelo apoio e ajuda em qualquer problema que pudesse surgir e que, para além de co-orientadores, também foram grandes amigos. Bro fist aos meus colegas de mestrado que, nos melhores e piores momentos, sempre mantiveram o espírito do grupo em alta, sempre dispostos a ajudar em qualquer problema quando possível e pelas gargalhadas que ajudavam a animar nos momentos mais baixos. Um abraço para Shanker Keshavdas por ter ajudado a resolver um problema com o software que, caso não tivesse sido resolvido, poderia ter deitado toda a dissertação por terra. E "the last but not the least" uma agradecimento especial a todas as pessoas dos vários laboratórios do ISR pela boa recepção e pelo ambiente acolhedor que proporcionou uma rápida e boa adaptação.

Resumo

Esta dissertação tem como finalidade a criação e simulação de missões de busca e salvamento através do ambiente virtual do Gazebo e o Robotic Operating System (ROS). O simulador é avaliado beneficiando de exploração de múltiplos robôs com o algoritmo Robotic Darwinian Particle Swarm Optimization (RDPSO), considerando fenómenos reais, tais como propagação de sinais de radiofrequência (RF) e voz. Embora o algoritmo RDPSO já tenha sido desenvolvido e testado, a sua implementação em ambiente de simulação (tipicamente Matlab) e em plataformas reais tem sido vista como dois processos independentes. Para ultrapassar esta desvantagem, ROS foi escolhido para migrar o algoritmo RDPSO e, devido à sua compatibilidade com o ROS, o Gazebo foi a plataforma de simulação escolhida para avaliar sistemas de múltiplos robôs em cenários de busca e salvamento. Para o algoritmo RDPSO ser completamente implementado, a propagação de sinais de RF e voz foram de igual modo implementadas em ROS.

Para o efeito, são analisados modelos matemáticos previamente propostos na literatura providenciando um estudo da influência do ruído ambiente em missões de busca e salvamento baseadas na localização por voz. O modelo de RF foi simulado usando o método multi-wall, que considera não só a perda de sinal em espaço livre mas também as perdas relativas a paredes de diferentes características (e.g., espessuras e tipo). O modelo de voz é baseado no modelo RF anterior manipulando os parâmetros para se adaptar melhor às propriedades da voz. O algoritmo RDPSO, implementado em ROS, usa o modelo de RF para simular a sua componente de conectividade da rede ad hoc móvel (MANET). O algoritmo RDPSO é utilizado no contexto de encontrar vítimas num cenário de busca e salvamento, utilizando o modelo de propagação de voz para simular o pedido de ajuda das vítimas. No cenário da simulação foi introduzido ruído de fundo, que influenciou o sucesso de salvamento das vítimas ao dificultar a audição destas pela equipa de salvamento. Esta influência torna-se maior quanto maior o nível de ruído ambiente, ao qual é necessária uma melhor exploração do mapa pois a equipa de salvamento tem de estar cada vez mais próxima das vítimas para poder distinguir a voz de ruído de fundo.

Palavras-chave: Gazebo, swarm robotics, busca e salvamento, MANET, modelo multi-wall.

Abstract

This dissertation intends to create a simulation of search and rescue (SaR) missions using Gazebo virtual world and Robot Operating System (ROS). The simulator is evaluated with multiple robots search using the Robotic Darwinian Particle Swarm Optimization (RDPSO) exploration algorithm, considering several real-world phenomena, such as radio frequency (RF) and voice. Although the RDPSO algorithm has already been developed and evaluated in Matlab environment, it cannot be directly implemented in real platforms. To address this disadvantage, ROS was chosen to implement the algorithm and, due to its compatibility with ROS, Gazebo was chosen as the simulation platform to evaluate multi-robot systems in SaR scenarios. For the RDPSO to be fully implemented, the RF signal and voice propagation models were equally implemented in ROS.

For that matter, mathematic models, previously proposed in the literature, are analyzed and a study about the environment noise influence in missions of SaR based on voice localization is conducted. The RF model was simulated using the multi-wall method, which does not only consider the free space signal loss but also the loss of walls with different properties (e.g., thickness and type). The voice model is based on the RF model by adjusting the parameters to better adapt to the voice properties. The RDPSO algorithm, implemented in ROS, uses the RF model to simulate its mobile ad hoc network (MANET) connectivity component. In the context of finding victims, the RDPSO uses the voice propagation model to simulate the call for help by the victims. Environment noise was introduced in the simulation, which influenced the victims' rescue rate by making harder for the rescue team to listen to the victim's call. This influence is bigger as the environment noise level gets higher, thus there is the need for a better exploration of the map as the rescue team has to walk closer and closer to the victims, so that voice and noise can be distinguished.

Keywords: Gazebo, swarm robotics, search and rescue, MANET, multi-wall model.

Contents

Agradecimentos v
Resumo vii
Abstract ix
List of Acronyms xiii
List of Figures xv
List of Tables xvii
CHAPTER 1 1
1 - Introduction 1
1.1 Context and Motivation 1
1.2 Objectives 1
1.3 Outline of the dissertation 2
CHAPTER 2 3
2 - Mobile Robot Simulators 3
2.1 Most used mobile robot simulators 3
2.1.1 Commercial Simulators 4
2.1.2 Open source/free-to-use simulators 5
2.2 Comparison of commercial and free-to-use simulators 7
Characteristics 7
2.3 Example of a SaR simulation using the MRSim simulator 9
2.4 Summary 10
CHAPTER 3 11
3 - Gazebo Simulator 11
3.1 Description 11
3.2 Integration with ROS 11
3.2.1 ROS-Integrated plugin 12
3.3 Application example 13
3.4 Evaluation of Gazebo's performance 13
3.5 Summary 14
CHAPTER 4 15
4 - Models for Simulating Robotic Missions in Urban Firefighting Scenarios 15
4.1 Radio Frequency communication signal propagation model 15
4.2 Voice propagation model 18
4.3 Implementation in ROS 20

4.4 Summary ... 20
CHAPTER 5 ... 21
5 - Particle Optimization Algorithms for Multi-Robot Search Simulations ... 21
5.1 Particle Swarm Optimization and Darwinian Particle Swarm Optimization ... 21
5.1.1 Particle Swarm Optimization ... 21
5.1.2 Darwinian Particle Swarm Optimization ... 22
5.2 Robotic PSO and Robotic Darwinian PSO ... 24
5.2.1 Robotic Particle Swarm Optimization ... 24
5.2.2 Robotic Darwinian Particle Swarm Optimization ... 24
5.3 Algorithm Simulation ... 35
5.4 Results and Discussion ... 40
5.4.1 Results ... 40
5.4.1 Discussion and analysis ... 43
5.5 Summary ... 46
CHAPTER 6 ... 47
6 - Conclusion and Future Work ... 47
6.1 Conclusion ... 47
6.2 Future Work ... 48
References and Bibliography ... 49
Appendix 1 – ROS Plugin ... 53
Appendix 2 – Gazebo World, Model and Plugin tutorial ... 57

List of Acronyms

2D	Two dimensional
3D	Three dimensional
CC	Command center
CHOPIN	Cooperation between Humans and rObotic teams in catastroPhic INcidents
DPSO	Darwinian Particle Swarm Optimization
FC	Fractional calculus
ISR-UC	Institute of Systems and Robotics of University of Coimbra
MANET	Mobile ad hoc network
MRL	Mobile Robotics Laboratory
PSO	Particle Swarm Optimization
OS	Operating System
OSRF	Open Source Robotics Foundation
RDPSO	Robotic Darwinian Particle Swarm Optimization
RF	Radio frequency
ROS	Robot Operating System
RPSO	Robotic Particle Swarm Optimization
SaR	Search and rescue
SEBS	State Exchange Bayesian Strategy

List of Figures

Figure 1: Some mobile robotic platforms, iRobot Create, Turtlebot, Mindstorm NXT, e-puck, MarXbot, SRV-1 Blackfin and Pioneer 3-DX. 3
Figure 2: Robot 2D arena (Stage), real arena and 3D arena (Gazebo). 3
Figure 3: Search and Rescue simulation scenario in DEEC garage. 10
Figure 4: Pioneer mobile robot equipped with a camera and Hokuyo laser range finder. 11
Figure 5: Virtual arena from MRL in Gazebo and the created map visualized in ROS with the RVIZ tool. 13
Figure 6: Radio Signal loss maps with 2 different transmitter positions. 17
Figure 7: Voice signal strength maps from different sources, lc =-78 and $\gamma = 5$. 19
Figure 8: Comparison of propagation ranges for different voice propagation model parameters. 19
Figure 9: Flock of birds and fish school. 21
Figure 10: Cognitive and Social components in a 2D scenario. 22
Figure 11: Example of the topology of a MANET due to signal quality loss in an office-like scenario. 26
Figure 12: DEEC garage in stage (2D) 35
Figure 13: DEEC garage simulated in Gazebo (3D) 35
Figure 14: Social included and excluded robot's behavior during the SaR simulation. 36
Figure 15: Simulation set-up configuration, waypoints and victims. 37
Figure 16: Lowest voice detection range for different background noise levels with 4 victims. 38
Figure 17: Highest voice detection range for different background noise levels with 4 victims. 38
Figure 18: MANET size for different connection thresholds for a team of three robots. 39
Figure 19: Victims saved for different levels of environment noise. 43
Figure 20: Mission durations for different levels of environment noise. 43
Figure 21: Waypoints explored for different levels of environment noise. 44
Figure 22: Number of times the MANET connection was endangered for different levels of environment noise. 44

List of Tables

Table 1: Comparison commercial simulators. 8
Table 2: Comparison of free-to-use simulators. 9
Table 3: Multi-wall parameters 17
Table 4: Adjacency matrix A of the scenario in Fig. 11. 28
Table 5: Connectivity matrix $C^{(7)}$ of the scenario in Fig. 11. 28
Table 6: Number of victims saved and respective simulation duration for 3 different environment noise levels 41
Table 7: Number of waypoints explored and number of times the MANET connection was endangered for 3 different environment noise levels. 42
Table 8: Mean, median and mode values for Table 6. 45
Table 9: Mean, median and mode values for Table 7. 45

CHAPTER 1

1 - Introduction

1.1 Context and Motivation

For many years, virtual simulations have played an important role in robotic research by enabling quick testing and data gathering of algorithms without endangering people, the environment and robotic platforms.

Due to the limited computer power, 2D simulators were mainly preferred over 3D. However, in recent years, with the evolution of hardware, 3D simulators have started to gain some recognition within the research community, thus allowing the simulation of realistic world environments and "bodies". One of the most well-known 3D simulators is Gazebo [1]. Gazebo not only simulates real world physics and rigid-body dynamics but is also compatible with the Robot Operating System (ROS) [2], used worldwide in mobile robot research and whose controllers can be implemented directly in real life robotic platforms. This compatibility allows previously developed algorithms in ROS to be simulated in 3D scenarios with little to no changes in the code through Gazebo.

At the Institute of Systems and Robotics[1] from University of Coimbra (ISR-UC) many algorithms have been developed and implemented in robots. Unfortunately, some of these algorithms have been simulated only in 2D scenarios or in other simulation environments whose code cannot be ported directly into robotic platforms. In this dissertation, we illustrate how robotic algorithms can be simulated in 3D with two open-source software platforms, Gazebo and ROS, using one algorithm initially developed and evaluated in Matlab, as a case study.

1.2 Objectives

Gazebo was created and developed to run with *Player* [3]. Latter on, it was adapted to work with ROS and, recently, it has split himself from any 3rd party software and became self-sufficient. Although it no longer requires ROS to run, a customizable "bridge" between both can be created in order to keep viable the simulation in Gazebo of algorithms developed in ROS. The first objective of this dissertation is to provide and thoroughly explain such bridge. Moreover, this

[1] http://www.isr.uc.pt/

dissertation presents a tutorial around the creation and use of worlds in Gazebo, in order to run past and future works on ROS in the "new" standalone Gazebo.

This dissertation also presents the implementation of some real-world phenomena in ROS, such as voice and radio frequency signal (RF), so as to extend Gazebo simulator to search and rescue (SaR) applications. Afterwards, the Gazebo interface will be evaluated using a swarm robotic algorithm. The swarm algorithm previously proposed in Couceiro *et al.* [4], denoted as Robotic Darwinian Particle Swarm Optimization (RDPSO), consists on an algorithm to control multiple robots within exploration missions such as SaR. This was evaluated in the context of the CHOPIN[2] project at ISR, and simulated in MRSim (Matlab) which does not possess any cross-compilation tool to directly port the code into real robots. To extend the applicability of the RDPSO approach, the algorithm was fully developed in ROS.

1.3 Outline of the dissertation

This dissertation is divided into 6 chapters, the first of which presents the context and motivation of this dissertation. The second chapter briefly analyses some of the most known robotic simulators, comparing them by resorting to the related work. Chapter 3 presents a brief description of the Gazebo simulator, followed by a tutorial on how to create a ROS-Integrated plugin (*i.e.* bridge between ROS and Gazebo) which is exemplified by controlling a Gazebo model with some ROS stacks. Chapter 4 explains some relevant propagation models (RF, voice) and how to implement them in ROS. Afterwards, in chapter 5, a simplified version of the RDPSO algorithm is implemented in the ROS architecture and tested under those previously presented models. Finally, chapter 6 presents the future work and outlines the main conclusions.

[2] http://chopin.isr.uc.pt/

CHAPTER 2

2 - Mobile Robot Simulators

2.1 Most used mobile robot simulators

Over the past years, virtual simulators have played an important role in robotic research, thus avoiding common hardware problems, such as short battery life, hardware failures and unexpected or dangerous behaviors. Fig. 1 depicts some robotic platforms commonly used for research purposes [5].

Figure 1: Some mobile robotic platforms, iRobot Create, Turtlebot, Mindstorm NXT, e-puck, MarXbot, SRV-1 Blackfin and Pioneer 3-DX.

Due to the computational weight required, the 2D simulators have been preferred over the 3D ones for many years, being these latest heavily restricted by hardware limitations in terms of computational power when simulating world physics. Yet, with the recent technology developments, 3D simulators have started to be more commonly used. In Fig. 2 we see the same arena depicted in Stage, real life and Gazebo.

Figure 2: Robot 2D arena (Stage), real arena and 3D arena (Gazebo).

As such there has been an increase in the number of available robot simulators (pay-to-use/commercial, free-to-use/license and open source). This chapter briefly presents some of the

more well-known simulators and their main features, like supported operating system, simulation type, programming language, portability, among others.

2.1.1 Commercial Simulators

Let us first consider the pay-to-use/commercial simulators.

Webots

The *Webots* is a 3D simulator for Mac OSX, Windows and Linux for simulation, modeling and programing mobile robots [6]. *Webots* can use C, C++, Java, Matlab and Python to create the control programs which can be implemented in real life robot platforms like e-puck, Pioneer, iRobot, NAO robot, etc. *Webots* supports several sensors like odometry, range, camera, light and touch.

Simulated models have several customizable attributes like shape, color, texture, mass and friction. These attributes allow the simulation of real body dynamics and world physics, thus requiring a nowadays standard graphic card for better results. One of the available tools is the terrain generator using Google maps and GeoNames to generate textured terrains. A simulation can have as many robots as needed. The memory and the CPU power are the only limits.

Webots can also connect to ROS using roscpp (C++) or rospy (Python) controller interfaces, allowing the use of all ROS stacks.

Easy-Rob

Easy-Rob is a Windows-only 3D simulator (Windows 7, XP and vista) for planning and simulation in manufacturing plants that operate within work cells [7]. *Easy-Rob* allows the user to program and visualize 3D processes using robots (single or multiple robots) in industrial tasks such as handling, assembly, coating and sealing.

The simulation controllers' code cannot be implemented in real robots. Since it does not simulate realistic rigid body dynamics nor world physics, it has low hardware requirements and, as a consequence, it can run and simulate several robots and kinematics synchronized in real time, being possible to easily run in any 32 or 64 bit software designed for any standard Windows PC operating with the OpenGL graphic library.

MRSim

Multi-Robot Simulator (MRSim) [8] is a 2D simulator for MatLab, being a toolbox for Mac OSX, Linux and Windows created in 2012 at the Mobile Robotics Laboratory[3] (MRL) from ISR-UC. Although *MRSim* toolbox is open source, MatLab is a pay-to-use tool. The controller's code is

[3] http://mrl.isr.uc.pt/

written in MatLab language and it is not portable to real robotic platforms. It allows simulating distance sensors (typically sonars and laser sensors) and already includes a large number of real-world features such as RF, voice and fire propagation.

Nevertheless, it does not simulate object dynamics or any other 3D feature and thus has low hardware requirements running in anything that can run MatLab. Due to all these simplifications, it can simultaneously simulate a high number of robots.

RoboticsLab

RoboticsLab is a Windows-only 3D robot simulator (Windows 7, Vista and XP) [9]. The creation of the control program uses C, C++, Python and Java, being portable to real robotic platforms. It simulates range, camera and odometry sensors. Moreover, it also allows simulation of realistic environments, world physics and object dynamics, thus requiring, at least, an Intel core2duo with 1GB RAM and a standard graphic card with OpenGL lib.

V-Rep

V-Rep is a 3D robotic simulator for Windows, Mac OSX and Linux [10]. The programming can be made using C, C++, Python, Java, Lua, MatLab and Urbi. The code is only portable into real robotic platforms if used with ROS. However, ROS has the downside of offering limited services to V-Rep. It provides simulations for camera and proximity/range sensors for collision avoidance. It requires the nowadays standard graphic card to generate realistic environments world physics and object dynamics hence reducing the number of robots simulated in real time. On a side note, *V-Rep* is free to use for students only.

2.1.2 Open source/free-to-use simulators

Now we are going to briefly present the most known free-to-use and open source simulators.

Stage

Stage is a 2D simulator for Linux, Mac OSX and Windows for robot simulations [3], most commonly used as a *Player* plugin module. The controllers can be coded in C, C++, Python and Java through Player or ROS and the written code is portable to real robotic platforms. It simulates range and odometry sensors.

It has low hardware requirements due to the fact of being 2D providing a basic simulation environment that can be scaled to model one to hundreds of robots at a time. Although it can create 3D visual simulations, the data and mapping is always in 2D.

Gazebo

Gazebo is a Linux-only 3D robotic simulator [1]. The controllers can be written in C, C++, Python and Java by using Player or ROS and the written code is portable to real robotic platforms. It supports simulations for odometry, range and camera sensors. It is capable of simulating very realistic environments, world physics and sensor feedback. Moreover, it also simulates very accurately rigid body dynamics, requiring for that matter a good graphic card and CPU processor (not much better than a nowadays standard hardware) and OpenGL Lib. Therefore, the number of robots simulated in real time is limited to the hardware performance.

On a side note, *Gazebo* and *Stage* are both compatible with ROS. As result, a client written for one of them (*i.e.* either Gazebo or Stage) can usually run on the other with little to no modification. While *Stage* provides real time simulation for a high number of robots with low fidelity, *Gazebo* provides real time simulation for a low number of robots, highly dependable on the PC hardware, but with very high fidelity.

Simbad

Simbad is a Java only 3D robotic simulator for Linux, Mac OSX and Windows for scientific and educational purposes [11]. The code portability is limited due to its exclusivity to Java language. It simulates vision, distance and contact sensors.

Although it can simulate 3D environments, it does not simulate world physics or object dynamics. Hence, its main purpose is to be a tool for studying Situated Artificial Intelligence, Machine Learning and more generally Artificial Intelligence (AI) algorithms, in the context of Autonomous Robotics and Autonomous agents, *i.e.*, it is not intended for real world simulation. This simplicity gives it low hardware requirements to run in comparison to other 3D simulators, only requiring Java 3D library, and grants it a higher number of robots that can be simulated in real time.

CARMEN

Carnegie Mellon Robot Navigation Toolkit (CARMEN) is a Linux only 2D robotic simulator [12]. The code is written in C or Java and is portable into real robotic platforms. It supports odometry, distance and GPS sensors. Nevertheless, it cannot simulate realistic environments, world physics and object dynamics and thus has low hardware requirements. Being less hardware dependent it can simulate higher number of robots in real time.

USARSim

Unified System for Automation and Robot Simulation (USARSim) is a 3D simulator for Windows and Linux based on the Unreal Tournament (UT) game engine [13]. The controller's code can be written in C, C++ and Java and it is only portable to real robotic platforms if using *Player* or *ROS*. It supports odometry, distance, camera and touch sensors. It simulates realistic environments, world physics and object dynamics, requiring a standard graphic card, UT2004 with 3339 or latter patch. For controllers, it is recommended *Mobility Open Architecture Simulation and Tools* (MOAST) which is fully integrated with *USARSim* but can also be used Player 1.4rc2 or higher with Player USARSim drivers.

Although USARSim is also compatible with ROS, the bridge required is still under development and does not offer a level of compatibility as high as Gazebo.

RDS

Microsoft Robotics Developer Studio (MRDS or RDS) is a Windows-only 3D robotic simulator [14]. As *MRDS* is licensed, the developer can use it for free and is granted with some rights to use the software although all other rights are reserved to Microsoft. The controllers can be created using VPL (Visual Programing Language), C#, Visual Basic, Jscript, Iron-Python and are portable into real robotic platforms. It supports odometry, distance and camera sensors and allows simulating realistic environments, world physics and objecting dynamics. The latest version, *RDS4*, requires a PC that is capable of running Windows7 and MRDS with Microsoft DirectX 9.0c compatible graphic card, dual-core processor (minimum of 2GHz), 10GBs available disk space and minimum of 2GB memory (4GB are recommended).

MissionLab

MissionLab is a Linux only 3D simulator for multi-agent robotics mission specification and control [15]. The controllers are built using VPL (*Visual Programing Language*) and are portable to real robotic platforms. It supports odometry and range sensors. Since it does not recreate realistic environments, world physics and object dynamics, it has low hardware requirements.

2.2 Comparison of commercial and free-to-use simulators

Characteristics

Tables 1 and 2 compare the aforementioned simulators and several of their characteristics based on the current state-of-the-art. Table 1 compares commercial simulators while Table 2 focuses on free-to-use simulators. The simulators are compared through the following features:

- *Operative System*: OS in which the simulator can run.
- *Simulator Type*: 2D or 3D
- *Programing language*: language in which the controllers can be written.
- *Portability*: if the controllers' code is directly portable into real robotic platforms.
- *Sensors*: most requested supported sensors.
- *Requirements*: *Low*-the simulator can run on basically every nowadays PC; *Standard*-nowadays average/good commercial PC will be able to run the simulator; *High*- requires nowadays expensive hardware to run; *Specific*- specific hardware or software required to run.
- *Realistic environment*: if it is able to model realistic environments and world physics.
- *Object Dynamic*: if it is able to simulate rigid body dynamics or object interaction (lifting, pushing, dropping, etc.).

	Webots	Easy-Rob	MRSim	RoboticsLab	V-Rep
Operating System	Mac, Win, Linux	Win	Mac, Linux, Win	Win	Win, Mac, Linux
Simulator type	3D	3D	2D	3D	3D
Programming language	C, C++, Java, MatLab, Python	C++	MatLab	C, C++, Python, Java	C, C++, Python, Java, Lua, Matlab, Urbi
Portability	Yes	No	No	Yes	With ROS only
Sensors	Odometry, range, camera, GPS	Odometry	Range	Camera, range, odometry	Camara, range
Requirements	Standard	Low	Low	Standard	Standard
Realistic environment	Yes	No	Yes	Yes	Yes
Object dynamics	Yes	Object interaction	No	Yes	Yes

Table 1: Comparison commercial simulators.

	Stage	Gazebo	Simbad	CARMEN	USARSim	RDS	MissionLab
Operating System	Linux, Mac, Win	Linux	Linux, Mac, Win	Linux	Linux, Win	Win	Linux
Simulator type	2D	3D	3D	2D	3D	3D	3D
Programming language	C, C++, Python, Java	C, C++, Python, Java	Java	C, Java	C, C++, Java	VPL, C#, Visual Basic, Jscript, Iron-Python	VPL
Portability	Yes	Yes	Limited	Yes	Yes	Yes	Yes
Sensors	Odometry, range	Odometry, range, camera	Camara, range, contact	Odometry, range, GPS	Odometry, range, camera, touch	Odometry, range, camera	Odemetry, range
Requirements	Low	Standard	Low	Low	Specific	Specific	Low
Realistic environment	No	Yes	No	No	Yes	Yes	No
Object dynamics	No	Yes	No	No	Yes	Yes	No

Table 2: Comparison of free-to-use simulators.

In Tables 1 and 2, we can see a summary of the information presented in the section 2.1. Table 1 covers commercial simulators, the simulators with more programming languages and compatibility of OS that still offer a large range of sensors and are able to simulate realistic physics are Webots and V-Rep. V-Rep comes second to Webots only due to its limited portability.

Table 2 covers open-source simulators. These simulators are in general more limited than the commercial ones but still offer a variety of good characteristics. Those that can simulate realistic and 3D environments, *i.e.*, Gazebo, USARSim and RDS, all offer good variety in programming languages and different kind of sensors. Gazebo was chosen for this dissertation due to its superior compatibility with ROS among 3D simulators.

2.3 Example of a SaR simulation using the MRSim simulator

MRSim toolbox for Matlab was created at ISR-UC with the purpose of testing algorithms and models for SaR scenarios. Some of the models include fire, voice and radio frequency signal propagations, as well as firefighters' and victims' behaviors. One of the algorithms tested was the RDPSO [4].

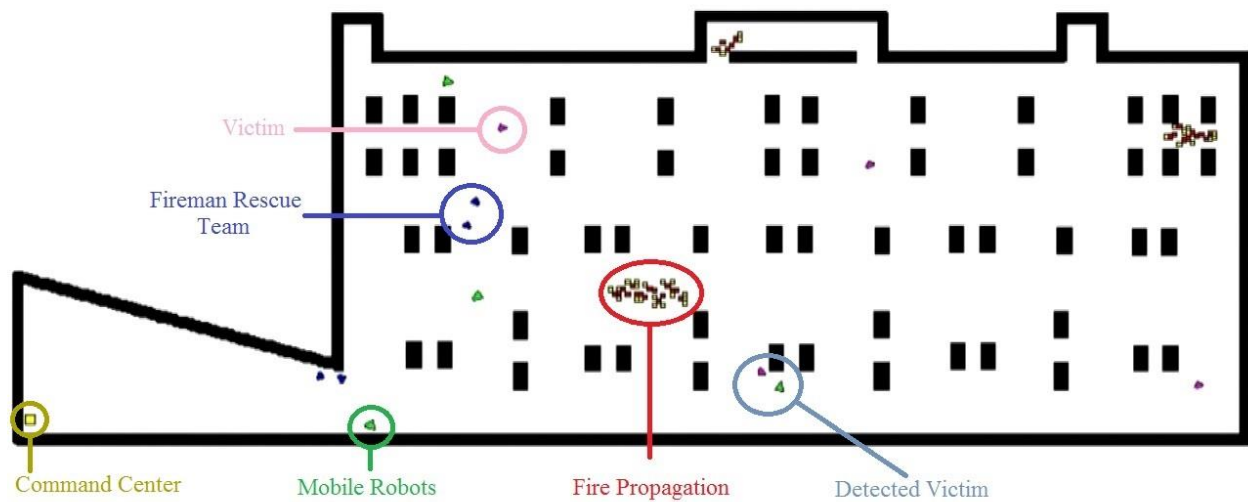


Figure 3: Search and Rescue simulation scenario in DEEC garage.

In Fig.3, the simulation of a fire incident in a large basement garage (ISR-UC garage) using MRSim is depicted. The mission objective is to rescue all the victims (pink cells). Victims' location is unknown to the firemen team (blue cells), so mobile robots (green cells) are deployed to help find their location. As soon as a victim is detected by a robot, its location is transmitted to the command center (CC) (yellow cells), thus affecting the firemen's behaviors towards the victim to "pick" it up and bring it safely to the CC. All this is done while maintaining a mobile ad hoc network (MANET) between the robots, the firemen and the CC. A MANET is a wireless network that supports multi-hop, the communication between two nodes (i.e., robots) is carried out through a number of intermediate robots by relaying information from one node to another. Meanwhile, the fire propagates along the scenario (red cells) and may block possible paths for both fireman and robots. In the worst case scenario, it can even cause casualties in the robot team, firemen team and victims. The MRSim main disadvantage relies on the fact that it is completely written for Matlab. As such, the simulation models and controllers cannot be directly implemented into robotic platforms. Besides it is limited to 2D simulations. To address these limitations, this dissertation aimed at migrating some of these models to the ROS architecture and recreate part of the simulation using Gazebo.

2.4 Summary

As this chapter suggests, many good simulators are available. Taking into account the environment realism, the array of available sensors and ROS compatibility, Gazebo and ROS were chosen to simulate the several real-world models and algorithms initially validated on the MRSim toolbox.

CHAPTER 3

3 - Gazebo Simulator

3.1 Description

Gazebo is a 3D simulator supported by the Open Source Robotics Foundation[4] (OSRF). In Fig. 4, we see a Pioneer mobile robot with a laser range finder and a camera on top in a Gazebo world.

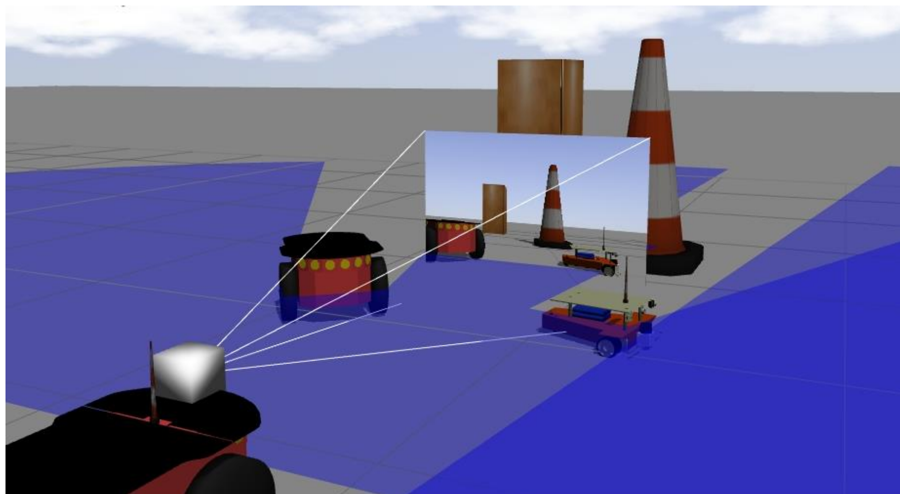


Figure 4: Pioneer mobile robot equipped with a camera and Hokuyo laser range finder.

Originally created to work over *Player*, it was further adapted to also work over ROS, eventually splitting from any 3rd party software and becoming independent and self-sufficient. Since both ROS and Gazebo use topics to share data within processes, a plugin bridge can be created, thus allowing the use of all ROS tools and stacks in Gazebo.

This ROS-Integrated plugin has to be created and customized according to simulation objectives, such as choosing and creating the needed topics. One example of one such plugin can be found in Appendix 1.

3.2 Integration with ROS

First, let us briefly introduce some of ROS's architecture keywords. ROS uses the concept of nodes, messages, topics, stacks, and packages, better described in [2]:

[4] http://www.osrfoundation.org/

- *Node* - A process that performs computation; nodes communicate with each other through messages.
- *Message* – A strictly type of data structure; a node sends a message by publishing it to a topic.
- *Topic* – Channel between two or more nodes; nodes communicate by publishing and/or subscribing to the appropriate topics.
- *Package* – Compilation of nodes that can easily be compiled and ported to other computers, necessary to build a complete ROS-based controller system.
- *Stack* – Groups of ROS packages making easier the process of sharing code with the community.

3.2.1 ROS-Integrated plugin

The plugin, as previously mentioned, needs to be customized accordingly to our needs. The ROS-Integrated plugin found in Appendix 1 is for a robot with a laser range finder. It is important to note that creating topics and publishing messages over them in ROS is different from doing the same in Gazebo. Hence, we have to use some ROS code for those actions even if the plugin is written in Gazebo. To do so, the following steps have to be done (Appendix 1):

1- Include the necessary libraries. Apart from the common C++ libs (e.g. "stdio.h", "math.h", etc.), we need to include Gazebo libs, more precisely "gazebo/sensors/RaySensor.hh" and "gazebo/sensors/SensorManager.hh", so that we can access the laser data. We also need to include the ROS library "ros/ros.h" and any type of message used in the ROS topics to be created (e.g. "geometry_msgs/Twist.h", "sensor_msgs/LaserScan.h", etc.).

2- Using the function *ROSModelPlugin()*, we initialize the plugin process as a ROS node.

3- The function *LOAD()* runs at the very start of the process, wherein we should create topics, pointers to sensors in the model and events such as defining the "main" function to run at each "iteration".

4- The "main" function of the plugin, *OnUpdate(),* is defined in *LOAD()* to execute in each "iteration" of the simulation. It is in this function that we create and publish messages, access sensor data and control the model.

A more detailed description on how to create a simple Gazebo world, a robot model and how to link a simple ROS-integrated plugin to a model is provided in the tutorial included in Appendix 2.

3.3 Application example

With a ROS-Integrated plugin, we can use any ROS stack within Gazebo. A simple simulation was made using two ROS stacks, *slam_gmapping*[5] stack to map the environment, *brown_remotelab*[6] stack to tele-operate the robot with the keyboard, and one ROS visualization tool, RVIZ[7], to show the created map.

During the simulation, the Gazebo robot sends odometry and laser scan messages to ROS, which in turn uses them to map the scenario using the *slam_gmapping* stack. The robot is controlled using keyboard input with the *brown_remotelab* stack, which sends from ROS to Gazebo a twist message (linear and angular velocities). In Fig. 5 we can see the 3D scenario in Gazebo and the map created by the *slam_gmapping* stack.

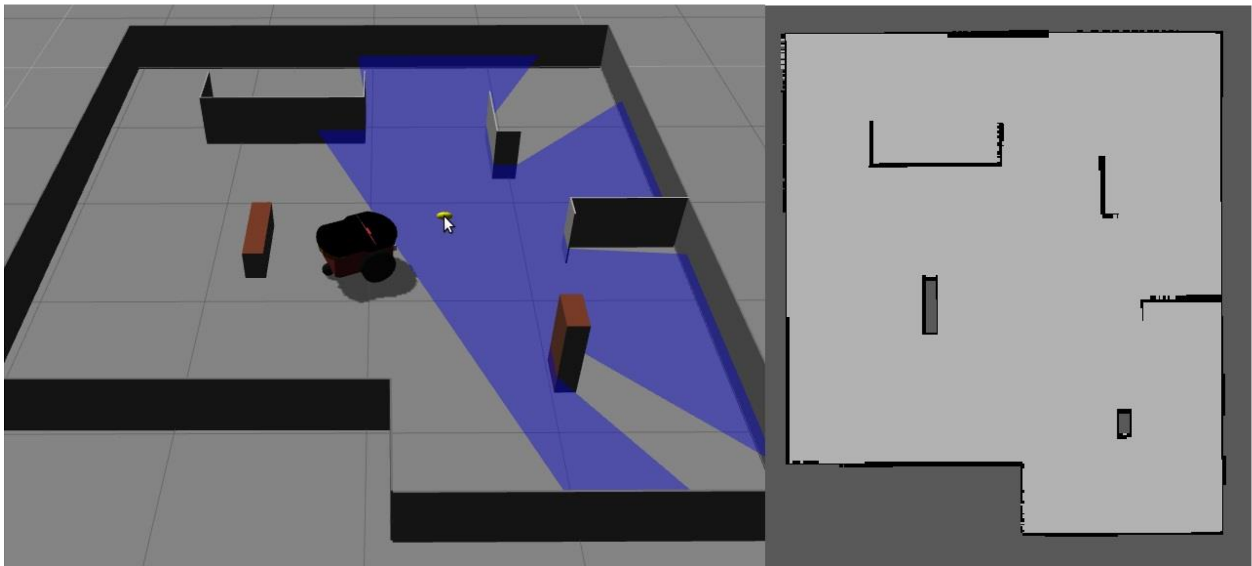


Figure 5: Virtual arena from MRL in Gazebo and the created map visualized in ROS with the RVIZ tool.

3.4 Evaluation of Gazebo's performance

To evaluate Gazebo's capabilities we simulated a SaR mission in a Gazebo world of a large basement garage. The RDPSO algorithm was used for exploration and search for victims, which are detected by voice. Both RF and voice propagations models were implemented for their use with the RDPSO algorithm, the radio frequency was used to simulate the connectivity loss between robots and the voice simulates the victims' call for help.

[5] http://www.ros.org/wiki/slam_gmapping
[6] http://ros.org/wiki/brown_remotelab
[7] http://www.ros.org/wiki/rviz

3.5 Summary

A ROS-Integrated plugin uses ROS topics to share messages with ROS nodes. These nodes can then share data messages with Gazebo, such as velocity commands or positions, through the same topics.

The ROS-Integrated plugin, described with more detail in Appendix 1, will be used to control Gazebo robots through ROS controller nodes in the simulations reported in chapter 5.

CHAPTER 4

4 - Models for Simulating Robotic Missions in Urban Firefighting Scenarios

Cooperation between Humans and rObotic teams in catastroPhic INcidents (CHOPIN) is a research and development project taking place at the Mobile Robotics Lab of (ISR-UC), in Portugal.

The CHOPIN project aims at studying the cooperation between human teams and robotic teams, including collaborative context awareness and efficient information sharing. The project exploits this human-robot symbiosis in the development of human rescuers' support systems for small scale SaR missions in urban catastrophic incidents.

As a result, the evaluation of the proposed techniques under the CHOPIN project requires the use of simulators with real-world properties, more specifically within the context of SaR scenarios [8].

4.1 Radio Frequency communication signal propagation model

The wireless communication among robots is based on with radio frequency (RF) signals. In order to simulate the signal strength in a robot, we need a propagation model for RF signals. One such model is known as the multi-wall model [16], described below. Although this model does not include the reflection on the materials, it is simple enough to be implemented while giving accurate data for the simulation.

In simulations, for the sake of simplicity, it is usual to assume that a transmitter can always communicate to a receiver. However, in real world applications there is a diminishing in the signal strength along the path between transmitter and receiver and, eventually, the received signal may be too weak (< -94 dB for most wireless equipment) to reliably share data. This path loss can be obtained by [16]:

$$L\,(d) = 10\gamma \log(d)\,, (dB)\;, \tag{1}$$

where γ is the path loss exponent and d the distance between transmitter and receiver. This path loss exponent usually has value between 2 and 4 [17], being 2 for propagation in free space (*free space loss*) and 4 for cluttered environments. For indoor environments or buildings, the

parameter γ can reach values between 4 and 6 and in some cases like tunnels, which act as waveguide, it can be less than 2.

Assuming a linear dependency between the path loss in dB and the logarithm of the distance d between the receiver and the transmitter locations and initial loss, one may define the *one-slope* model as:

$$L_{os}\,(d) = l_0 + 10\gamma \log(d)\,, (dB), \tag{2}$$

being l_0 the constant path loss at 1 meter for a given signal frequency.

The *one-slope* model (2) is simple to use but only takes into account free space loss and discards any obstacle in between the transmitter and the receiver. To include obstacles, like walls or doors, we can generalize it with the addition of a new attenuation loss for the correspondent walls, floors and/or doors penetrated by the direct path between the transmitter and the receiver [18].

$$L(d) = L_{os}(d) + M_w \;, (dB)\;, \tag{3}$$

in which this multi-wall component, M_w , is expressed as:

$$M_w = l_c + \sum_{i=1}^{I} K_{wi}\, l_i + \sum_{n=1}^{Nd} X_n\, l_d + \sum_{n=1}^{Nfd} \lambda_n\, l_{fd} \;+\; k_f^{\left(\frac{k_f+2}{k_f+1}-b\right)} L_f\;, (dB) \tag{4}$$

being l_c a constant loss; K_{wi} the number of walls of type i penetrated; l_i the loss associated to wall of type i; N_d and N_{fd} the number of doors and fireproof doors penetrated, respectively; X_n and λ_n are binary values for the door n status (0 open, 1 closed); l_d and l_{fd} the losses associated with normal doors and fireproof doors, respectively; K_f the number of floors transversed; L_f the floor loss and b an empirical value. If all experiments are carried out in the same floor, one may simply ignore the multi-floor loss, *i.e.*, K_f =0. Also the multi-wall model does not include the reflection on materials but only the refraction component.

The authors of [18] obtained the following values presented on Table 3 for the model of equations (2) and (4) for a frequency of 2.45 GHz after an experimental campaign within the office buildings of the University of Rome “Tor Vergata”.

Mw model parameter	Empirical Value (dBm)	Meaning
l_c/l_0	47.4	Constant factor loss, l_0 included
l_1	3.8	Attenuation of wall thickness [0,20] cm
l_2	3.9	Attenuation of wall thickness]20,40] cm
l_3	5.7	Attenuation of wall thickness]40,60] cm
l_4	12.4	Attenuation of wall thickness]60,80] cm
l_d	1.4	Attenuation of normal door
l_{fd}	10.2	Attenuation of fireproof door
$10\,\gamma$	23.2	Propagation exponent

Table 3: Multi-wall parameters.

With the multi-wall model and the above values, we can create a RF loss map by assuming that each cell in the map grid is a receiver. By doing so the map will represent the signal loss from the transmitter location to each cell. In Fig. 6, we can see the individual influence of each wall in the RF signal loss.

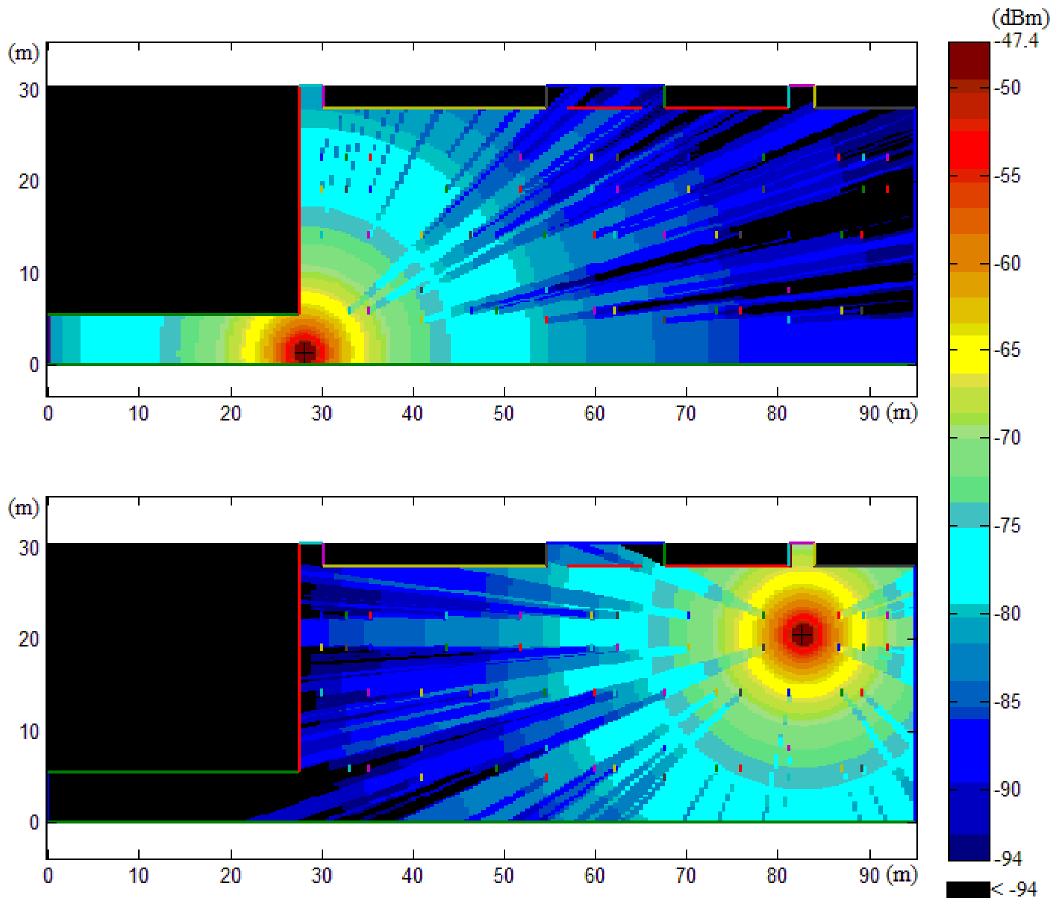


Figure 6: Radio Signal loss maps with 2 different transmitter positions.

4.2 Voice propagation model

A call for help may achieve a level between 72 and 78 dB at approximately 1 m from the source, from a very loud voice to shouting level [19], also for every doubling of the distance of the source the voice level is reduced by 6 dB. The sound level is further reduced by passing through obstacles and walls. This attenuation can be described as

$$A = A_0 e^{-\mu d} \,, \tag{5}$$

where A_0 is the attenuated amplitude of the wave before hitting an obstacle, before any obstacle is passed by. A_0 corresponds to the initial amplitude of the signal at the source, d is the distance traveled within the obstacle, such as wall thickness, and μ is the attenuation coefficient of the traveling wave, which can vary from 0.01 to 0.035 for concrete walls [19].

Voice propagation properties share similarities with other types of communication propagation, such as received power, interference, etc., including RF signal [19]. The major differences between propagations reside in intrinsic physical parameters, such as frequency wavelength and amplitude. These differences can be diminished by manipulating the parameters of the multi-wall model for the RF propagation model.

Going back to equations (2) and (3), and simplifying the multiwall component to:

$$M_w = l_c + \sum K \, l_w \,, \tag{6}$$

wherein K is the number of obstacles/walls between the transmitter and receiver and l_w is the loss associated to the obstacles and walls, we obtain:

$$L(d) = l_c + 10\gamma \log(d) + \sum K \, l_w \,. \tag{7}$$

Using equation (7), we can model the voice propagation by considering $l_w = 2\,dB$, path loss exponent γ between 5.0 and 6.0 and l_c between -78 and -72.

Based on the same method mentioned in the previous section about RF propagation, we can create a voice propagation map where each cell represents the level of voice from that cell (position of a firefighter) to the source (position of a victim), as seen in Fig. 7.

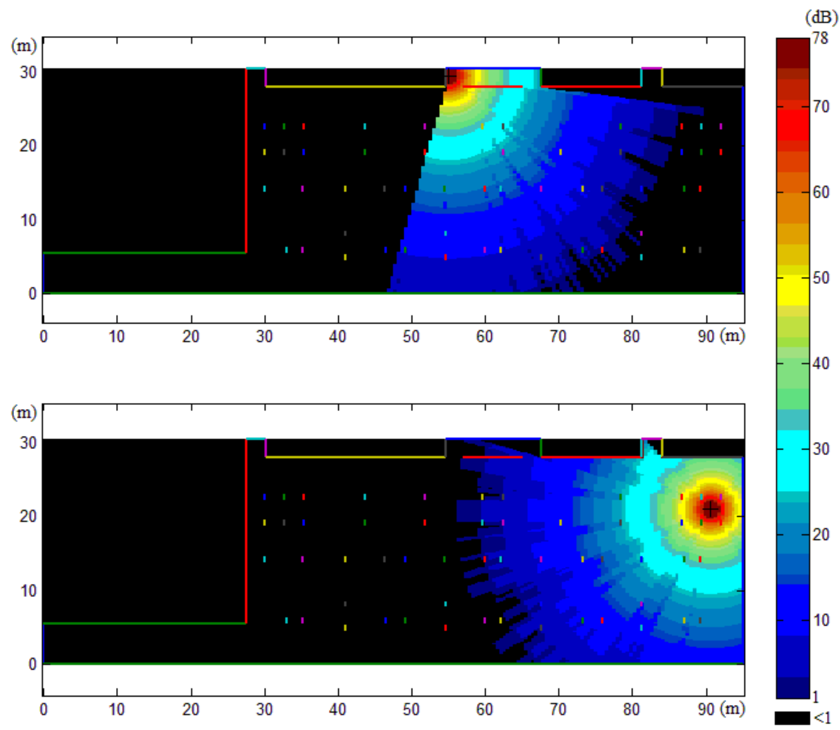


Figure 7: Voice signal strength maps from different sources, *lc* =-78 and γ = 5.

The propagation range is larger when *lc* = -78 and γ = 5 and lower for *lc* = -72 and γ = 6 as depicted in Fig. 8.

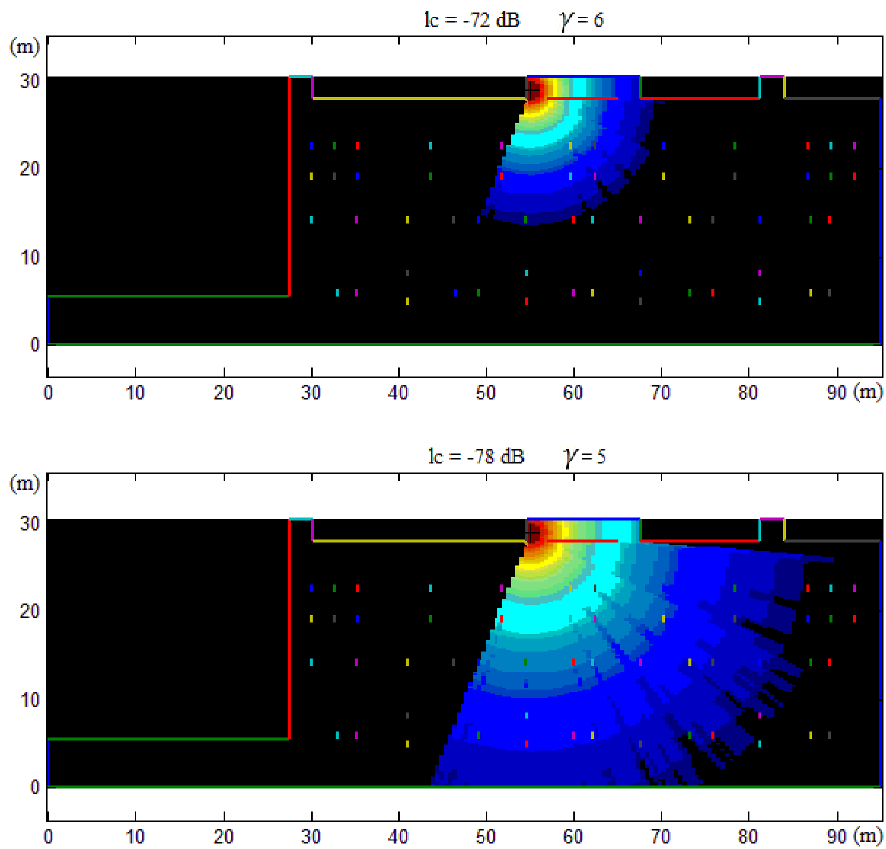


Figure 8: Comparison of propagation ranges for different voice propagation model parameters.

4.3 Implementation in ROS

Assuming the knowledge of the map beforehand, position of walls and obstacles, an occupation grid map is created. The walls position is necessary for the multi-wall model in both the RF and voice propagation models.

Using the occupation map, a voice map can be created by knowing the location of the victim (source) and by using equation (7) to calculate the voice value between the source and each free cell in the occupation map. The voice map is used as objective function, or fitness function, for the robots to search and follow. When there are multiple victims, each victim will have their lc and γ randomly chosen between [-78, -72] and [5, 6], respectively, at the start of any simulation. When creating the voice map, each cell will have the value of the highest voice signal of all victims in that cell in order to lead the robot team to the closest victim.

In addition, a constant environment/background noise is also taken into account for the voice map creation. The robot can only distinguish voice signals above this noise value, e.g., for a noise of 20 dBs only voice signals above this value are identified as voice by the robot. When creating the voice map, if a voice signal value is below the noise level threshold then it is assumed to be 0 dB for that cell.

The RF model implementation is similar to the voice one. By knowing the position of each robot in the map, we can obtain the signal loss between the robots, by directly using the RF propagation model between two positions (transmitter - receiver), this is explained in more detail in chapter 5.

4.4 Summary

The above mentioned models can be used in realistic simulations, such as virtual firefighting scenarios, in which the RF model can simulate the wireless connection status among multiple agents and the voice model can be used to simulate a victim calling for help.

On the next chapter, the RF model will be used to simulate the connectivity of a mobile ad hoc network (MANET) created within a team of robots and the voice model will be used as objective function for a victim to be found and rescued by a robot team.

CHAPTER 5

5 - Particle Optimization Algorithms for Multi-Robot Search Simulations

5.1 Particle Swarm Optimization and Darwinian Particle Swarm Optimization

In 1987, Craig Reynolds [20] proved that it is possible to program a realistic bird flock just by implementing three simple rules: match your neighbors' velocity, steer for perceived center of the flock and avoid collisions.

5.1.1 Particle Swarm Optimization

The *Particle Swarm Optimization* (PSO) proposed by James Kennedy and R.C. Eberhart in 1995 [21] is a population-based stochastic optimization method based on the social behavior of groups of organisms like bird flocks or fish schools (Fig. 9). Each individual of a given swarm is referred as a particle that flies in a given search space, wherein each of them represents a potential solution. Each movement a particle does is conditioned by not only its own search but also by the search of its neighbors. In other words, each particle's change in position, velocity, is influenced by the knowledge of its neighbors.

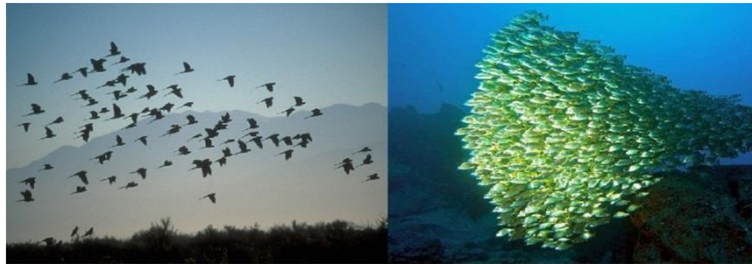


Figure 9: Flock of birds and fish school.

A given swarm is made of *n* particles, in which particle *n* is defined by a position vector, $x_n(t)$, and a velocity vector, $v_n(t)$, in the search space at time *t*. The position, $x_n(t+1)$, is given by:

$$x_n(t+1) = x_n(t) + v_n(t+1) \; . \tag{9}$$

The velocity vector, $v_n(t+1)$, of particle n is composed by several components, the current velocity, $v_n(t)$, the local best position vector, $x_{LBn}(t)$, and the global best position vector, $x_{GBn}(t)$ as equation (10) depicts.

The local best vector, known as "cognitive" component, represents the best search position of particle n in its vicinity, while the global best vector, also known as "social" component, represents the best position detected by all particles. These components can be seen in Fig. 10. The final velocity vector is given by:

$$v_n(t+1) = v_n(t) + c_1 r_1 (\, x_{LBn}(t) - x_n(t)\,) + c_2 r_2 (\, x_{GBn}(t) - x_n(t)\,)\,, \tag{10}$$

wherein r_1 and r_2 are positive random numbers between 0 and 1, and c_1 and c_2 are the weighs of the cognitive and social components, respectively.

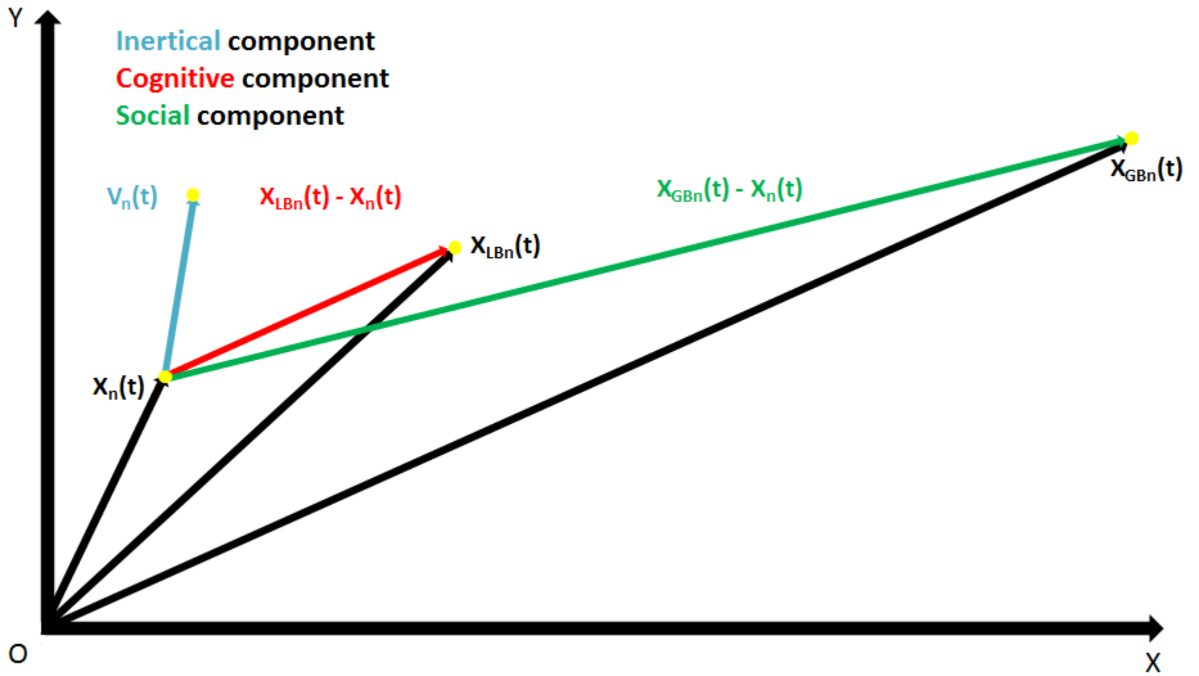


Figure 10: Cognitive and Social components in a 2D scenario.

By joining equations (9) and (10) we obtain:

$$\begin{cases} v_n(t+1) = v_n(t) + c_1 r_1 (x_{LBn}(t) - x_n(t)) \\ \qquad\qquad + c_2 r_2 \big(x_{GBn}(t) - x_n(t)\big); \\ x_n(t+1) = x_n(t) + v_n(t+1); \end{cases} \tag{11}$$

5.1.2 Darwinian Particle Swarm Optimization

A general problem of the PSO algorithm is that of becoming trapped in local optimum, being then unable to proceed to the global solution. The *Darwinian Swarm Optimization* (DPSO) was proposed in [22] as a solution for the mentioned problem by imposing a "natural selection" mechanism. In the DPSO each particle behaves like in PSO but there are some rules governing the multiple swarms.

In nature, individuals with better adaptation are more likely to live longer and procreate while individuals with worse adaptation will most likely see their life shortened.

The “natural selection” in DPSO has 3 basic assumptions:

- The longer a swarm lives, the more chance it has of producing offspring by giving each swarm a small chance of spawning a new swarm.
- A swarm has its life-time extended (reward) by finding a better state.
- A swarm has its life-time reduced (punishment) by not finding a better state.

In s given number of swarms of particles, each swarm’s solution is evaluated, if a swarm fails to improve its solution during a set number of steps then it is “punished” by deleting his worse performing particle, if then the swarm is left with a number of particles below a set minimum the whole swarm is deleted. On the other hand, whenever a swarm has achieved a new global best it will be “rewarded” by spawning a new particle, if it does not exceed the maximum number of particles per swarm.

To decide if swarm s should be “punished”, the number of steps a swarm fails to improve is tracked with a search counter, SC_s. If this counter exceeds a maximum threshold, SC_s^{max}, a particle of the swarm is deleted. When swarm s is created it has its search counter set to 0, when a particle of swarm s is deleted its search counter is set to a value tending towards SC_s^{max}, this value is obtained with:

$$SC_s(N_{kill}) = SC_s^{max} \left[1 - \frac{1}{N_{kill}+1}\right], \qquad (12)$$

wherein N_{kill} is the number of particles deleted from the swarm during the period it fails to improve its solution. The objective of (12) is to improve the DPSO by deleting the less adaptive swarms faster and keep the better ones.

At any time, each swarm has a chance of spawning a new swarm if and only if $N_{kill} = 0$ and if the maximum number of swarms is not exceeded. The probability of spawning a swarm is given by:

$$p = \frac{f}{N_s}, \qquad (13)$$

wherein f is a uniform random number in [0, 1] and N_s is the number of swarms.

To spawn the child swarm, half of its particles are randomly selected from the parent swarm and the other half are randomly selected from a random swarm (mate) of the group of swarms.

5.2 Robotic PSO and Robotic Darwinian PSO

One of the first adaptations of the PSO algorithm to handle obstacle avoidance for robots was presented in [23], wherein the authors adjust the velocity and direction of the robot to avoid obstacles in real time by complementing equation (11) with the inertia influence weight, w, assigned to the velocity, $v_n[t]$, thus resulting in equation (14):

$$\begin{cases} v_n[t+1] = wv_n[t] + c_1r_1(x_{LBn}[t] - x_n[t]) \\ \qquad\qquad\qquad + c_2r_2(x_{GBn}[t] - x_n[t]), \\ x_n[t+1] = x_n[t] + v_n[t+1], \end{cases} \tag{14}$$

5.2.1 Robotic Particle Swarm Optimization

The *Robotic Particle Swarm Optimization* (RPSO) presented by Couceiro *et al.* in [4], like the PSO algorithm, consists on a swarm of robots that move in an obstacle populated search space in search for a global optimum. The RPSO assumes that each robot is equipped with sensors capable of detecting obstacles in their vicinity within a finite sensing radius r_s. By doing so, a sensing function, $g(x_n[t])$, can be created to represent the distance of the robot to nearby obstacles. As such, equation (6) can be extended with an “obstacle avoidance” component:

$$\begin{cases} v_n[t+1] = wv_n[t] + c_1r_1(x_{LBn}[t] - x_n[t]) \\ \qquad\qquad\qquad + c_2r_2(x_{GBn}[t] - x_n[t]) \\ \qquad\qquad\qquad + c_3r_3(x_{OBn}[t] - x_n[t]); \\ x_n[t+1] = x_n[t] + v_n[t+1]; \end{cases} \tag{15}$$

wherein r_3 is a positive random number between 0 and 1, c_3 is the weight associated to the “obstacle avoidance” component and $x_{OBn}[t]$ is the obstacle avoidance best position vector that optimizes the sensing function, $g(x_n[t])$.

Unfortunately, the RPSO inherits a major problem from the PSO: having the chance of being stuck in local minima or local maxima. To that end, the same authors in [4] presented an evolutionary alternative.

5.2.2 Robotic Darwinian Particle Swarm Optimization

The *Robotic Darwinian Particle Swarm Optimization* (RDPSO), also presented in [4], is a robotic adaptation of the DPSO to multi-robot systems in order to address the common drawback of PSO and RPSO mentioned above. The RDPSO is very much like the RPSO but extended with the “natural selection” concept of the DPSO algorithm, granting it the ability to better escape local optima.

The RDPSO and the DPSO mainly differ in 3 aspects:

- Obstacle avoidance
- Social exclusion and social inclusion:
- Ensuring MANET connectivity

The obstacle avoidance aspect of the RDPSO is identical to obstacle avoidance of the RPSO, in which a sensing function, $g(x_n[t])$, is used to obtain the obstacle avoidance best position vector, $x_{OBn}[t]$.

The *social exclusion* and *social inclusion* are the equivalent of punishing and rewarding actions of the DPSO's "natural selection" concept.

Social exclusion, in order to adapt the concept of "punishment" to mobile robot systems, instead of deleting a particle (i.e., robot) from a swarm, the worst performing robot is excluded from the punished swarm and added to a socially excluded group. These socially excluded robots do not search for the global optimum, like the ones in the active swarms, but instead wander randomly in the search space.

Social Inclusion, during the "rewarding" event of a swarm, instead of spawning a new particle, the best performing robot of the socially excluded group is added to the rewarded swarm, only if there are any robots in the socially excluded group and if the number of robots of the swarm does not exceed a predefined maximum number of robots per swarm.

To ensure the MANET connectivity, i.e. to maintain the communication within the robot swarm, the robot's position must fulfill a given communication quality constrain, such as maximum distance or minimum signal quality. The MANET's connectivity algorithm, that influences the robots' position, will be described in the next section.

Ensuring MANET connectivity

On mobile robotic systems, it is important to maintain the wireless communication within a swarm. To achieve this, the position of the robot must be controlled with respect to constrains like maximum distance or minimum signal quality. In a scenario without a reliable communication infrastructure it is necessary that each robot acts like a node/router for the network in order to support multi-hop communication in a MANET by relaying information from one robot to another.

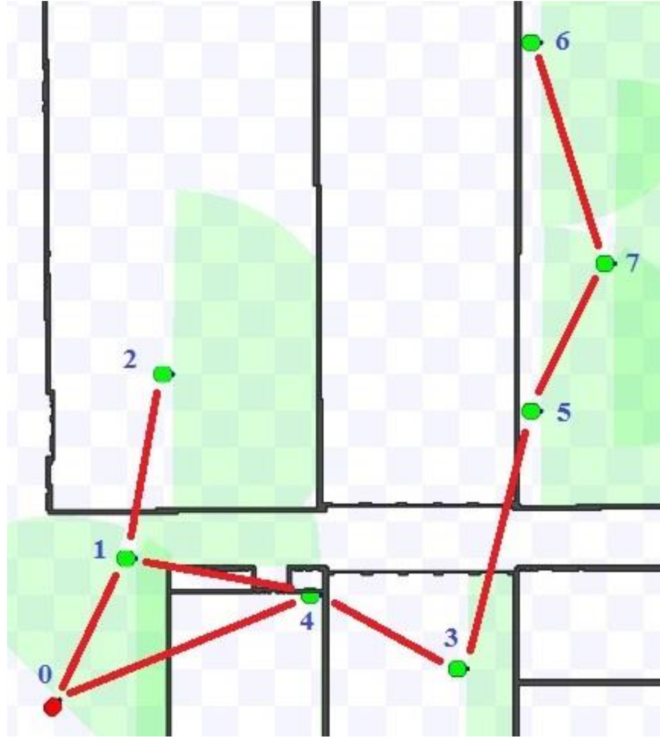


Figure 11: Example of the topology of a MANET due to signal quality loss in an office-like scenario.

As seen in Fig. 11, if robot 0 wants to send data to robot 6, because it is not able to communicate directly, it will have to use robots 4, 3, 5 and 7, acting like forwarding routers, as intermediary nodes of the multi-hop network, for a minimum number of 5 hops, as seen later.

According to the algorithm presented in by Couceiro *et al.* [24], let us consider a swarm of N robots where each robot is simultaneously an exploring agent and a mobile node of the MANET performing multi-hop package forwarding.

The *Link matrix* $L = \{l_{ij}\} - N \times N$ matrix describes the connectivity between robots, wherein each l_{ij} entry represents the link between robot i and j. These values are defined according to one of the following approaches:

1- Calculating l_{ij} values as functions of the distance between pairs of nodes, *link distance*.

2- Calculating l_{ij} values as functions of the radio frequency (RF) signal between pairs of nodes, *link quality*.

On either of the above approaches, for $i = j$ the value of $l_{ij} = 0$.

Only taking into account the communication range d_{max} to maintain the network connectivity (approach 1) is not very realistic due to the propagation model. It is more complex because the signal does not depend only of the distance but also from the multiple paths which have losses introduced by walls or other obstacles in between the robots (approach 2). Assuming f_{ij} as the value that describes the link between robot i and j according to the chosen approach:

$$l_{ij} = \begin{cases} f_{ij}, & if\ i \neq j, \\ 0, & if\ i = j, \end{cases} \tag{16}$$

The *Adjacency matrix* $A = \{a_{ij}\} - N \times N$ matrix that represents the one-hop connectivity between robot *i* and *j*. Depending on the which of the above approaches is chosen, the value of the entry a_{ij} is 1 if there is direct connection between robot *i* and *j* and 0 if there is not. The diagonal of A, i.e. $i = j$, is always 0 (eq.17).

$$a_{ij} = \begin{cases} 1, & connection\ between\ i\ and\ j, \\ 0, & no\ connection\ between\ i\ and\ j. \end{cases} \tag{17}$$

Assuming the network system supports multi-hop, and using the non-diagonal 0 values of the *adjacency matrix* A, we can obtain the smallest number of hops needed to connect two nodes.

The *Connectivity matrix* $C^{(k)} = \{c_{ij}{}^{(k)}\} - N \times N$ matrix wherein each, $c_{ij}{}^{(k)}$, entry represents the minimum number of hops needed to connect robot i to robot *j*, *k* represents the iteration of $C^{(k)}$ creation, max $k = N - 1$, and *h* is the number of hops, *h* ε [1 , *k*]:

$$c_{ij}^{(k)} = \begin{cases} h, & node\ i\ connected\ to\ j\ by\ h\ hops, \\ 0, & no\ connection\ possible\ between\ i\ and\ j. \end{cases} \tag{18}$$

Note that the diagonal values of the connectivity matrix ($i = j$) are always equal to 0. For the first iteration, the connectivity matrix assumes the values of the adjacency matrix ($C^{(1)} = A$). For $k > 1$, i.e. for multi-hop connections, an *auxiliary matrix* $B^{(k)} = \{b_{ij}{}^{(k)}\}$ is calculated based on the number of hops (iteration *k*) and adjacency and connectivity matrixes:

$$b_{ij}^{(k)} = \begin{cases} 0, & c_{ij}^{(k-1)} > 0, \\ k, & \sum_{t=1}^{N} c_{it}^{(k-1)} . a_{tj} > 0\ and\ c_{ij}^{(k-1)} = 0. \end{cases} \tag{19}$$

Like before, the diagonal elements of the auxiliary matrix are always equal to 0, $B^{(1)}$ is a matrix of zeros. The connectivity matrix can then be calculated by equation (20):

$$C^{(k)} = C^{(k-1)} + B^{(k)}. \tag{20}$$

After *N-1* iterations, the connectivity matrix $C^{(N-1)}$ represents the multi-hop connectivity of the network. In the example of Fig. 11, for the connection between robot 0 and robot 6, the entries $c_{06}{}^{(N-1)}$ and $c_{60}{}^{(N-1)}$ of the connectivity matrix, $C^{(N-1))}$, would be 5 hops, as seen in Table 5 below.

ROBOT	0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0	0
2	0	1	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0
4	1	1	0	1	0	0	0	0
5	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	1	1	0

Table 4: Adjacency matrix A of the scenario in Fig. 11.

ROBOT	0	1	2	3	4	5	6	7
0	0	1	2	2	1	3	5	4
1	1	0	1	2	1	3	5	4
2	2	1	0	3	2	4	6	5
3	2	2	3	0	1	1	3	2
4	1	1	2	1	0	2	4	3
5	3	3	4	1	2	0	2	1
6	5	5	6	3	4	2	0	1
7	4	4	5	2	3	1	1	0

Table 5: Connectivity matrix $C^{(7)}$ of the scenario in Fig. 11.

If $c_{ij}{}^{(N-1)} = 0$, for $i \neq j$, then it is not possible for robot *i* to connect with robot *j* even using multi-hop. If this case happens, a *binary connectivity matrix* $C_B = \{c_{Bij}\}$, in which each entry c_{Bij} is equal to 1 if $c_{ij}{}^{(N-1)} \neq 0$ and equal to 0 otherwise, as seen in equation (21),

$$c_{Bij} = \begin{cases} 1, & c_{ij}^{(k-1)} \neq 0, \\ 0, & c_{ij}^{(k-1)} = 0, \end{cases} \tag{21}$$

and by multiplying element-by-element the link matrix and the logical inverse (binary NOT) of matrix C_B we obtain the *break matrix* $C_{break} = \{c_{breakij}\}$, wherein each value represents the break of connection between nodes.

The matrix $C^{(N-1)}$ and the auxiliary matrixes C_B and C_{break} are used as information about the network topology.

If a break in the network's connectivity is detected then the robot's position $x[n+1]$ of the RPSO and RDPSO algorithms must be influenced in order to restore the MANET connectivity. To do so we need to extend equation (15) with a new "communication" component:

$$\begin{cases} v_n[t+1] = wv_n[t] + c_1r_1(x_{LBn}[t] - x_n[t]) \\ \qquad + c_2r_2(x_{GBn}[t] - x_n[t]) \\ \qquad + c_3r_3(x_{OBn}[t] - x_n[t]) \\ \qquad + c_4r_4(x_{CBn}[t] - x_n[t]), \\ x_n[t+1] = x_n[t] + v_n[t+1], \end{cases} \tag{22}$$

wherein r_4 is a random positive number, c_4 is the weight of the "communication" component and $x_{CBn}[t]$ is the communication best position vector.

The communication best position vector, $x_{CBn}[t]$, is the position of the nearest neighbor of robot *n* increased by the maximum communication range, d_{max}, toward the robot's current position.

Besides considering the communication between robots, one also needs to consider robots' dynamics. To that end, one can resort to fractional calculus concepts.

Fractional order convergence

As explained in [25] and demonstrated in [26], equation (22) can be further extended by improving the PSO's inertial influence component with the concept of fractional calculus (FC). One of the most common approaches based on this concept is the discrete time *Grünwald-Letnikov* definition given by the following equation:

$$D^{\alpha}\left[v_n[t+1]\right] = \frac{1}{T^{\alpha}} \sum_{k=0}^{r} \frac{(-1)^k \alpha(\alpha+1) v_n[t+1-kT]}{\Gamma(k+1)\Gamma(\alpha-k+1)} \;, \tag{23}$$

in which T is the sampling period, r is truncation order that manages the memory complexity of the algorithm, α is the fractional coefficient that weights the relevance of past events.
From the equations (22) and (23), considering $T = 1$ and $r = 4$, the inertial component previously defined as $wvn[t]$ of robot n, is now defined as:

$$w_n[t] = \alpha v_n[t] + \frac{1}{2}\alpha v_n[t-1] + \frac{1}{6}\alpha(1-\alpha)v_n[t-2] + \frac{1}{24}\alpha(1-\alpha)(2-\alpha)v_n[t-3]. \quad (24)$$

With equations (22) and (24) we obtain:

$$\begin{cases} v_n[t+1] = \alpha v_n[t] + \frac{1}{2}\alpha v_n[t-1] + \frac{1}{6}\alpha(1-\alpha)v_n[t-2] \\ \qquad + \frac{1}{24}\alpha(1-\alpha)(2-\alpha)v_n[t-3] \\ \qquad + c_1 r_1 (x_{LBn}[t] - x_n[t]) \\ \qquad + c_2 r_2 (x_{GBn}[t] - x_n[t]) \\ \qquad + c_3 r_3 (x_{OBn}[t] - x_n[t]) \\ \qquad + c_4 r_4 (x_{CBn}[t] - x_n[t]), \\ x_n[t+1] = x_n[t] + v_n[t+1]. \end{cases} \quad (25)$$

Context based Evaluation Metrics for RDPSO

In order to improve the performance of the RDPSO algorithm for different scenarios it is necessary to adjust the weights of each component c_l, $l = 1,2,3,4$, as the mission progresses. This adaptability will help optimize the swarm's convergence rate when faced with different environments. To achieve this goal, a set of evaluation metrics are proposed in [27] by Couceiro *et al.* in order to measure the swarm's behavior facing the obstacle and communication constrains, as seen below.
Also, it is important to note that, although adjusting each parameter of the RDPSO can lead to better convergence, it is the combination of all of them that determines the convergence properties and the best performance is achieved, and always abiding by the following restriction:

$$c_T + c_3 + c_4 = 2\ , \quad (26)$$

in which:

$$c_T = c_1 + c_2\ . \quad (27)$$

Exploration vs exploitation

Exploration – swarm behavior related to the algorithm's diversification, allowing the exploration of new solutions and better avoidance of local optima, however if the exploration's level is too

high then the time needed for the algorithm to find the global optima may be too long, reducing its desired effectiveness.

Exploitation – swarm behavior related to the algorithm's convergence, unlike the *exploration*, a high level of exploitation will lead to a faster global solution but will also make the algorithm more susceptible to getting stuck in local optima.

To address this unbalance between both behaviors, a common solution is to adjust the inertia weight systematically. Higher inertia weight means better exploration behavior while lower inertia weight will lead to better exploitation.

By using the fractional calculus (FC) strategy to control the convergence of the swarm, adjusting systematically the fractional coefficient, α, will provide a higher level of exploration while ensuring the optimal solution of the mission is not compromised.

To adjust α in order to obtain a smooth transition between behaviors we need to have knowledge of the swarm's current activity. Let us define the *swarm activity*, $A_s[t]$, of a given swarm *s* as the norm of the swarm *s* center of mass velocity, $v_s[t]$:

$$A_s[t] = \frac{\|v_s[t]\|}{v_{max}}, \tag{28}$$

wherein v_{max} is the maximum step between iterations. In (28) we analyze the activity of the swarm as a whole, each robot of the swarm may have an high activity while the swarm's activity is low.

When $A_s[t] = 0$, the swarm has no activity at all and therefore α should increase. On the other hand, when $A_s[t] = 1$ the swarm presents a high chaotic activity and α should decrease.

Although this adjustment helps control the swarm as a whole, each individual robot behavior is also important and their own cognition and socialization levels must also be taking into account.

Cognition vs socialization

Cognition – is related with the cognitive component, c_1, and represents the individual decision of each robot to follow their own best solution, *local best solution*.

Socialization – is related with the social component, c_2, this influences the robots movement towards the best solution found so far by the swarm, *global best solution*.

High coefficient c_2 will summon the robots to the global optimal solution faster, acting as "blind" followers, but at the risk of prematurely trapping the swarm in local optima. Also, having high coefficient c_1 may lead each robot to its local best solution to much and promote excessive

wandering, increasing the mission time. As mentioned in [27], a smaller c_1 and a bigger c_2 can significantly improve the algorithm's performance.

Although the cognitive and social component weights, c_1 and c_2, are not critical for the mission success, proper adjustment of their values will lead to a better performance in speed of convergence and sub-optimal solution avoidance.
In order to decide whether to increase or decrease the weight values the *robot socialization* metric is created, which is defined as the Euclidean distance of robot *n* to the swarm *s* global best robot:

$$S_n[t] = 1 - \frac{\|XGB_s[t] - x_n[t]\|}{\|XGB_s[t] - \max x_n[t]\|}, \quad (29)$$

wherein $XGB_s[t]$ is the position vector of the robot with the best solution found so far by the swarm *s*, $\max x_n[t]$ and $x_n[t]$ is the position vector of the robot *n*. A robot with a social level of $S_n[t] = 0$ means it is the farthest robot of the swarm to the global best, $x_n[t] = \max x_n[t]$, and therefore c_2 should be high. On the other hand if the social level is $S_n[t] = 1$ then robot *n* is the one sensing the global best of the swarm, $x_n[t] = XGB_s[t]$, and c_2 should be low.

Obstacle susceptibility

The efficiency the RDPSO in search missions is greatly influenced by the obstacle density. The obstacle avoidance component of the RDPSO helps avoid collisions but at the cost of convergence time, to increase the algorithm performance the robot needs to be able to adjust the weigh c_3 accordingly to its surroundings.
If no obstacles are detected inside the sensing radius r_s then c_3 will be 0, this will allow the algorithm to "focus" more in the convergence to the global optima by allowing a wider range of values of the other coefficients in (27) while always abiding by (26). On the other hand, when faced with obstacles in its vicinity c_3 will have to increase as the distance to the obstacle lowers, this will allow the robot to better avoid the obstacles in its path.
To accomplish this intelligent adaptation we need a metric to evaluate the surroundings of the robot, thus the *robot avoidance* was defined:

$$O_n[t] = \frac{r_s - g(x_n[t])}{r_s}, \quad (30)$$

wherein the sensing function $g(x_n[t])$ will return r_s when no obstacle is sensed in the surroundings of robot n, so in an obstacle-free path $O_n[t] = 0$ and the obstacle avoidance component can be neglected, $c_3 = 0$. On the other hand as the robot nears an obstacle $O_n[t]$ tends towards 1 and c_3 will have to increase accordingly.

Connectivity susceptibility

In mobile robot missions, the wireless network has a vital role allowing the robot team to share information about the search space (victims, dangerous areas, dead ends, etc.). To maintain a mobile ad hoc network (MANET) within the swarm, each robot will have to plan its movement in order for the MANET to be maintained, thus a maximum distance, d_{max} or minimum signal quality, q_{min}, between robots must be kept while still being able to spread as much as possible. Considering the d_{max} constraint, the *robot proximity* can be defined as:

$$P_n[t] = \begin{cases} 1 - \frac{d_{nm}[t]}{d_{max}}, & d_{nm}[t] \leq d_{max} \\ 0, & d_{nm}[t] > d_{max} \end{cases}, \tag{31}$$

wherein $d_{nm}[t]$ is the distance between robot n and its nearest neighbor m. Likewise, if we consider the q_{min}, constrain then:

$$P_n[t] = \begin{cases} 1 - \frac{q_{min}}{q_{nm}[t]}, & q_{nm}[t] \geq q_{min} \\ 0, & q_{nm}[t] < q_{min} \end{cases}, \tag{32}$$

wherein $q_{nm}[t]$ is the minimum signal quality between robot n and its nearest neighbor m.

In real situations and to ensure the MANET connectivity, equation (31) based on the maximum distance is not the best approach as the radio frequency (RF) model is not as simple, the signal loss does not depend only on the distance but also on the walls and obstacles in between the robots.

The robot proximity, $P_n[t]$, measures how close robot n is to the threshold, d_{max} or q_{min}, defined for communication to be successful with its nearest robot.

Using only relations between pairs of robots helps keeping the MANET locally connected, but to ensure the global MANET connectivity we need a new metric. As presented by Nathan *et al.* in [28], the graph connectivity of the swarm's MANET can be represented by *Fiedler value*, λ_2, the second smallest eigenvalue of the *Laplacian matrix Lp* defined by:

$$Lp = \Delta - A, \tag{33}$$

wherein Δ is the *valency matrix*, diagonal matrix with the node *n* degree (number of direct connections to robot *n*) in entry Δ_{nn}, and A is the adjacency matrix.

If the *Fiedler value* is greater than zero, $\lambda_2>0$, the graph is connected, if not then it means at least one link of the graph is "broken". The *swarm connectivity* evaluates the connection of the graph and is defined as:

$$c_s[t] = \begin{cases} \frac{\lambda_2}{N_s}, & \lambda_2 \geq 0 \\ 0, & \lambda_2 < 0 \end{cases} , \tag{34}$$

wherein N_s is the number of robots in swarm *s*. When all robots of the swarm are directly connected then $\lambda_2 = N_s$ and $C_s[t]=1$, meaning a fully connect swarm. As $C_s[t]$ decreases towards 0, c4 should increase to ensure the MANET connectivity.

Based on the contextual information provided by the above metrics we can control the robots behavior by adjusting the RDPSO's weighs according to the following rules, presented in [27]:

IF $A_s[t]$ **IS** *Active* **THEN** α **IS** *Small*

  ELSE α **IS** *Large*

IF $O_n[t]$ **IS** *Close* **OR** $P_n[t]$ **IS** *Far* **OR** $C_s[t]$ **IS NOT** *Connected*

  THEN α **IS** *Nominal*

IF $S_n[t]$ **IS** Social **OR** $O_n[t]$ **IS** *Close* **OR** $P_n[t]$ **IS** *Far* **OR** $C_s[t]$ **IS NOT** *Connected*

  THEN c_2 **IS** *Small*

  ELSE c_2 **IS** *Large*

IF $O_n[t]$ **IS** *Close*

  THEN c_3 **IS** *Large*

  ELSE – IF $P_n[t]$ **IS** *Far* **OR** $C_s[t]$ **IS NOT** *Connected*

    THEN c_3 **IS** *Small*

IF $P_n[t]$ **IS** *Far* **OR** $C_s[t]$ **IS NOT** *Connected*

  THEN c_4 **IS** *Large*

  ELSE – IF $O_n[t]$ **IS** *Close*

    THEN c_4 **IS** *Small*

5.3 Algorithm Simulation

Unfortunately, due to a bug already recognized by the developers, the standalone Gazebo can only run one plugin at a time, meaning that only one robot can be controlled. Although this is an ongoing work by Gazebo developers, they estimated its solution to become available only by the end of the summer of 2013 when this dissertation will have already been defended. To address this problem, we decided to use Stage in addition to Gazebo to run the simulations. Since both are compatible with ROS, both the controllers and the plugin were adjusted in order for the same controller to work on both Gazebo and Stage without distinction. In a simulation of three robots, the first two will be stage robots and the last one will be a Gazebo robot. In the future, to replace the Stage robots with multiple Gazebo ones, simply copy the plugin in Appendix 1 to a new file, change the process name from “robot_0” to whichever ID wanted (robot_1, robot_2,…, robot_N) and attach the plugin to the respective robot model in the world file, Appendix 2.

With the above problem addressed, we can step to the simulation. Let us first briefly explain the simulation as a whole. To simulate the previously described models and algorithms, a SaR scenario in a large basement garage is used, Figures 12 and 13. In this scenario a team of 3 robots will explore given areas of the map, while moving to the target area, the robot is receiving sensor data from a “sensor world”, if voice is detected then the team’s movement will converge in the direction of the highest voice signal detected.

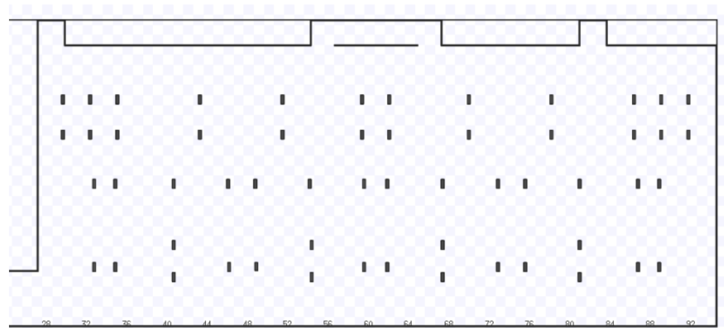


Figure 12: DEEC garage in stage (2D)

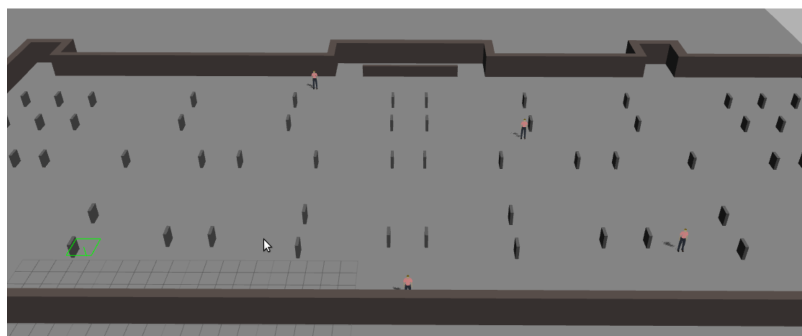


Figure 13: DEEC garage simulated in Gazebo (3D)

Once the victim's position is successfully located they are considered to be saved. The voice map is recreated without saved victims and then the team will resume their movement towards the previously designated area to explore until a new voice signal is detected. Each area to explore is considered a circle and defined by the circle's center position and radius.

Each robot's behavior, during the iterations, is described in Figure 14. A "social included" will pursue the mission objectives using the RDPSO algorithm. While a "social excluded" robot will wander randomly in the map, choosing a random direction to move until the simulation has ended.

The sensor world uses each robot's world position (odometry) to create sensorial data. This data includes the robot N local best position, the value for that local best, the distance between robot N and each victim and its distance to the center of the area to be explored.

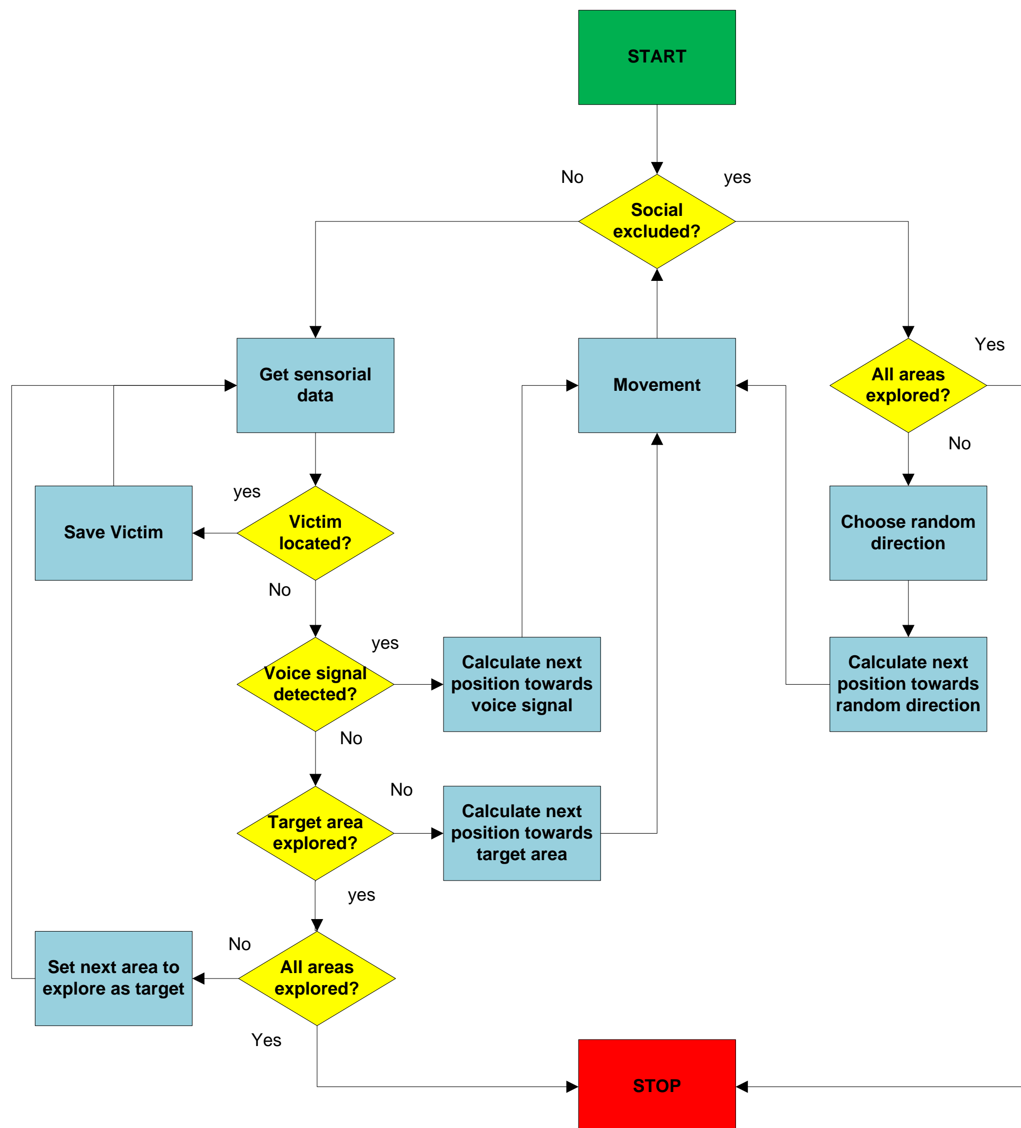


Figure 14: Social included and excluded robot's behavior during the SaR simulation.

The robot's local best will be the position of the cell with the highest voice level in its surroundings or, if no voice is detected, the cell whose position is closest to the area to explore. The local best positions and their values are shared between all robots in order for each robot to calculate the RDPSO's global best position. Each robot calculates their next position through the RDPSO algorithm.

The distances given by the sensor world are used for the actions of locating victims and exploring areas:

- If the distance to a victim is below a set value (1m), then that victim is considered saved, saving a victim removes her from the voice map.
- If a robot is inside the target area's circle, *i.e.*, the distance of the robot to the area's center is lower than its radius (2m), then that area is considered explored.

Conditions for the simulation to stop:

1. All areas are explored.
2. The time limit is exceeded (600 seconds).
3. All robots are socially excluded, *i.e.*, all swarms have been deleted (disbanded).

Simulation set-up and parameters:

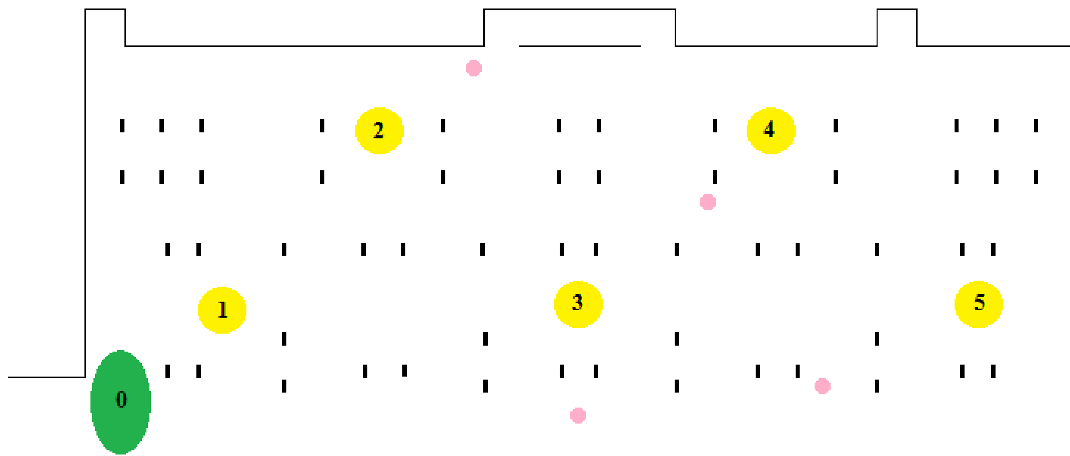


Figure 15: Simulation set-up configuration, waypoints and victims.

For the following simulations, we will use Fig. 15 configuration. We chose 5 arbitrary waypoints (yellow circles) and 4 stationary victims (pink circles). The robots are deployed at location 0 (green circle). The robots start at location 0 and explore the areas 1, 2, 3, 4 and 5 in this order.

At the start of each simulation, each victim will have their own voice model parameters lc and γ with values chosen randomly between [-78, -72] and [5, 6], respectively. As previously

mentioned, the lowest range of voice propagation is achieved for lc = -72 and γ = 6 (Fig. 16) and the biggest range for lc = -78 and γ = 5 (Fig. 17).

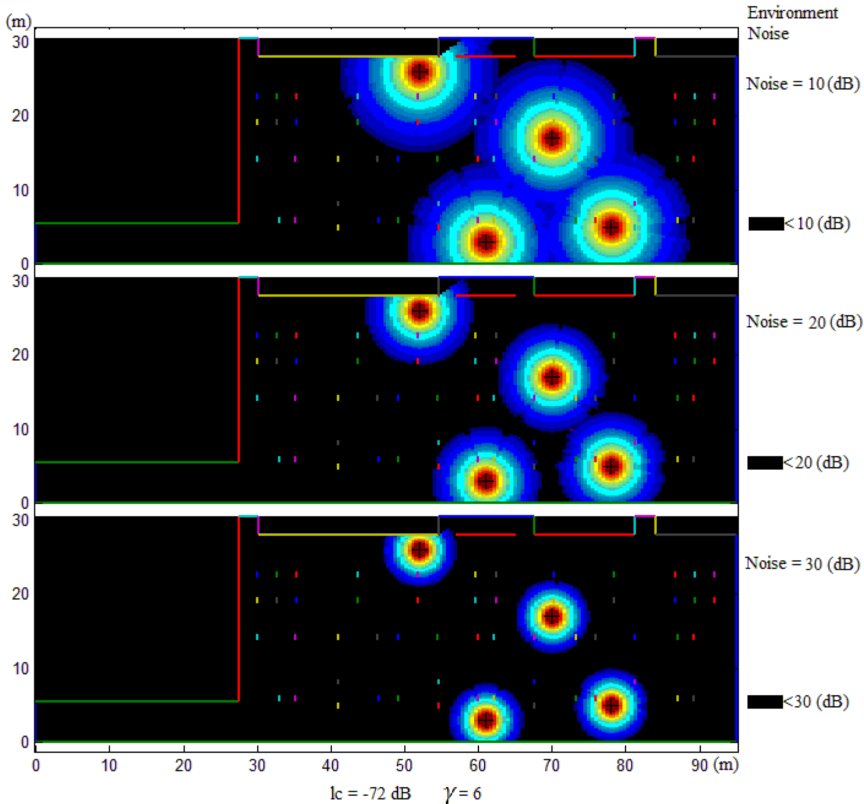


Figure 16: Lowest voice detection range for different background noise levels with 4 victims.

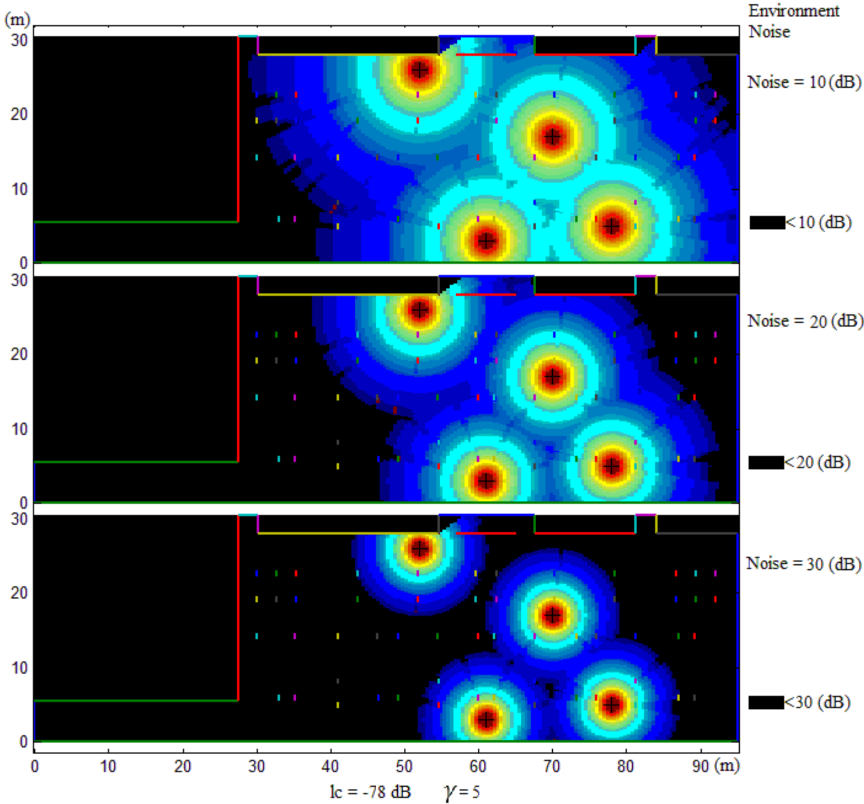


Figure 17: Highest voice detection range for different background noise levels with 4 victims.

In the above Figures 16 and 17, we can see the influence of the background noise levels in the detection range of the victims. In the following simulations we will have several sets of trials with different levels of noise (10, 20 and 30 dB). We can foresee that some of the victims will be harder to locate when the noise levels are higher.

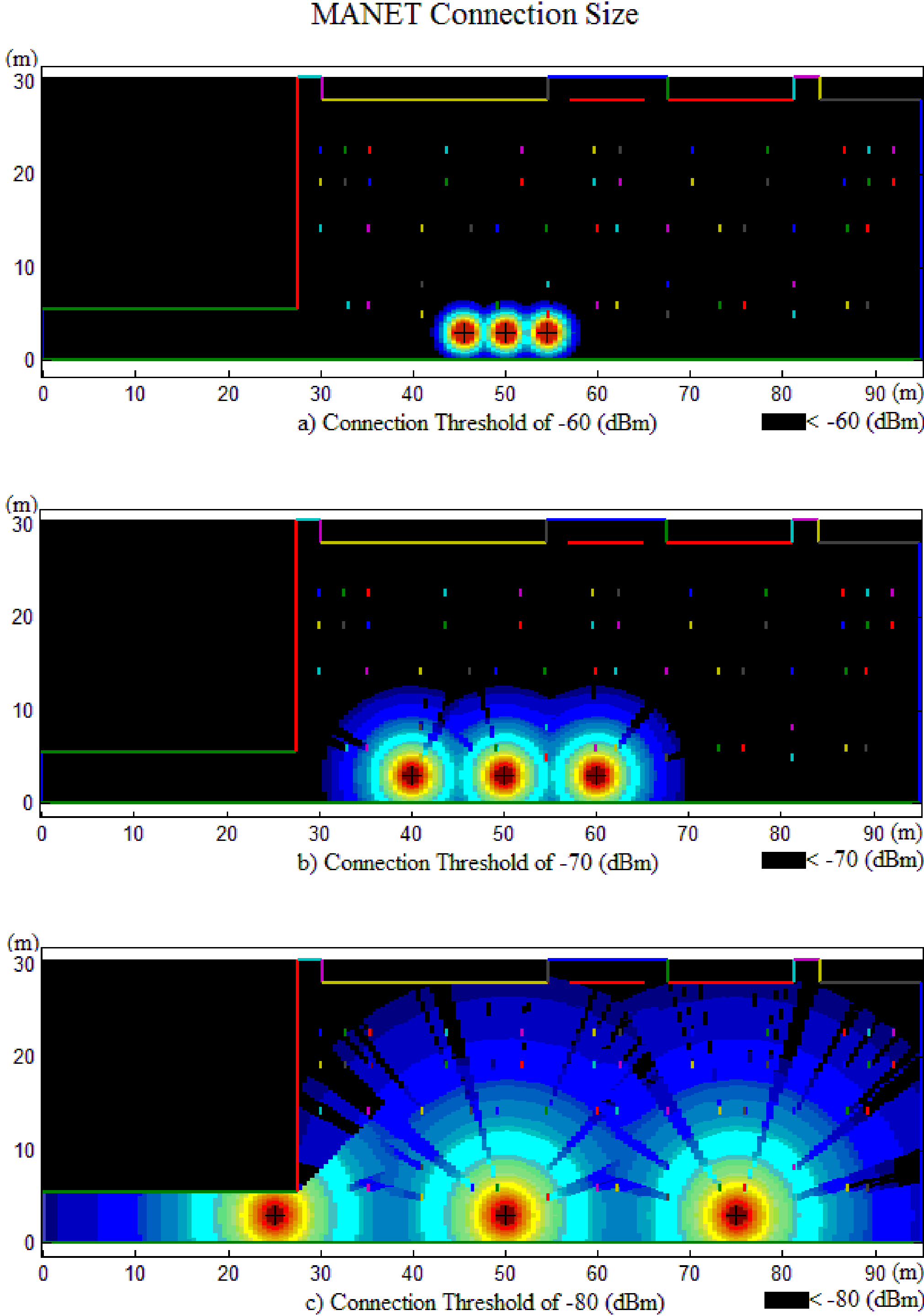


Figure 18: MANET size for different connection thresholds for a team of three robots.

In Fig. 18, we can compare the MANET connection size for different connection threshold values of the RDPSO algorithm. For a threshold of -60 (dBm) the range that each robot can communicate is approximately 4.5 meters, increasing to 10 meters for a value of -70 (dBm) and 25 meters for -80 (dBm).

It is very important to choose an adequate value for the threshold as it will influence how much the robot team can spread. Choosing values above -70 (dBm) can restrict the movement of the robots as they will always try to keep a maximum distance of less than 10 meters from its nearest neighbor, decreasing the area of exploration of the collective team and, thus, increasing the mission duration. The signal sensitivity for unreliable data transfer is -94 (dBm), in such a way that a signal loss between -80 and -94 (dBm) represents the danger zone for communication loss in which the RDPSO algorithm strongly influences the robots to move close to each other before the MANET is broken. Choosing values below -80 (dBm) can prove to be harmful for the mission as a simple turn of a corner can completely cut off a robot from the team and break the MANET. As such a value between -80 and -70 (dBm) should be chosen to improve the exploration while not endangering the MANET connection.

For the following simulations, the MANET communication threshold is defined as -80 (dBm) (approximately 25 meters in free space) to provide a good spreading size while maintaining the MANET's connection.

5.4 Results and Discussion

5.4.1 Results

A total of 90 simulations were conducted; 30 for each of the different environment noise levels, 10, 20 and 30 (dB). In each simulation trial, we acquired the simulation duration, the number of victims saved, the times the MANET connection was in danger of being broken and the number of way points explored.

In Table 6, we have the number of victims saved and the simulation time for the 90 trials. We can see an obvious decrease in rescue rate with increased the noise level.

The time limit for each trial, set to 600 seconds (10min), was only achieved during 2 trials (Table 6) during which all victims were still saved.

	Environment Noise Level					
	10 (dB)		20 (dB)		30 (dB)	
TRIAL	**Victims saved**	**Time (sec)**	**Victims saved**	**Time (sec)**	**Victims saved**	**Time (sec)**
1	4	426	2	333	2	324
2	4	438	4	376	2	372
3	4	447	2	350	2	327
4	4	451	3	335	2	375
5	4	491	2	348	2	338
6	4	436	3	361	2	322
7	4	447	3	373	2	341
8	4	496	4	448	3	380
9	4	600	2	348	2	352
10	4	402	3	352	2	320
11	4	418	4	428	2	335
12	4	447	3	340	2	336
13	4	413	3	346	2	386
14	4	444	4	457	2	310
15	4	600	4	417	2	328
16	4	435	2	327	2	348
17	4	586	2	323	2	358
18	4	427	4	434	2	346
19	4	509	4	407	2	338
20	4	427	3	337	2	327
21	4	423	4	436	2	346
22	4	473	4	406	2	318
23	4	468	4	388	2	359
24	4	468	4	400	2	341
25	4	436	4	453	2	382
26	4	501	4	403	2	309
27	4	382	4	412	2	328
28	4	410	3	365	4	390
29	4	404	3	350	2	326
30	4	483	4	408	2	342

Table 6: Number of victims saved and respective simulation duration for 3 different environment noise levels.

In Table 7, we see the number of waypoints explored by the robot team during each trial. Also in Table 7, we can observe the number of times the MANET connection was endangered, *i.e.*, the connection threshold was exceeded, during the simulations, as observed, this event never occurred for a threshold of -80 (dBm).

	Environment Noise Level					
	10 (dB)		20 (dB)		30 (dB)	
TRIAL	**Explored Waypoints**	**Connection endangered**	**Explored Waypoints**	**Connection endangered**	**Explored Waypoints**	**Connection endangered**
1	5	0	5	0	5	0
2	5	0	5	0	5	0
3	5	0	5	0	5	0
4	5	0	5	0	5	0
5	5	0	5	0	5	0
6	5	0	5	0	5	0
7	5	0	5	0	5	0
8	5	0	5	0	5	0
9	4	0	5	0	5	0
10	5	0	5	0	5	0
11	5	0	5	0	5	0
12	5	0	5	0	5	0
13	5	0	5	0	5	0
14	5	0	5	0	5	0
15	4	0	5	0	5	0
16	5	0	5	0	5	0
17	5	0	5	0	5	0
18	5	0	5	0	5	0
19	5	0	5	0	5	0
20	5	0	5	0	5	0
21	5	0	5	0	5	0
22	5	0	5	0	5	0
23	5	0	5	0	5	0
24	5	0	5	0	5	0
25	5	0	5	0	5	0
26	5	0	5	0	5	0
27	5	0	5	0	5	0
28	5	0	5	0	5	0
29	5	0	5	0	5	0
30	5	0	5	0	5	0

Table 7: Number of waypoints explored and number of times the MANET connection was endangered for 3 different environment noise levels.

5.4.1 Discussion and analysis

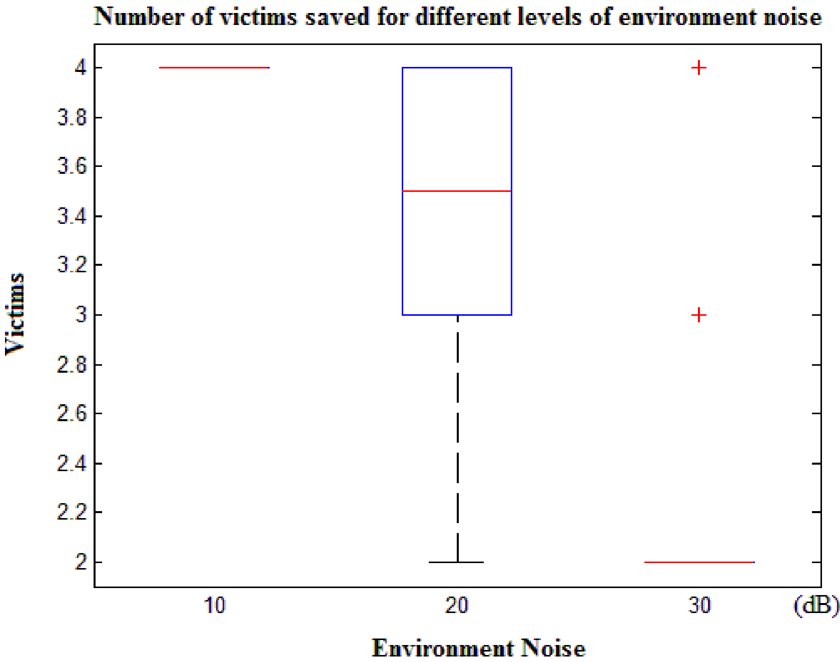


Figure 19: Victims saved for different levels of environment noise.

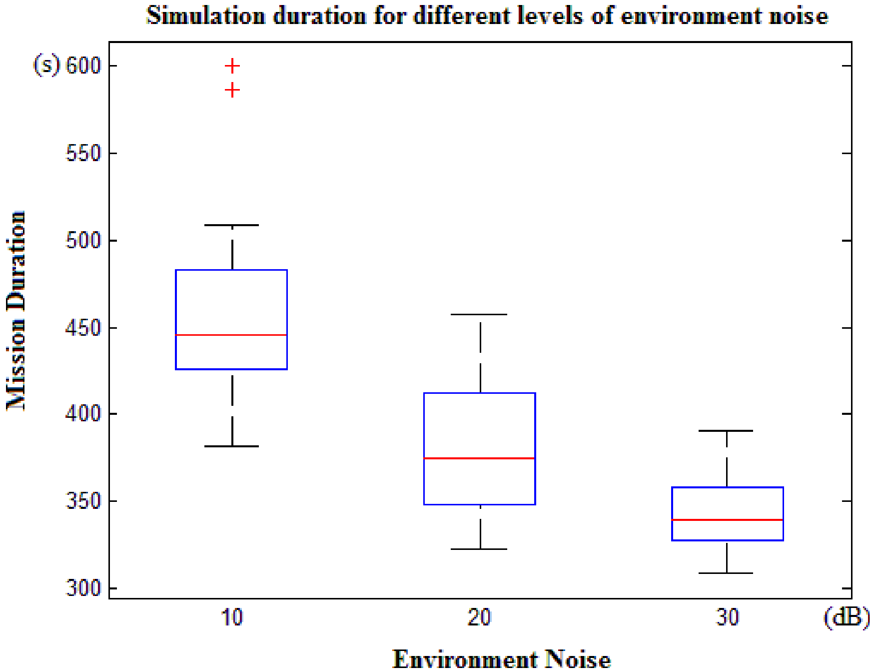


Figure 20: Mission durations for different levels of environment noise.

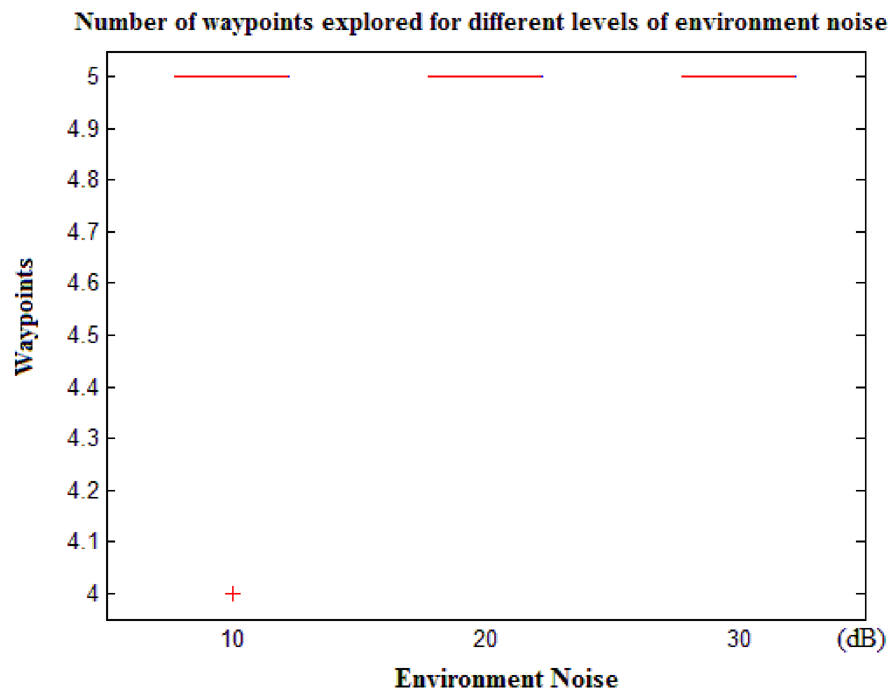


Figure 21: Waypoints explored for different levels of environment noise.

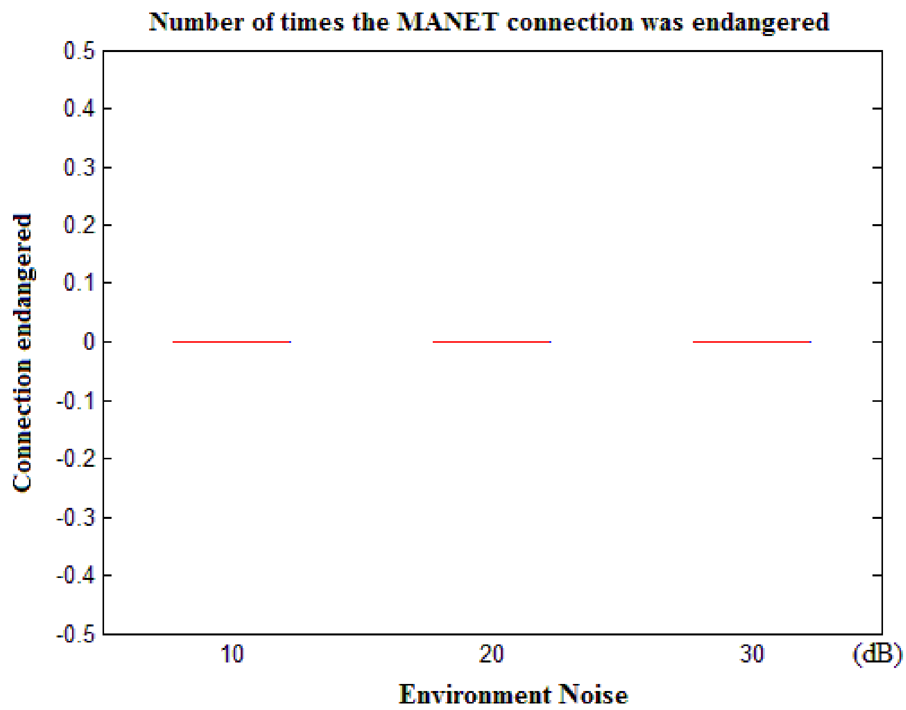


Figure 22: Number of times the MANET connection was endangered for different levels of environment noise.

	Environment Noise Level					
	10 (dB)		20 (dB)		30 (dB)	
	Victims saved	**Time (sec)**	**Victims saved**	**Time (sec)**	**Victims saved**	**Time (sec)**
Mean	4	459,6	3,3	382	2,1	343,5
Median	4	445,5	3,5	374,5	2	339,5
Mode	4	447	4	348	2	327

Table 8: Mean, median and mode values for Table 6.

	Environment Noise Level					
	10 (dB)		20 (dB)		30 (dB)	
	Explored Waypoints	**Connection endangered**	**Explored Waypoints**	**Connection endangered**	**Explored Waypoints**	**Connection endangered**
Mean	4,93	0	5	0	5	0
Median	5	0	5	0	5	0
Mode	5	0	5	0	5	0

Table 9: Mean, median and mode values for Table 7.

From Fig. 19, we can observe that with increased background noise it is harder for the robot team to detect victims due to the detection range becoming lower. This forces the rescue team to "walk" closer of the victims to be able to detect them.

By looking at Fig. 20 we see a decrease in duration of the mission with increased noise level, this is only due to the number of victims saved being lower and the robot team does not expend as much time saving victims, it is not directly influenced by the environment noise itself.

The influence of the background noise over victim detection can be seen in Table 8, with a level of 10 dB of noise all victims are saved in each mission. From noise levels of 20 dB the number of victims in average lowers a little. This decrease in the number of victims saved is much higher for a level of noise of 30 dB, in which only half of the victims are saved.

A solution for this problem could be the use of an efficient patrolling algorithm that can adapt the robot team's path of exploration according to the background noise levels. With more noise the patrolling and exploration of the area needs to be stricter, *e.g.*, the robot team has to explore more of the map to detect all victims. On the other hand, with less noise the detection range increases and the robot team does not need to cover as much ground to detect all victims, which can lead to a significant drop on the mission's duration.

Fig. 22 shows that, as expected, choosing a connection threshold of -80 (dBm) leads to a good performance as the rescue team did not expend time repositioning themselves to maintain the MANET's connectivity and had the "freedom" of movement to pursue the mission's objectives, exploring all areas and rescuing victims when possible, Fig. 21 and Table 8.

5.5 Summary

In this chapter, we saw how the effectiveness of a SaR mission based on voice detection can drastically drop with the environment sounds influence. Since the detection range is lower the rescue team exploration has to be bigger and so their patrolling behavior will have to adapt accordingly to the background noise.

In the next chapter we present our conclusions and discuss future work.

CHAPTER 6

6 - Conclusion and Future Work

6.1 Conclusion

Gazebo provides a good interface and environment for research and simulation of mobile robotic algorithms. Its compatibility with ROS allows continue research of robot controllers within Gazebo worlds and the use of several packages that ROS has to offer completing the Gazebo simulations.

The radio frequency model, the voice propagation model and the RDPSO algorithm were implemented in ROS architecture and can be used for future research in SaR scenarios.

In SaR scenarios where the vision is highly restricted, the use of audio, *e.g.*, voice signal, to detect and locate victims is strongly limited to the environment noise levels, making it harder for the firefighting entities to find and rescue victims. We conclude that a more thoroughly exploration of the map is necessary to assure the best rescue rate possible with higher background noise.

All initial objectives were achieved except for one, to simulate in Gazebo a ROS stack, already fully developed and tested with Stage, which includes the *State Exchange Bayesian Strategy* (SEBS) algorithm [29], but not achieved due to the submission deadline. One optional objective, also not achieved, was the simulation of fire propagation in the scenario, although the fire propagation model was studied it was not implemented.

With this dissertation, apart from the plugin and tutorial in Appendixes 1 and 2, we contribute with a set of ROS libraries used in the algorithms, propagation models and the creation of the maps necessary for the RF and voice models. In addition, we offer several Gazebo models, such as the Stingbot robot model, the ISR-UC's floor 0 and garage scenario models, etc., created during this dissertation. For the scenario models, we also make available the respective 3D SketchUp and COLLADA (.dae) files and respective 2D Stage map versions, allowing their use in any future work outside of Gazebo.

Our simulation does not support multiple swarms, *i.e.*, it is limited to only 1 swarm, and is limited to a maximum number of robots, this max number can be increased with the creation of

additional ROS topics to match the additional robots. It does not possess an adaptive patrolling algorithm, *i.e.*, be able to calculate the next exploration area based on the mission context. Another limitation of our simulation is the need for extended knowledge of the map beforehand (*e.g.*, position, type and thickness of the walls), necessary for the correct simulation of the propagation models.

6.2 Future Work

We suggest the implementation of a patrolling algorithm to decide which areas to explore at each given time, based on the scenario context, in order to optimize the SaR mission, both in time and success rate. One algorithm suggested is the SEBS [29], which will allow a higher rescue rate, by increasing the exploration effectiveness.

In addition to the RF and voice propagation models, a fire propagation model could be implemented. This fire propagation could mimic the models used in [8], mentioned as related work, and be used as a dynamic obstacle for the rescue team, both human and robotic, and may eventually simulate casualties on the rescue teams and victims. Another suggestion is the implementation of victim behavior during the mission, in our simulations all victims remained static, which would not happen in real life scenario.

References and Bibliography

[1] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-source Multi-Robot Simulator," in *International Conference on Intelligent Robots and Systems*, Sendal, Japan, 2004.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software,* 2009.

[3] B. P. Gerkey, R. T. Vaughan and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," *Proceedings of the International Conference on Advanced Robotics (ICAR 2003),* pp. 317-323, 2003.

[4] M. S. Couceiro, R. P. Rocha and N. Ferreira, "A Novel Multi-Robot Exploration Approach based on Particle Swarm Optimization Algorithms," *Proceedings of the 2011 IEEE International Symposium on Safety, Security and Rescue Robots,* pp. 327-332, 2011.

[5] A. Araújo, D. Portugal, M. S. Couceiro and R. P. Rocha, "Integrating Arduino-based Educational Mobile Robots in ROS," in *Proceedings of 13th International Conference on Autonomous Robot Systems and Competitions*, Lisbon, Portugal, 2013.

[6] O. Michel, "Webots: Symbiosis Between Virtual and Real Mobile Robots," *Virtual Worlds 98,* pp. 254-263, 1998.

[7] W. Dong, F. Palmquist and S. Lidholm, "A simple and effective emulation tool interface development for tricept application," in *Proceedings of the 33rd ISR (International Symposium on Robotics)*, October 7-11, 2002.

[8] M. S. Couceiro, D. Portugal and R. P. Rocha, "A Collective Robotic Architecture in Search and Rescue Scenarios," *In Proc. of 28th ACM Symposium on Applied Computing (ASC 2013), Coimbra, Portugal,* pp. 64-69, Mar. 18-22, 2013.

[9] T. Choi, H. Do, C. Park, D. Park, S. Lee and J. Kyung, "Software Platform for the Industrial Dual-Arm Robot," *Robot Intelligence Technol. & Appl.,* pp. 911-920, 2012.

[10] M. Freese, S. Singh, F. Ozaki and N. Matsushira, "Virtual Robot Experimentation Platform V-REP: A versatile 3D Robot Simulator," *SIMPAR,* pp. 51-62, 2010.

[11] L. Hugues and N. Bredeche, "Simbad: An Autonomous Robot Simulation Package for Education and Research," *SAB,* pp. 831-842, 2006.

[12] K. Ayush and N. K. Agarwal, "Constraint Optimised Path Tracking for Social Robots," *Undergraduate Academic Research Journal ,* vol. 1, 2012.

[13] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, "USARSim: a robot simulator for research and education," *2007 IEEE International Conference on Robotics and Automation,* Rome, Italy, 10-14 April 2007.

[14] J. Jackson, "Microsoft Robotics Studio: A technical Introduction," *Robotics & Automation Magazine,* pp. 82-87, 2007.

[15] A. Ranganathan and S. Koenig, "A reactive robot architecture with planning on demand," *Intelligent Robots and Systems (IROS),* pp. 1462-1468, 2003.

[16] F. P. Fontán and P. M. Espiñeira, Modeling the Wireless Propagation Channel, a Simulation Approach with Matlab, Sep. 2008.

[17] W. H. Tranter, K. S. Shanmugan and T. S. Rappaport, Principles od Communication Systems Simulation With Wireless Applications, PRENTICE HALL, 2003.

[18] A. Borreli, C. Monti, M. Vari and F. Mazzenga, "Channel models for IEEE 802.11b indoor system design," *IEEE International Conference on Communications,* vol. 6, pp. 3701-3705, June 2004.

[19] B. Truax, Hanbook for Acoustic Ecology, 2nd ed., World Soundscape Project, Simon Frase University and ARC Publications, 1999.

[20] C. W. Reynolds, "Flocks, herds and schools: A Distributed Behavioral Model," *Computer Graphics,* vol. 21, pp. 25-34, 1987.

[21] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proc. of IEEE International Conference on Neural Network,* pp. 1942-1948, 1995.

[22] J. Tillet, T. Rao, F. Sashin and R. Rao, "Darwinian Particle Swarm Optimization," *Proceedings of the 2nd Indian International Conference on Artificial Intelligence,* pp. 1474-1487, 2005.

[23] H. Q. Min, J. H. Zhu and X. Zheng, "Obstacle avoidance with multiple-objective optimization by PSO in dynamic environment," *Proceeedings of the Fourth International Conference on Machine Learning and Cybernetics,* pp. 2950-2956, Augusto 2005.

[24] M. S. Couceiro, R. P. Rocha and N. Ferreira, "Ensuring Ad Hoc Connectivity in Distributed Search with Robotic Darwinian Particle Swarms," *Proceedings of the 2011 IEEE International Symposium on Safety, Security and Rescue Robots,* pp. 284-289, 2011.

[25] M. S. Couceiro, F. Martins, R.P.Rocha and N. Ferreira, "Analysis and Paramater Adjustment of the RDPSO Towards an Understanding of Robotic Network Dynamic Partitioning based on Darwin's Theory," *International Mathematical Forum,* vol. Vol.7, pp. 1587-1601, 2012.

[26] M. S. Couceiro, F. M. L. Martins, R. P. Rocha and N. M. F. Ferreira, "Introducing the Fractional Order Robotic Darwinian PSO," *AIP Conference Proceedings,* pp. 242-251, 2012.

[27] M. S. Couceiro, J. Machado, R.P.Rocha and N. M. F. Ferreira, "A fuzzified systematic adjustment of the robotic Darwinian PSO," *Robotics and Autonomous Systems,* pp. 1625-1639, 2012.

[28] N. Michael, M. M. Zavlanos, V. Kumar and G. J. Pappas, "Maintaining Connectivity in Mobile Robot Nerworks," *The 11th International Symposium Experimental,* pp. 117-126, 2009.

[29] D. Portugal and R. P. Rocha, "Decision Methods for Distributed Multi-Robot Patrol," *Safety, Security and Rescue Robotics (SSRR),* pp. 1-6, 2012.

Appendix 1 – ROS Plugin

```
#include <boost/bind.hpp>
#include <gazebo.hh>
#include <physics/physics.hh>
#include <common/common.hh>
#include <stdio.h>

//ros libraries
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "sensor_msgs/LaserScan.h"
#include <nav_msgs/Odometry.h>
#include "geometry_msgs/Vector3.h"
#include "geometry_msgs/Quaternion.h"

//gazebo libraries
#include "gazebo/gazebo.hh"
#include "gazebo/physics/physics.hh"
#include "gazebo/common/common.hh"
#include "gazebo/sensors/RaySensor.hh"
#include "gazebo/sensors/SensorManager.hh"

#include "math/Vector3.hh"
#include <math/gzmath.hh>


static float vlin=0.0, vang=0.0;

namespace gazebo
{
  class ROSModelPlugin : public ModelPlugin
  {

     public: ROSModelPlugin()
     {
          // Start up ROS
           // name will also be the sufix of the rostopic
           // ex1: name = "robot_0", with the below cmd_vel topic, results in the topic name: /robot_0/cmd_vel
           // ex2: name = "", will lead to the topic name: /cmd_vel , commonly used in simulations with only 1 robot
          std::string name = "robot_0";
           int argc = 0;
          ros::init(argc, NULL, name);
     }

     public: ~ROSModelPlugin()
     {
          delete this->nh;
     }

     public: void Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/)
     {
          // Store the pointer to the model
          this->model = _parent;

          // ROS Nodehandle
          this->nh = new ros::NodeHandle("~");

          // ROS Subscriber
```

```
        this->subvel = this->nh->subscribe<geometry_msgs::Twist>("cmd_vel", 1000,
&ROSModelPlugin::VelocityCallback, this );

        // ROS publishers
        this->nh = new ros::NodeHandle("~");
        this->pubscan = this->nh->advertise<sensor_msgs::LaserScan>("base_scan",1000);

        this->nh = new ros::NodeHandle("~");
        this->pubodom = this->nh->advertise<nav_msgs::Odometry>("odom", 1000);

        //************** LASER MODEL **********************
        // Search the model for a sensor named "laser"
        sensors::SensorPtr sensor = sensors::SensorManager::Instance()->GetSensor("laser");
        if(!sensor)
                printf("sensor is NULL\n");

        this->raysensor = boost::shared_dynamic_cast<sensors::RaySensor>(sensor);

        if(!this->raysensor)
                printf("raysensor is NULL\n");
        //**************************************************

        // Listen to the update event. This event is broadcast every
        // simulation iteration.
        // for gazebo 1.5 or higher it should be ConnectWorldUpdateBegin
        this->updateConnection = event::Events::ConnectWorldUpdateStart(
        boost::bind( &ROSModelPlugin::OnUpdate, this));
  }

  // Called by the world update start event
  // this function runs on every world "iteration", the "main" function
  public: void OnUpdate()
  {
        std::vector<double> rangesgz;
        static double qw,qx,qy,qz, Rrad, Prad, Yrad;
        static double px,py,pz;

        // ************* QUATERNION / POSE DATA *****************
        //returns the model's cartisian position
        math::Vector3 p = model->GetWorldPose().pos;
        px=p.x;  py=p.y;  pz=p.z;

        //returns the model's angular position
        math::Quaternion r = model->GetWorldPose().rot;

        //from quaternin to Roll Pitch Yaw, in radians
        qw=r.w;          qx=r.x;  qy=r.y;  qz=r.z;
        Rrad=atan2(  2*(qw*qx+qy*qz),  1-2*(qx*qx+qy*qy)  );      //Roll
        Prad=asin(2*(qw*qy-qz*qx));                                //Pitch
        Yrad=atan2(  2*(qw*qz+qx*qy),  1-2*(qy*qy+qz*qz)  );      //Yaw

        // *********** ROS Times ******************************
        ros::Time current_time, last_time;
        current_time = ros::Time::now();
        last_time = ros::Time::now();

        // *********** SCAN DATA ****************************
        //Publish Scan msg
        sensor_msgs::LaserScan scan2ros;
        scan2ros.header.stamp=current_time; //ros::Time::now();
        scan2ros.header.frame_id="base_scan";
        scan2ros.angle_min=this->raysensor->GetAngleMin().Radian();
```

```
scan2ros.angle_max=this->raysensor->GetAngleMax().Radian();
scan2ros.angle_increment=this->raysensor->GetAngleResolution();
scan2ros.time_increment=0.0;
scan2ros.scan_time=0.0;
scan2ros.range_min=this->raysensor->GetRangeMin();
float rmn=scan2ros.range_min;
scan2ros.range_max=this->raysensor->GetRangeMax();
float rmx=scan2ros.range_max;
// ***************************
// it is necessary to pass the whole laser values to the msg
this->raysensor->GetRanges(rangesgz);
int raynumber=this->raysensor->GetRangeCount();
scan2ros.ranges.resize(raynumber);
for (int iray=0;iray<raynumber;iray++)
{
        // the laser scan subtracts, to each laser measure,
        // the min range defined on the laser model
        // to "correct" this when building the scan message for ROS we add
        // the value of the min range if the sum is not greater than max range
        float rg = this->raysensor->GetRange(iray);
        if(rg+rmn<=rmx)
        {scan2ros.ranges[iray]=rg+rmn;}
        else {scan2ros.ranges[iray]=rmx;}
}
// **************************
// publish the message
this->pubscan.publish(scan2ros);
// *****************************************************************


// *********** ODOMETRY DATA ***********************************
geometry_msgs::Quaternion odom_quat;
odom_quat.x=qx;
odom_quat.y=qy;
odom_quat.z=qz;
odom_quat.w=qw;


//publish the odometry message over to ROS
nav_msgs::Odometry odom;
odom.header.stamp =ros::Time::now(); //current_time;
odom.header.frame_id = "odom";

//set the position for odom message
odom.pose.pose.position.x = px;
odom.pose.pose.position.y = py;
odom.pose.pose.position.z = 0.0; //since the model moves in a 2D plane
odom.pose.pose.orientation = odom_quat;

//set the velocity for odom message
odom.child_frame_id = "base_link";
odom.twist.twist.linear.x = vlin;
odom.twist.twist.linear.y = 0.0;
odom.twist.twist.angular.z = vang;

//publish the message
this->pubodom.publish(odom);
// ******************************************************************

// ********* SET MODEL Velocity ***************************************
//set velocities
float velx,vely;
```

```
            velx=vlin*cos(Yrad);
            vely=vlin*sin(Yrad);
            this->model->SetLinearVel(math::Vector3(velx, vely, 0));
            this->model->SetAngularVel(math::Vector3(0, 0, vang));
            // ***********************************************

            ros::spinOnce();
    }


    // This Callback function is "linked" to the cmd_vel topic created above
    // Everytime data is sent to their topic Callback functions will execute
    void VelocityCallback(const geometry_msgs::Twist::ConstPtr& msg)
    {
            // updates the model linear and angular velocities values
            vlin=msg->linear.x;
            vang=msg->angular.z;
    }


    // Pointer to the model
    private: physics::ModelPtr model;

    // Pointer to Laser model
    private: sensors::RaySensorPtr raysensor;

    // Pointer to the update event connection
    private: event::ConnectionPtr updateConnection;


   // *** ROS ************
   // ROS Nodehandle
   private: ros::NodeHandle* nh;

   // ROS Subscriber
   ros::Subscriber subvel;

   // ROS Publisher
   ros::Publisher pubscan;
   ros::Publisher pubodom;
   //*************
};

  // Register this plugin with the simulator
  GZ_REGISTER_MODEL_PLUGIN(ROSModelPlugin)
}
```

Appendix 2 – Gazebo World, Model and Plugin tutorial

This tutorial assumes that both Gazebo and ROS have been correctly installed.

Beginner Level

ROS Plugin

Assuming you have done and comprehended the tutorial on how to set up a ROS-plugin compiler, http://gazebosim.org/wiki/Tutorials/1.4/ros_enabled_model_plugin, and that for now you are using the same folder, `gazebo_ros_plugin`.

1- Go to `gazebo_ros_plugin` folder and edit the `CMakeLists.txt` file by adding at the end the following code:

```
add_library(rosplugin SHARED robot_0_plugin.cc)
set_target_properties(robot_0_plugin PROPERTIES COMPILE_FLAGS "${roscpp_CFLAGS_OTHER}")
set_target_properties(robot_0_plugin PROPERTIES LINK_FLAGS "${roscpp_LDFLAGS_OTHER}")
target_link_libraries(robot_0_plugin ${roscpp_LIBRARIES})
install (TARGETS robot_0_plugin DESTINATION ${CMAKE_INSTALL_PREFIX}/lib/gazebo_plugins/)
```

2- In the same folder, create a new file named robot_0_plugin.cc, open and copy the content of Appendix 1 inside. On a terminal window enter the `gazebo_ros_plugin/build` folder and type `make`, this will compile our plugin.

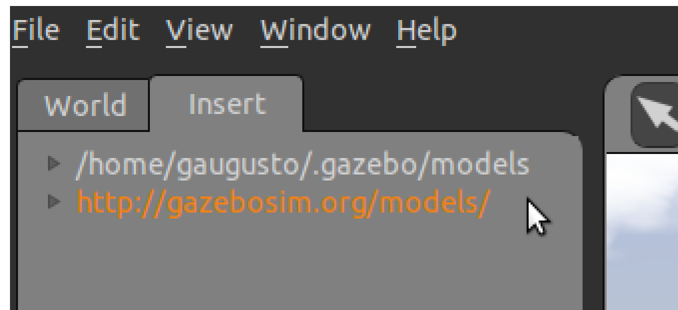
Models

In order to use the above plugin we will need some models. Models of Gazebo world are saved in the `HOME/user/.gazebo/models` folder. One can create his own models or use existing ones. In this tutorial we are going to use 3 already existing models available online.

First let's download these models:

1- In a terminal window, type `gazebo`. This will open the Gazebo interface with an empty world.

2- On the upper left corner click on the "Insert" tab, open the tab *http://gazebosim.org/model/* and then click in Hokuyo, Jersey barrier and Pioneer 2DX. This will automatically download the models to your computer `.gazebo/models/` folder. Some of these models will be changed in this tutorial, be careful not to download them again as they will replace the changed ones.



3- Now go to the models folder. A quick shortcut, press Ctrl+h to show the .gazebo folder. In the `pioneer2dx` folder open the `model-1_3.sdf` with your favorite editor (*e.g.*, gedit), scroll down to the bottom and delete the following code, line 150 to 154, this will be "replaced" with our own plugin later on.

```
146          <xyz>0 1 0</xyz>
147        </axis>
148      </joint>
149
150      <plugin filename="libDiffDrivePlugin.so" name="diff_drive">
151        <left_joint>left_wheel_hinge</left_joint>
152        <right_joint>right_wheel_hinge</right_joint>
153        <torque>5</torque>
154      </plugin>
155
156    </model>
157  </sdf>
```

4- Now go to the hokuyo folder model-1_3.sdf, in the lines 43 to 46 replace the values of the <samples> parameter from 640 to 181 (this is the number of laser rays the model will have), change the <min_angle> parameter from -2.26889 to -1.57075 (this is the "right" angle of the laser in radians, -90º) and change the parameter <max_angle> from 2.268899 to 1.57075 (the left side angle).

What we did to the laser model was change its parameters to suit better our needs, with a 180 angle (-90º to 90º), 181 samples and resolution of 1 we will obtain an array where each entry is the range measured for each degree (0º to 180º).

```
38        <sensor name="laser" type="ray">
39          <pose>0.01 0 0.0175 0 -0 0</pose>
40          <ray>
41            <scan>
42              <horizontal>
43                <samples>640</samples>
44                <resolution>1</resolution>
45                <min_angle>-2.26889</min_angle>
46                <max_angle>2.268899</max_angle>
47              </horizontal>
48            </scan>
49            <range>
50              <min>0.08</min>
51              <max>10</max>
52              <resolution>0.01</resolution>
53            </range>
54          </ray>
```

```
38        <sensor name="laser" type="ray">
39          <pose>0.01 0 0.0175 0 -0 0</pose>
40          <ray>
41            <scan>
42              <horizontal>
43                <samples>181</samples>
44                <resolution>1</resolution>
45                <min_angle>-1.57075</min_angle>
46                <max_angle>1.57075</max_angle>
47              </horizontal>
48            </scan>
49            <range>
50              <min>0.08</min>
51              <max>10</max>
52              <resolution>0.01</resolution>
53            </range>
54          </ray>
```

World

Let us now create an empty world just with ground plane, light source and no gravity.

1- Inside `gazebo_ros_plugin` folder create a `tutorial_map.world` file and copy the following inside:

```
<?xml version="1.0" ?>
<sdf version="1.3">
  <world name="default">

    <physics type="ode">
      <gravity>0 0 0</gravity>
      <ode>
        <solver>
          <type>quick</type>
          <dt>0.001</dt>
          <iters>40</iters>
          <sor>1.0</sor>
        </solver>
        <constraints>
          <cfm>0.0</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>100.0</contact_max_correcting_vel>
          <contact_surface_layer>0.0</contact_surface_layer>
        </constraints>
      </ode>
    </physics>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

  </world>
</sdf>
```

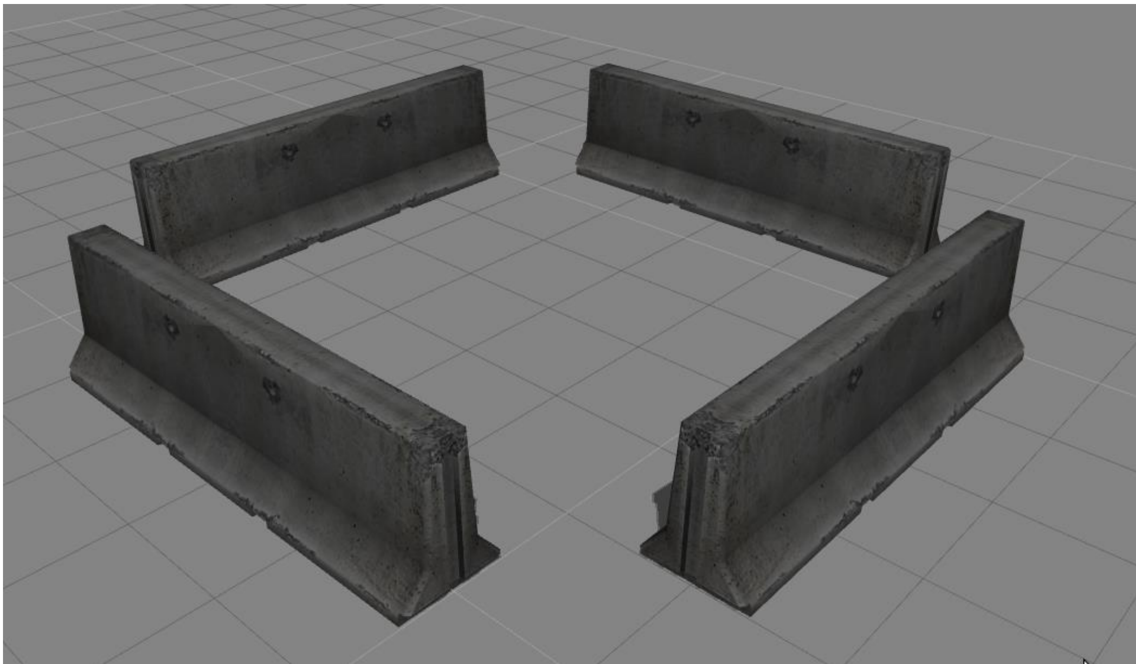
2- Now on that we have our world, let’s include some obstacles in it. For that add the following code below the sun model:

```
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

   <model name="barrier1">
            <include>
             <uri>model://jersey_barrier</uri>
            <pose> 3 0 0 0 0 0</pose>
             <static>true</static>
             </include>
  </model>
   <model name="barrier2">
            <include>
             <uri>model://jersey_barrier</uri>
            <pose> 0 3 0 0 0 1.57075</pose>
             <static>true</static>
             </include>
  </model>
   <model name="barrier3">
            <include>
             <uri>model://jersey_barrier</uri>
            <pose> 3 6 0 0 0 0</pose>
             <static>true</static>
             </include>
  </model>
   <model name="barrier4">
            <include>
             <uri>model://jersey_barrier</uri>
            <pose> 6 3 0 0 0 1.57075</pose>
             <static>true</static>
             </include>
  </model>

  </world>
</sdf>
```

Re-launch the world and you should get the tutorial arena:

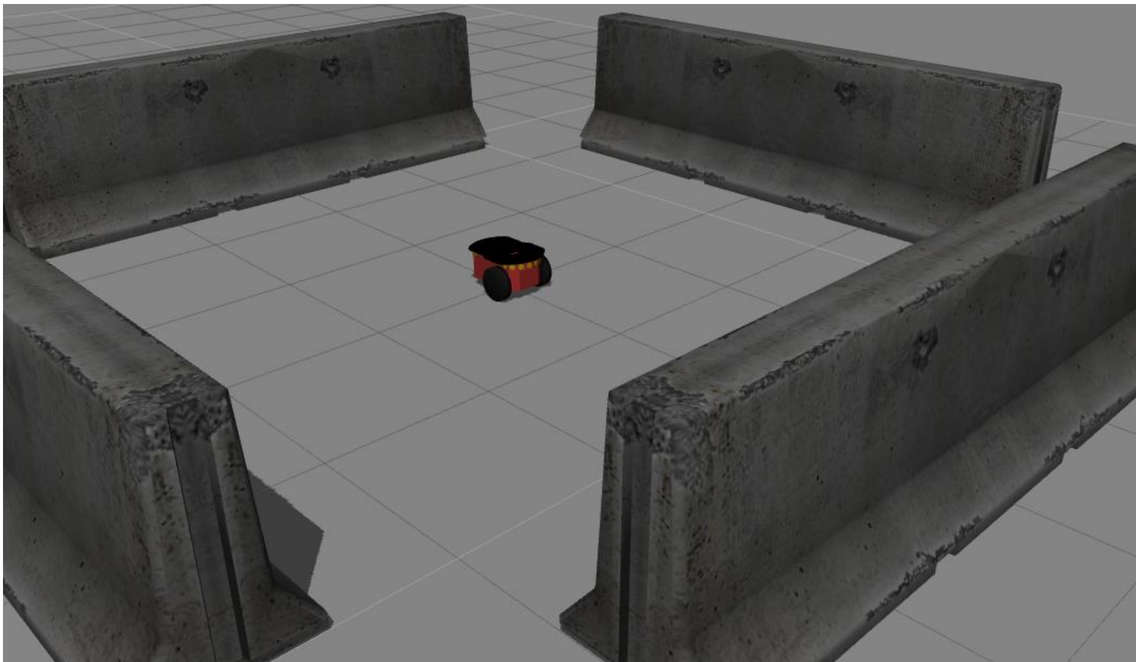


3- Now let’s add the pioneer model in the middle of the arena:

```
 <model name="barrier4">
          <include>
           <uri>model://jersey_barrier</uri>
          <pose> 6 3 0 0 0 1.57075</pose>
           <static>true</static>
           </include>
 </model>

 <model name="robot_0">
          <pose> 3 3 0 0 0 0</pose>
          <include>
          <uri>model://pioneer2dx</uri>
          <static>false</static>
          </include>
 </model>

 </world>
</sdf>
```



4- Sadly this pioneer model does not have any sensors attached. To equip a laser range finder we can use the Hokuyo model previously downloaded and equip it on the pioneer. Go to `HOME/user/.gazebo/models` folder and inside the `pioneer2dx` folder open the `mode-1_3.sdf` file and add the following code at the end:

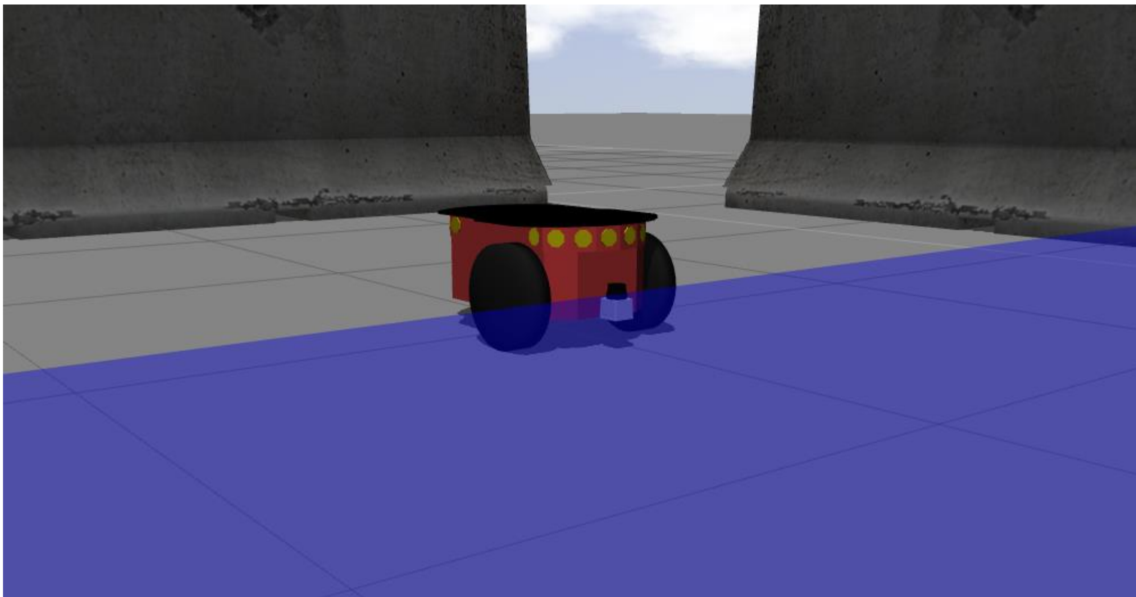
```
<!-- JOINT RIGHT WHEEL -->
<joint type="revolute" name="right_wheel_hinge">
  <pose>0 0 0.03 0 0 0</pose> <!-- 0 0 0.03 -->
  <child>right_wheel</child>
  <parent>chassis</parent>
  <axis>
    <xyz>0 1 0</xyz>
  </axis>
</joint>

        <!-- sensor -->
<include>
         <uri>model://hokuyo</uri>
         <pose>0.24 0 0.102 0 0 0</pose>
</include>
<joint name="hokuyo_joint" type="revolute">
        <child>hokuyo::link</child>
        <parent>chassis</parent>
```

```
            <axis>
            <xyz>0 0 1</xyz>
            <limit>
                <upper>0</upper>
                <lower>0</lower>
             </limit>
             </axis>
      </joint>

    </model>
</sdf>
```

Open the world and now we have our pioneer equipped with the hokuyo model.

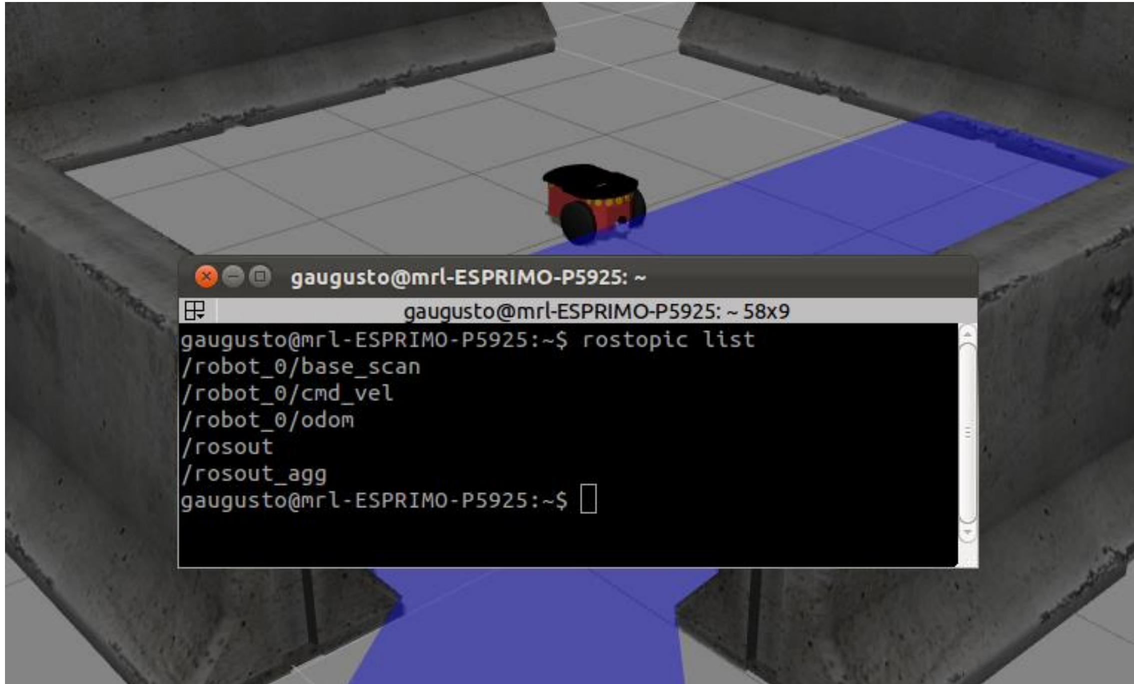


5- All that remains to have a controllable robot model is attaching the plugin to the model. In the `tutorial_map.world` file, in the pioneer section add the following line:

```
  <model name="robot_0">
            <pose> 3 3 0 0 0 0</pose>
            <include>
            <uri>model://pioneer2dx</uri>
            <static>false</static>
            <plugin name="robot_0_plugin.cc" filename="build/librobot_0_plugin.so"/>
            </include>
  </model>
```

Re-launching the world, it does not seem any different, and it shouldn't, the impact comes on the ROS side.

6- Open a new terminal window and type `rostopic list`, this command will show all active ROS-topics, you should get something similar to the image below.



Congratulations you have a controllable Gazebo robot through ROS.

TESTING

Using the above created map, we can test the plugin with a simple ROS controller available at http://wiki.ros.org/brown_remotelab. To download this stack, open a terminal window, type `roscd` to go to your workspace and then type: `svn co https://brown-ros-pkg.googlecode.com/svn/trunk/distribution/brown_remotelab`.

Now enter the `brown_remotelab` folder through the terminal window and type `rosmake`. Open `brown_remotelab/teleop_twist_keyboard/bin/teleop_twist_keyboard.py` file and change line 61 from

```
pub = rospy.Publisher('cmd_vel', Twist)
```

to

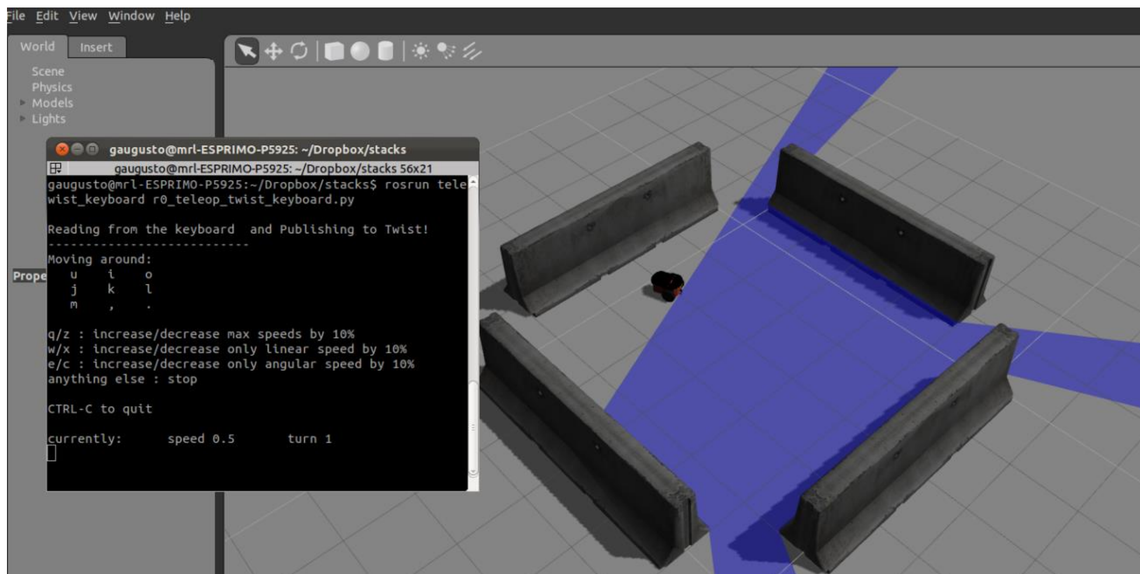
```
pub = rospy.Publisher('robot_0/cmd_vel', Twist)
```

in order to match the rostopic.

Launch the `tutorial_map.world`, open a new terminal window and type `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

You will need to have the teleop window in front for it to function.

By using the I J K L keys (and others around them) you can drive the robot around.



HOMEWORK

Edit the ROS plugin in order for when the min range measured by the laser is less than 2.0 m the robot's linear velocity is halved (i.e., vlin/2) and when the min range is below 0.3 m to never let the linear velocity be above 0 (vlin <=0).

Have Fun!