UNIVERSITY OF MACEDONIA
GRADUATE PROGRAM
DEPARTMENT OF APPLIED INFORMATICS

SWARM ROBOTICS SIMULATION USING ARGoS

Master Thesis

of

Athanasios Lentzas

Thessaloniki, 06/2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΣΟΜΟΙΩΣΗ ΡΟΜΠΟΤΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΣΜΗΝΟΥΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARGoS

Διπλωματική Εργασία

του

Αθανασίου Λέντζα

Θεσσαλονίκη, 06/2016

ΠΡΟΣΟΜΟΙΩΣΗ ΡΟΜΠΟΤΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΣΜΗΝΟΥΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARGoS

Αθανάσιος Λέντζας

Πτυχίο Εφαρμοσμένης Πληροφορικής, ΠαΜακ, 2013

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Ηλίας Σακελλαρίου

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21/6/2016

Ηλίας Σακελλαρίου	Ιωάννης Ρεφανίδης	Άγγελος Σιφαλέρας
.....	.....	.....

Αθανάσιος Λέντζας

.....

Abstract

Robotic swarms consist of a relative large number of simple robots and it is a new approach on multi robot systems. The present thesis aims to investigate how the concept of swarm robotic can be applied to a complex real world scenario, more specific on wildfire firefighting. The proposed approach is evaluated through simulation in the ARGOS platform, a tool specifically designed for swarm robotics simulation. Proposed approach is explained in detail both on the design choices made and the simulation implementation. Design of the swarm was based on a reactive architecture and the behavior is designed as a finite state machine. Proposed solution was tested on multiple simulated experiments investigating any potential value and evaluate the results on the context of wildfires.

Keywords: robotic swarms, ARGoS, robotic swarm simulation, agents, multi agent systems.

Περίληψη

Οι ρομποτικές διατάξεις σμήνους (Swarm Robotics) απαρτίζονται από ένα μεγάλο σχετικά αριθμό απλών ρομπότ και αποτελούν μια νέα προσέγγιση σε συστήματα πολλαπλών ρομπότ. Η παρούσα διπλωματική εργασία, επιχειρεί να ερευνήσει κατά πόσο μια ρομποτική διάταξη σμήνους μπορεί να χρησιμοποιηθεί στο πρόβλημα των δασικών πυρκαγιών. Η προτεινόμενη λύση αξιολογείται με χρήση προσομοίωσης στην πλατφόρμα προσομοίωσης ARGoS, έναν εργαλείο σχεδιασμένο για προσομοιώσεις ρομποτικών διατάξεων σμήνους. Παράλληλα παρουσιάζεται η σχετική βιβλιογραφία και αναλύονται οι βασικές τεχνικές ανάπτυξης συμπεριφορών που έχουν προταθεί. Στην συνέχεια, η προτεινόμενη λύση αναλύεται τόσο από σχεδιαστική σκοπιά όσο και από την σκοπιά της υλοποίησης της προσομοίωσης. Σχεδιαστικά η πρόταση βασίζεται στην αντιδραστική αρχιτεκτονική (reactive architecture) και έχει σχεδιαστεί ως finite state machine. Το κομμάτι της υλοποίησης επιχειρεί να εντοπίσει πιθανή εφαρμογή της λύσης καθώς και να την αξιολογήσει στο τρέχον πρόβλημα.

Λέξεις Κλειδιά: Ρομποτικές διατάξεις σμήνους, ARGoS, προσομοίωση ρομποτικών διατάξεων σμήνους, πράκτορες, πολυπρακτορικά συστήματα.

Περιεχόμενα

Figure index

1. Introduction

Robotic applications started to bloom over the last decade, attracting significant research. These applications can vary from simple tasks to rather complicated ones. Unfortunately, the more complicated the task, the more complicated becomes the corresponding robotic design, a fact that increases manufacturing cost as well as the maintenance cost. Swarm robotics constitutes an alternative, fairly new approach to robot development that aims to overcome this issue. The concept is inspired by agent societies in the natural world, more specifically by social insects that organize into societies capable of tasks that a single individual could not possibly accomplish. For instance, ants co-operate to find and transport large amounts of food back to their nest quickly, a task impossible for only one of them.

The Swarm consists mainly of relatively simple robots that have limited capabilities when viewed individually, however their power lies in the interaction and coordination of these relatively simple units. Research in the field includes both the physical design and the behaviour control, i.e. achieving meaningful/useful interactions. The cost of each robot has to be as low as possible making a swarm solution fairly cheap and keeping the replacement cost as low as possible. Swarm robotics can be used in multiple areas, some of the most famous are self-assembly, where the robots form a large functional object, and search and rescue events.

1.2 Aim

The present thesis aims to investigate how the concept of swarm robotics can be applied to a complex real world scenario, i.e. wildfires. Towards this goal, an initial design of a robotic swarm was developed that was tested in an appropriate simulation platform. This approach, design-simulation, etc. is usually followed, since it leads to shorter development cycles and obviously of a much lower cost than experimenting with a physical robotic agent. Thus, the swarm was simulated using ARGoS - a swarm robotics simulator - and it was tested in various wildfire scenarios. It should be noted that no new robots were designed in this thesis, instead the robots already implemented in ARGoS were employed in the experiments.

This overall research aim was decomposed into several simpler targets listed below:

- To present a brief comparison of multiple simulation environments. ARGoS is compared to some of the most famous simulation tools, such as WeBots, USARSim and the Gazebo/Stage project.
- To describe in more detail the ARGoS simulator. The simulator architecture is extensively described. Additionally, the main components and their usage are presented.
- To design and develop of the simulated environment of the wildfires scenario. The simulation environment setup as well as the components used are described in this thesis. The wildfire spread model used is also presented.
- To design and develop the robot. The foot-bot robots forming the swarm are introduced. The design of their behavior is also analyzed.
- To develop the experiment setup and testing on multiple experiments. Aforementioned behavior is tested, evaluating the swarm's performance on various environments.
- To state any further improvements. Additional research and improvements are suggested in the final chapter of the thesis.

1.3 Questions - Assumptions

One of the assumptions made in this research is that a wildfire warning system is already in place. This system provides the swarm with the GPS coordinates of the wildfire. In addition to that it is assumed that only one fire can burn at a time.

1.4 Contribution

Wildfires are one of the major natural disasters. According to USA National Oceanic and Atmospheric Administration, approximately 413 million acres were burned due to wildfires over the last decade in US according to data from National Oceanic and Atmospheric Administration (NOAA). Detection systems do exist but by the time firemen arrive, the fire has already spread through the forest.

Since swarm robotics is a fairly new concept, research is quite limited. Although there are some papers focused on wild fire fighting, most of them are focused on using aerial robots. Ground robots have not been used for that purpose that much, however

they've been used as a fireman assist tool i.e. as guiding assistance when human vision is obscured due to smoke.

The robotic swarm proposed in this research, proved to be a viable solution that can contribute in wildfire firefighting. The swarm's behavior was researched and tested in various experiments in order to investigate its performance and detect any potential value. Although further research is needed to make an application feasible, simulation shown that a swarm can effectively contain a fire or completely take it out.

1.5 Basic terminology

Below is a list of most important terms used in the thesis.

- **Agent**: A computational system (hardware or software) with the following characteristics: autonomy, social ability, reactivity, pro-activeness (Wooldrigde & Jennings, 1995).
- **Robot**: A machine capable of carrying out a complex series of tasks, that works automatically or is controlled by a computer. A robot is assumed to have feedback components, such as sensors and actuators. Robots can be considered as physical agents operating in the real world.
- **Coordination**: The organization of different elements of a complex system so as to enable them to work together effectively.
- **Collective behaviour**: Coordinated behavior of large groups of similar individuals as well as emergent properties of those groups (i.e. costs and benefits of a group membership, information transfer across the group etc.)
- **Swarm Robotics**: "*Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment*." (Sahin, 2005).
- **Simulation**: Imitation of the operation of a real world process or system over time. Robotic simulation is a kinematics simulation tool, used as a highly detailed, cell-level validation tool. (Farrington et al., 1999).
- **Simulation Platforms**: Tools used to design and create simulations. A physics and graphic engine is usually integrated to the platform.
- **ARGoS**: Autonomous Robots Go Swarm. A multi-physics simulator for large-scale swarm of robots.

1.6 Thesis structure

The rest of the thesis is organized as follows. Chapter 2 presents an overview of the field of swarm robotics and the related approaches reported in the literature. In the same chapter ARGoS, the simulation tool of choice is presented in detail and is compared with similar well-known simulation environments. The third chapter describes wildfire simulation, i.e. the simulation architecture and components used. The description includes the simulated environment, the fire spread model used and a detailed presentation of the swarm behavior. In the same chapter, the design is evaluated and performance results are reported. Finally, chapter 4 summarizes the results of the research, stating the limitations and propose future work directions.

2. Swarm Robotics: Theoretical Background & Simulation Tools

2.1 Swarm robotics

Swarm robotics is a rather recent approach in robotics. Although multi-robot approaches have been proposed earlier, robotic swarms differ in the following characteristics, as proposed by Sahin (2005):

- Robots are autonomous.
- The environment can be modified by robots acting upon it.
- All communication and sensing capabilities of robots are local.
- There is no global knowledge or centralized control.
- Robots cooperate to achieve a given task.

Social insects and animals provided the main inspiration for swarm robotics. Bonabeau et al (1999) and Dorigo and Birattari (2007) investigated the swarm intelligence and the behavior of groups of social insects/animals. Their behavior was found to be robust, scalable and flexible.

Robustness is the ability to react and cope on a possible individual loss. Social animals promote that characteristic by the absence of a leader. Scalability is the ability to keep a stable performance no matter the size of the group. Local sensing and communication is used by social animals to promote scalability. Flexibility is the ability to handle different environments and tasks. Social animals achieve that by having a relatively simple behavior.

The swarm-bot concept presented by F. Mondada et al (2004) is a fine example of how the above swarm intelligence characteristics apply on robotic swarms. The swarm proposed by Mondada et al was deployed on post catastrophic environments to help on search and rescue operations. Robustness was needed to overcome hardware failures that are frequent on extreme environments. That was ensured with distributed hardware and control. Every robot that forms the swarm is completely autonomous, capable of sensing and acting based on local information. On the same concept the swarm was designed to be scalable. Adding new individuals or removing any number of them, had a small or no impact to the overall performance and behavior. The task is distributed in any numbers of individuals currently available and they cooperate with each other to complete it. Finally, flexibility was needed to operate in post catastrophic environment. The area the robots

operate, included a large number of obstacles in any form, such as fissures, holes, walls, tubes etc. The swarm was designed to be able to self-assemble to a single entity and disband when not any more necessary, using grippers and grip points. That ability is promoting flexibility to the swarm, allowing it to operate on different environments. The gipping ability can be seen on the following figure taken from the corresponding paper:

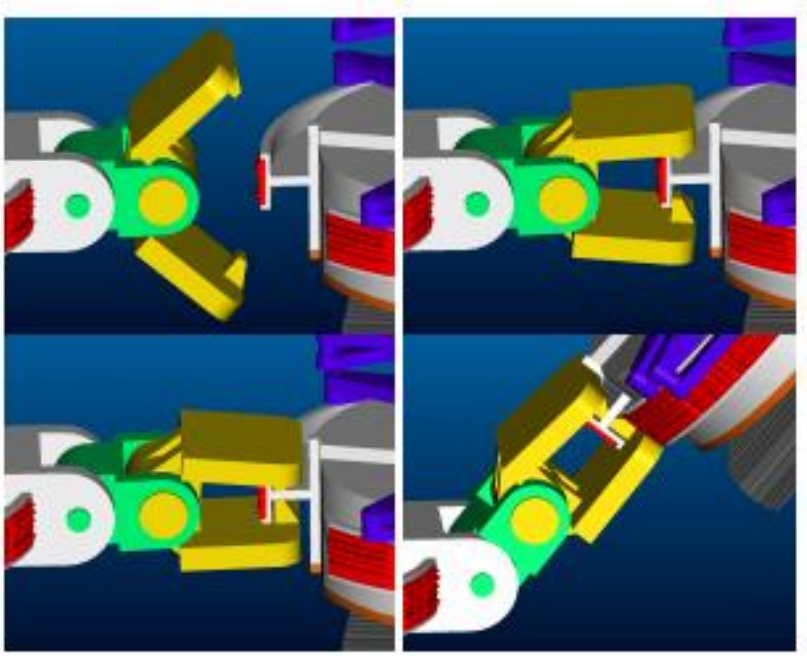

***Figure 1**: Gripper connectivity. Reprinted from F. Mondada et al (2004).*

Due to the emergent phenomena that manifest when large numbers of simple entities interact, the hardest challenge in swarm robotics is how to design them. Unfortunately, Swarm robotics do not have a specific behavior design method. Brambilla et al (2012), published an extensive review on swarm robotics from the swarm engineering perspective. The literature was analyzed, and two different taxonomies were proposed: the collective behavior and the methods taxonomy. The former categorized the main collective behaviors based on the task the swarm tries to achieve -such as self-assembly, pattern formation etc. The latter was focused on the design and analysis method. On that taxonomy Brambilla et al (2012) proposed a division of the design methods currently found in literature in two categories: behavior based and automated design.

2.1.1 Advantages and Disadvantages

Robotic swarms have some advantages that separate them from centralized robotic approaches. The most important is that a swarm can be adaptable. Building a solution that can solve a specific task is achievable using many approaches, but designing and developing a solution that can adjust on a new task is where a swarm thrives. The different parts that form the swarm can adjust and change the way they interact with each other to cope with the changes on their environment.

In order to achieve adaptability, a swarm also needs to be evolvable. Those two characteristics usually work together - if the robots can't evolve they can't adapt to the environment changes. Evolution is based on the ability of the robots to change their behavior and interaction patterns, whenever the operation space changes, allowing them to complete their given goal.

Another important benefit is that robotic swarms are resilient. In swarm systems, individuals are not critical for the operation of the whole system. Just like in nature, where a single bee is a replaceable resource and the hive can go on without it, in robotic approaches a single robot failure does not present a catastrophic problem for the whole system. Single robots that fail or are lost in the process can either be easily replaced or the swarm can continue by adapting to the changes that occurred.

Additionally, a swarm has distributed sensing and action. The former refers to the ability of the swarm to sense their environment through multiple individuals. That ability allows the sensing of a wider area and from different perspectives leading to a better understanding. The latter is the fact that a group of robots can operate on multiple locations, giving them a very important advantage compared to single robot solutions.

Despite the advantages listed above, swarm robotics also have some drawbacks compared to single robot solutions. Firstly, there is uncertainty concerning other robots' current intentions. Robots must coordinate to accomplish a given task and that coordination requires every robot to be aware of what the rest of the robots forming the swarm are intending to do. This can be unclear/uncertain due to various reasons -such as noise on sensory input- making the robots compete instead of cooperating.

Robots operating on the same space can sometimes interfere with each other. For instance, there are cases reported in the literature where robots can be trapped in closed areas, blocking each other's way. Especially when the swarm scales up, the possibility of swarm interference increases. That is because the time each individual spends on non-task relevant actions (i.e. obstacle avoidance) increases when the density of individuals rises.

Giovanni Pini et al (2014) proposed a task allocation solution to overcome the interfere problem. Tasks are partitioned and allocated to different robots reducing the need to access a common resource.

Lastly, the overall system cost can sometimes exceed the cost of a single robot approach, i.e. using more than one robots can make the economic cost higher. Swarm robotic systems intend to use many cheap and simple robots, so ideally that will not be the case. However, the total cost can sum to an amount larger than that of a single robot. In addition to that, there are tasks with high risk that can cause a lot of robots to fail or be destroyed completely.

2.1.2 Behavior Based design

The most commonly used design method employed in swarm robotics is behavior based. This method involves manually configuring and adjusting each individual behavior until the desired collective behavior is achieved. Social animals usually provide a source of inspiration for that kind of modeling. Brambilla et al (2012) divided the bibliography on behavior based design into three main categories.

First category involves approaches that adopt **probabilistic finite state machine** (PFSM, proposed by Minsky 1967) as the behaviour control component. Brooks (1986), on his *subsumption architecture* proposal, stated that on a swarm each individual acts based on its sensors input and memory and it does not plan his actions ahead. Subsumption architecture is based on layers, each of them is a PFSM, where higher level layers subsume lower level layers to create a viable solution. For example, a lower level of a robot would include obstacle avoidance while higher level world exploration. Transition probability between states can be either fixed or variable over time. Fixed probability was used by Soysal and Sahin (2005) on their work to investigate a swarm's aggregation based on the PFSM design. The probability to change states was defined at the beginning and used on all simulation experiments. A non-fixed probability, on the other hand, is when the probability is calculated using a mathematical model based on multiple system variables. One of the most used methods is the response threshold function proposed by Theraulaz et al (1990). Response threshold is a non-linear function where the probability to change state is related to the current state of the robot.

Second category is **virtual physics based design**. The source of inspiration for this method is physics domain. Every individual is modelled as a particle that applies forces to particles around it. Khatib (1986), was the first who used that concept. In his work all

robots were affected by virtual forces originating from the environment. Their goal was applying an attractive force, while obstacles a repulsive one. When we use that kind of design, we make the assumption that robots are aware of the relative position and distance from obstacles. Each robot calculates a force vector based on a potential function, direction and distance from an obstacle. The most common potential function is the Lennard-Jones function.

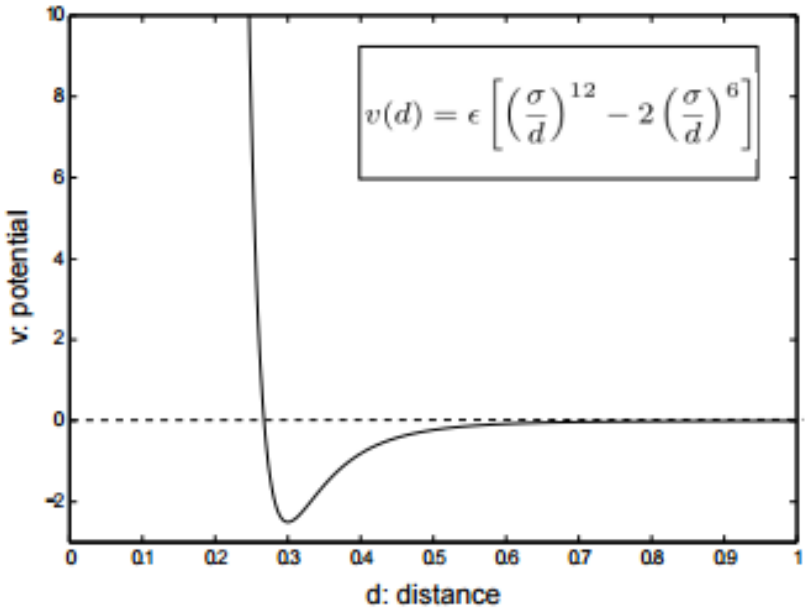

Figure 2*: Lennard-Jones potential function. Potential depends on the current distance (d) between two robots. σ is the desired distance between two robots. Reprinted from M. Brambilla et all (2012).*

The third category, according to Brambilla, includes the rest of proposed methods in literature, that do not fall in the previous two. Protoswarm scripting language was proposed by Bachrach et al (2010). That scripting language was based on the amorphous computational medium, introduced by Beal (2004). Amorphous computational medium assumes the environment is filled with robots equipped with local communication and able to perform calculations. Protoswarm allows users to program the individual behavior by scripting at the collective level.

A top down approach was proposed by Brambilla et al (2012). A properties based design was suggested, where each property is a logical formula that needs to be true at the final system. Authors suggested a process where the properties are defined, a macroscopic model is created based on those properties and a model checker, checks if those properties

are true on the suggested mode. After that, a simulation is produced using the macroscopic model as a guide for implementation, before the model is tested using real robots.

2.1.3 Automatic Design methods

Automatic design methods include those methods that behavior is automatically generated without a human interfering. Two main subcategories of automatic design methods are found on the literature: Reinforced learning and Evolutionary robotics.

Reinforced learning is learning through trial and error and is the principal learning method in autonomous systems. Each agent is getting some feedback, positive or negative, when interacting with the environment. That approach can be used on a single robot but it's not easily applied in swarm robotics. In the latter the task has to be collectively completed, however learning concerns the behaviour of each individual. The main problem, as described in Wolpert and Tumer (1999), is how we can split the environment reward to each robot - a problem called *spatial credit assignment*. Mataric (1997, 1998), after experimenting on a relatively small swarm (2 to 4 robots), proposed local communication or signaling to share the rewards. The environment the swarm is operating in, creates problems when using reinforced learning. Robots don't have a complete perception of the environment, thus searching for a behavior is a complex task. Mataric tried to solve that problem using communication between robots (1998) or behavioral decomposition (1997). Additionally, the environment, from an individual perspective, is non stationary. The actions a robot will perform are influenced by the actions performed by other robots or changes that happen in the environment itself. This problem has not been addressed yet in literature and remains an open research question.

Evolutionary robotics is a method applying evolutionary computation techniques on multi-robot systems, proposed by Nolfi and Floreano (2000). Evolutionary robotics on a swarm is similar to the evolutionary algorithms. At the beginning we generate a number of random individual behaviors. During each iteration we execute a number of experiments for each behavior. The same behavior is used by each robot and the collective behavior is evaluated using a fitness function. Individual behaviors with highest score are used to apply genetic operators -cross over and mutation-, and used in the next iteration. The evolutionary method is usually used to find the parameters of an artificial neural network, used for specifying individual behavior. Depending on whether or not memory of previously seen inputs is needed, a feed-forward -proposed by Fine (1999) - or recurrent

neural networks (as described by Beer and Gallagher, 1992; Elman, 1990) can be used respectively.

2.2 Swarm robotics Simulators

There is no argue that a simulation has many advantages when developing and testing a real world application, and this is particularly true to robotic applications. This fact has inevitably led to the development of a large number of simulators. In the sections that follow, three of the most well-known simulators are briefly presented, and compared to ARGoS, which is the simulator employed in the current project.

Robotic simulators allow the creation of embedded applications without access to the physical machine, reducing cost and time drastically. Most modern simulators include all the components users need to create a simulated robot and environment. A graphic API (application programming interface) used for 2D or 3D rendering is provided, with OpenGL[1] and OGRE[2] being the most famous of them. Physics engine is also an important component of any simulator. It provides an approximate simulation of physical systems or physical phenomena, such as collision detection, soft and rigid body dynamics etc. Lastly a framework is included, giving access to the users on various simulated elements (i.e. robots, sensors, actuators) used to create a simulated experiment. While most tools can be used with all the above prepackaged, most of them allows the integration of 3rd party graphic/physics engine.

2.2.1 Gazebo/Stage Simulator (Player project)

The player project[3] is a free simulator developed for usage on robotics research. The development team consists of many researchers and it's used by a large number of labs around the world. Player project consists of three parts: Player[4], Gazebo[5] and Stage[6]. Player is a robot abstraction layer where all devices are abstracted to predefined interfaces. All sensors and actuators as well as all the behavior coding is achieved using the Player software. Stage is a two dimensional (2D) multi-robot simulation environment while Gazebo is the three dimensional one. While one of the most famous simulators, the Player project has many drawbacks when it comes to swarm robotics simulation. Gazebo, the 3D

[1] https://www.opengl.org/
[2] http://www.ogre3d.org/
[3] http://playerstage.sourceforge.net/index.html
[4] http://playerstage.sourceforge.net/player/player.html
[5] http://playerstage.sourceforge.net/gazebo/gazebo.html
[6] http://playerstage.sourceforge.net/stage/stage.html

simulator doesn't support more than 10 robots simultaneously, which allows for rather limited in number swarms. A larger number of robots is supported using the Stage simulator. The main issue here is that Stage is 2 dimensional and there is no way to simulate sensor noise, a critical experiment parameter when dealing with real robotic systems.

Player project does have some features that make it the most used simulator at the moment. One of them is that the physics are simulated on a kinematic fashion, allowing a machine of average computational power to simulate a large number of robots, only on the 2D simulator though. In addition to that a large range of sensors are provided including sonar, laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry[7]. Also, Stage devices present a standard Player interface, making code transfer from simulation to real life robots really easy. Lastly, any controller developed for the Stage environment can be transferred and used on Gazebo with hardly any modification.

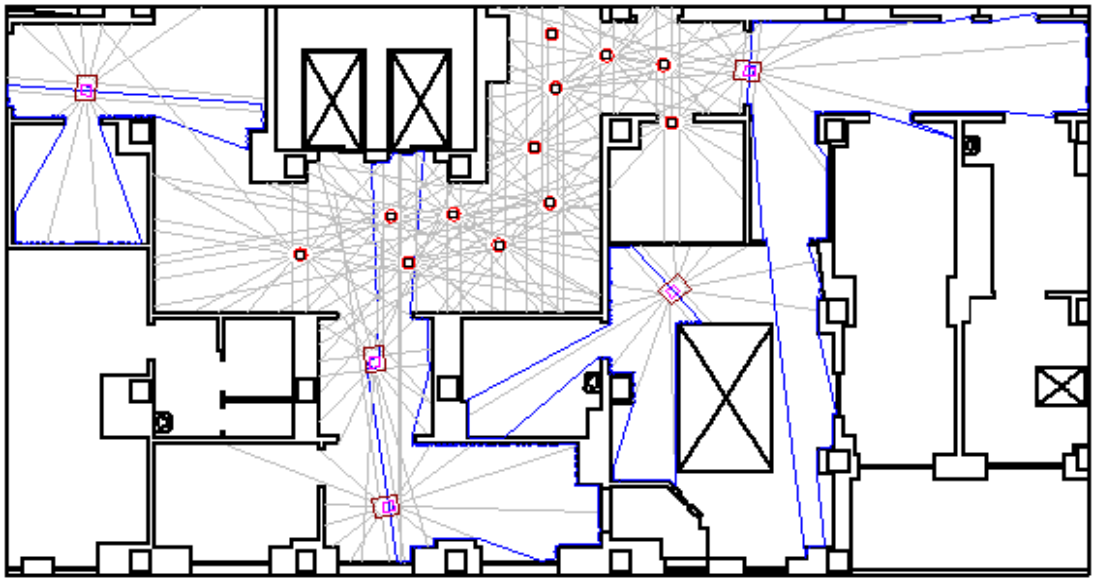

***Figure 3**: Stage 2D simulated environment.*

[7] http://playerstage.sourceforge.net/wiki/Basic_FAQ

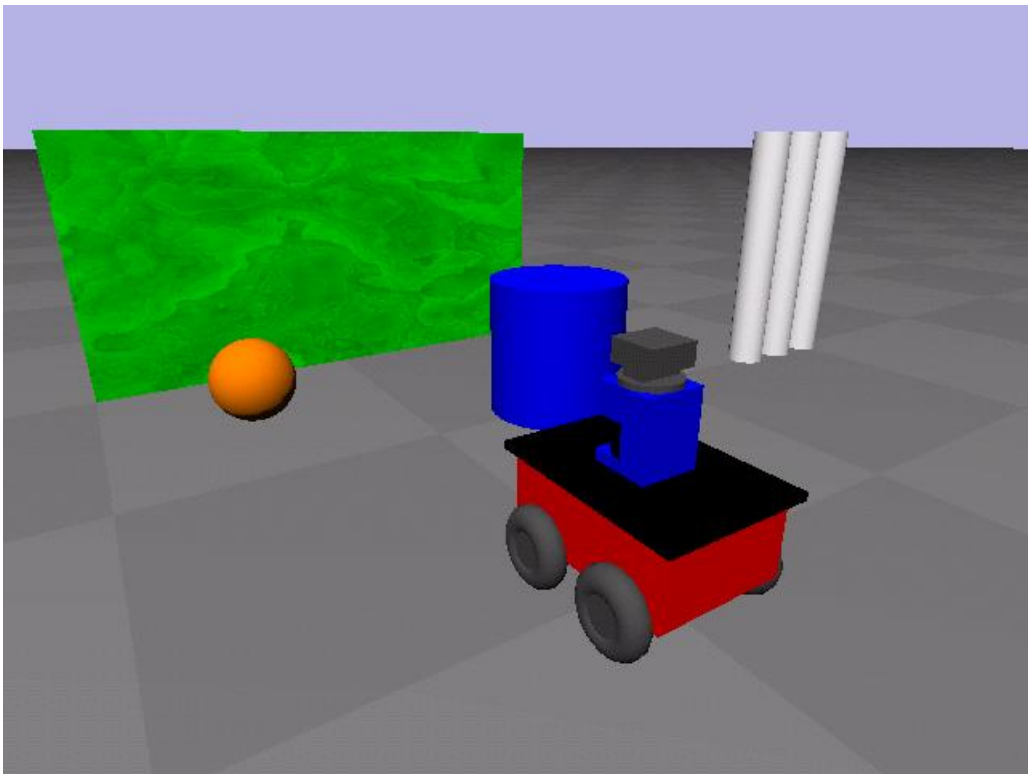

***Figure 4**: Gazebo 3D simulated environment*

2.2.2 USARSim

Urban Search and Rescue Simulation (USARSim[8]) is a robot simulator originally developed for the needs of search and rescue challenge of Robocup[9] contest. It is based on Unreal Engine[10], a game engine developed by Epic games[11]. This commercial, industry grade game engine provides the simulator with high accuracy physics, a lot of geometrical and physical models, as well as models with relatively low computational demands and high speed. USARSim ships with a collection of robots already implemented including wheeled, flying, underwater and legged models. All of them are fully customizable according to user's needs, and controlled using TCP/IP communication. All of the sensors provided can add noise to the data making the reported values closer to the real world readings.

USARSim is considered a state of the art simulator. When it comes to swarm robotics it is also a trustworthy simulator. The main drawback is the dependence on Unreal

[8] https://sourceforge.net/projects/usarsim/
[9] http://www.robocup.org/
[10] https://www.unrealengine.com/what-is-unreal-engine-4
[11] https://www.epicgames.com/

Engine, since the more robots we simulate the more computationally demanding our simulation becomes. Thus, a higher end system is required for the game engine to render and simulate a large swarm.

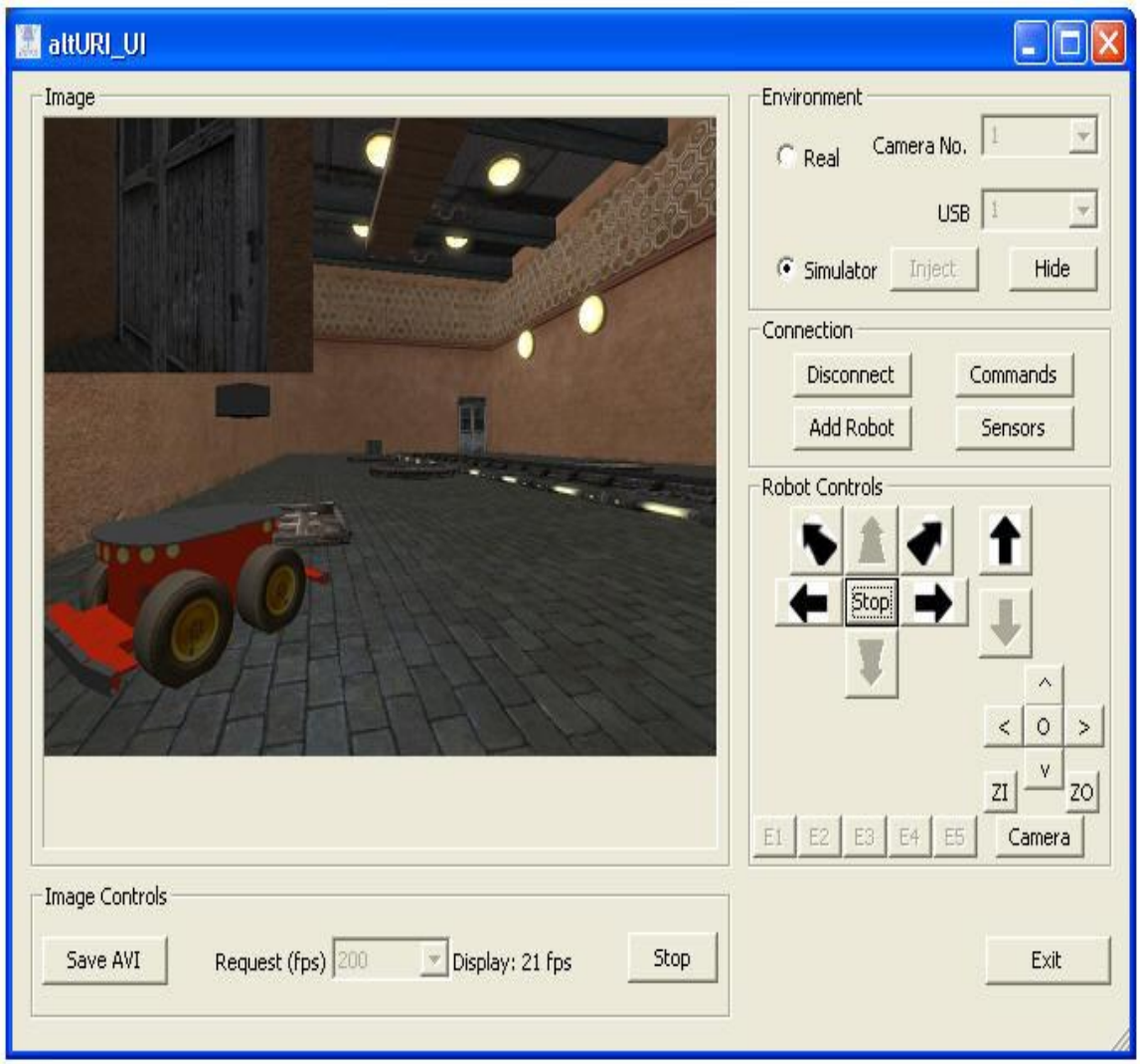

***Figure 5**: USARSim simulated environment with corresponding controls.*

2.2.3 WeBots

WeBots is a commercial simulator published by Cyberbotics Ltd[12]. In this simulator, an Open Dynamics Engine is used for collision detection and rigid body simulation. This library simulates accurately physical properties such as inertia, velocity,

[12] https://www.cyberbotics.com/overview

friction etc. WeBots supports a variety of robotic models, ranging from a human like NAO[13] (Aldebaran Robotics) to a drone like Swinglet (senseFly[14]). Users can design and implement their own robots in WeBots, by describing its graphical and physical properties. Swarm robot simulation is implementable in WeBots since the only parameter that limits the maximum number of robots is memory and CPU. Controllers can be developed using a variety of programing languages including C++, Java, Python and Matlab. Additionally, controllers can be ported to real life robots via ROS (Robot Operating System) supported by a variety of robots as well as NAOqi for NAO and any serial command protocol (such as Surveyor, Katana etc.). WeBots can be used on all operating systems (Linux, Windows, Mac), has a big user base and Cyberbotics update their product regularly. Its main drawback is the cost to buy the product. Although there is a free version, the robots have to be purchased separately or the user has to buy an upgraded different version. This makes WeBots a good but costly solution.

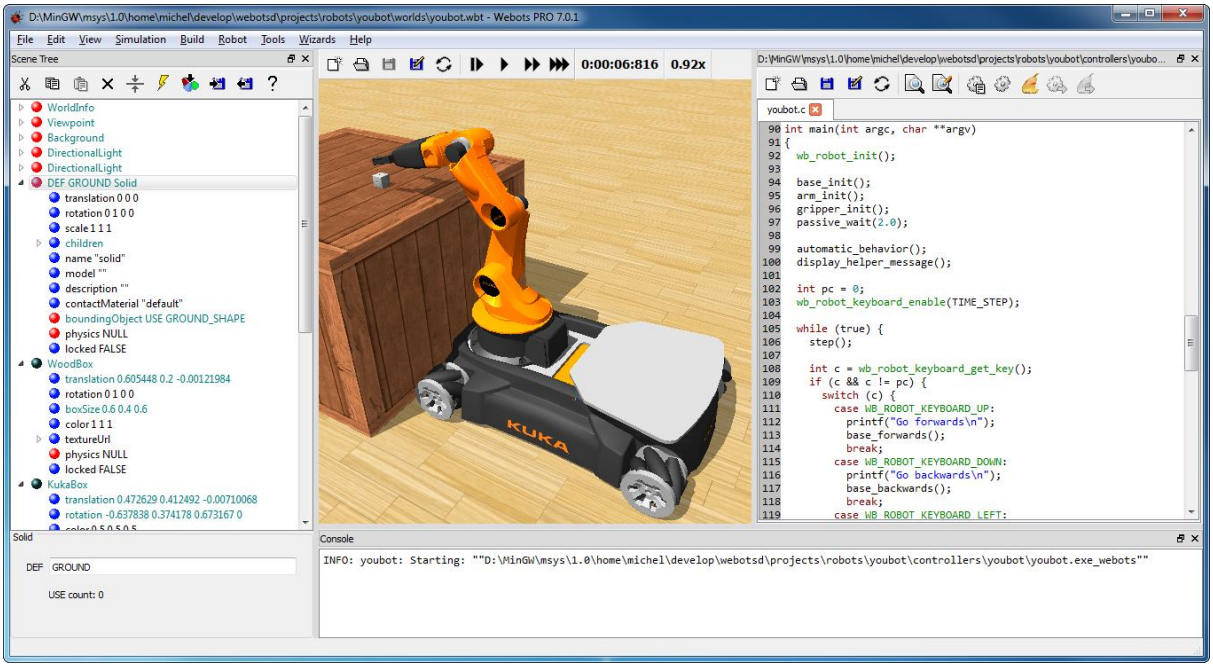

***Figure 6**: WeBots simulation.*

[13] https://www.ald.softbankrobotics.com/en/cool-robots/nao
[14] https://www.sensefly.com/drones/overview.html

2.3 The ARGoS Simulator

ARGoS[15] (Autonomous Robots Go Swarm) is a physics based discrete-time simulator originally developed for the Swarmanoid project, presented by Dorigo et al. (2012). Being a physics based simulator basically means that all agent bodies are simulated through physics models. The main issues ARGoS team tried to solve when it comes to swarms of heterogeneous robots are scalability and flexibility. The design choices made when ARGoS was designed and developed solved those issues. In the sections that follow, we describe in more detail the simulator, since the latter was selected to perform the experiments concerning wildfire management.

ARGoS simulator allows user to create simulations of large heterogeneous robotic swarms. Robots available to users include ground robots and a drone robot as well as their simulated sensors and actuators. Robots provided are the following:

- **E-puck**: a small open hardware differential wheeled mobile robot, designed for educational purposes by Mondada et al (2009)
- **Foot-bot**: a ground based robot, utilizing a combination of tracks and wheels to move around
- **Eye-bot**: quad-rotor aerial robot.
- **Generic model**: A generic differential wheeled robot model with basic sensors.

Implementation of new simulated entities is allowed, by specifying the physical properties of the entity, the visual representation and the sensors/actuators available (if any) as well as their physical properties, such as position and orientation.

An experiment in ARGoS consists of the experiment configuration file, the behavior of the robots and if needed a file including simulation management logic. Configuration file includes all the information needed to set up the simulated environment and the entities included in the simulation. The behavior of the robots is provided by specifying the location of the compiled user code. Developing in ARGoS can be done either using C++ or Luas -an embeddable scripting language. Interfering directly with the simulation, altering the environment based on certain conditions (such as adding more obstacles after certain amount of time) can also be done using user defined code.

[15] http://www.argos-sim.info/

2.3.1 Physics engine - a multi engine approach

A multi-physics engine approach was adopted for the development of the simulator. In this approach, multiple engines run in parallel, controlling different aspects/entities of the environment, thus taking advantage of modern multi-core architectures. These engines differ on the dimensions used (2D or 3D engine) and the type of the engine -rigid body, kinematic, particle engine. Argos not only allows users to choose a physics engine they prefer, but also have more than one at the same time. This approach not only provides users with high flexibility but also gives them serious advantages in terms of performance. For instance, separate physics engines can be used to manage different elements of the environment (i.e. some rooms can be managed from a 3D engine while others by a 2D one if the robots perform simpler tasks in them).

Robotic entities can be controlled by (belong to) different physics engines. The implementation assumes that robots controlled from separate physics engines can't physically interact (i.e. collide). This assumption only affects physical interaction, whereas any other interaction between robots, i.e. communication, is not affected. Sensors that allow robots to observe the environment, such as cameras and proximity sensors, can still perceive robots handled by different engines. The separation of the simulation computations performed by multiple engines affects only the physics rules used to update the simulated objects.

Another feature is that objects can change dynamically the physics engine they belong to during the simulation, i.e. it is possible to "move" an entity to a different engine if that's required. This gives the ability to the user to improve the simulation performance. We can have robots simulated in a simpler physics engine when a simple navigation is needed, and by the time a robot enters a room that a more complex task has to be done, it can switch it to a different physics engine. This process is done internally on ARGoS - the entity is not aware of the change.

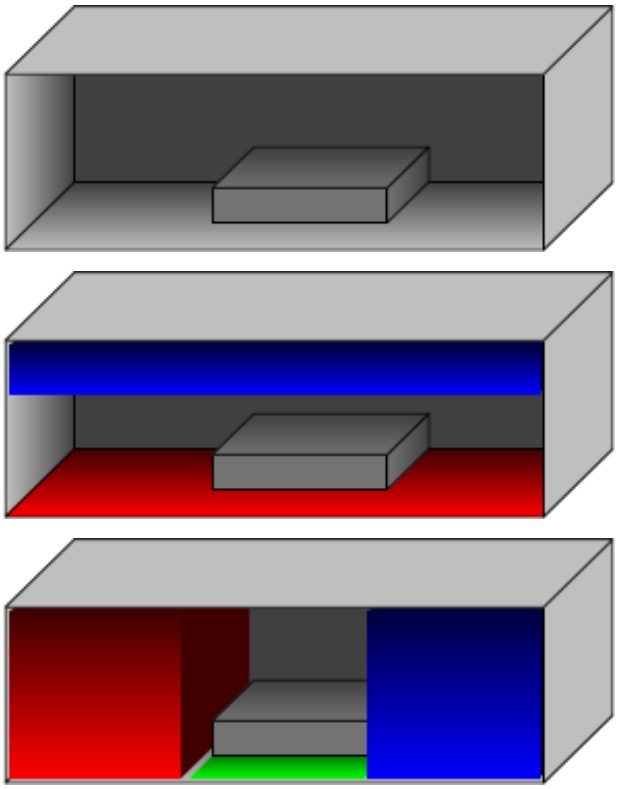

***Figure 7**: multiple physics engine. The simulated space can be divided and managed by separate engines. Each color above indicating a unique engine controlling that area. Reprinted from ARGoS website with permission.*

The multiple physics engine approach in ARGOS has many significant advantages. First of all, it allows users to integrate any well-known physics engine libraries (ODE, Bullet etc.) It is also particularly useful when a custom engine, specifically customized for the simulation needs has to be added. Moreover, keeping the entities on separate engines reduces the processing needed in some common situations, for instance when we check for collisions. In a swarm with a thousand robots, if we carefully allocate agents across multiple engines - separate for wheeled and flying robots for example - the execution of simulation can speed up considerably. Dividing the simulated environment into non-overlapping subspaces, where each of them is managed by a different physics engine is a feature unique in ARGoS.

2.3.2 Modularity

ARGoS was designed following a modular architecture - every component comes as a plug-in loaded at runtime. Users can add new functionality varying from sensors and robots to physics engines. A visual representation of the architecture can be found on the following figure.

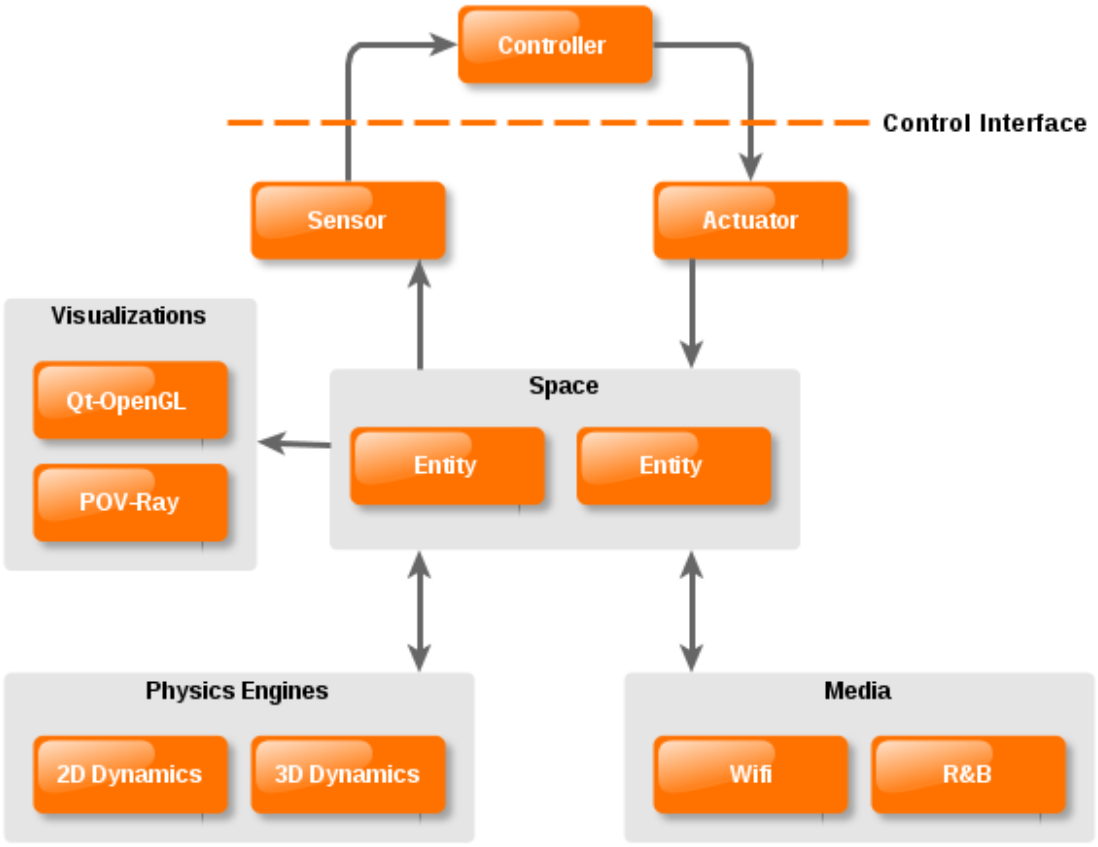

***Figure 8**: Modular architecture. Everything is a plugin in ARGoS. Reprinted with permission from ARGoS website*

The simulator uses a set of interfaces and a global simulated space to bring together all those different elements. That space is where every entity exists. It's also responsible for storing their position, orientation (i.e. physics related information) as well as any other information not related to physics, as for instance visualization information. The physics engine is responsible for updating the position and orientation of the entities. A set of design choices were made to ensure the global space integrity, as stated by Pinciroli et al (2012).

- The global space status is updated synchronously (discrete-time simulation). On every simulation step the physic engines are called to update the status of their entities. Thus, regardless the fact that each engine can be run on a different thread, ensures that our experiment is on sync.
- Entities can only belong to one physics engine at a time.
- Entities can interact with entities on the same physics engine.

- Immobile entities (like walls, obstacles etc.) can belong to multiple engines to ensure the consistency of the environment.

All the interactions between the robots and the world are done, using sensors and actuators. Sensors are responsible for perceiving the environment, and they achieve that by reading data from the global space. Reading data from that space ensures data integrity across multiple physics engines. Actuators on the other hand are responsible for updating the robot's state. After the state change ARGoS updates the global space to reflect the changes. A top level approach of the main simulation loop can be found on the following pseudocode.

```
1: Initialize
2: Visualize the simulated 3D space
3: while experiment is not finished do
4:    for all robots do
5:       Update sensor readings
6:       Execute control step                    } sense+control
7:    end for
8:    for all robots do
9:       Update robot status                     } act
10:   end for
11:   for all physics engines do
12:      Update physics                           } physics
13:   end for
14:   Visualize the simulated 3D space
15: end while
16: Cleanup
```

***Figure 9**: Simplified simulation loop pseudocode. Reprinted from ARGoS presentation paper by Pinciroli et al (2012).*

2.3.3 Multithreaded architecture

One of the most common issues robotic simulators face is scalability, especially when a robotic swarm of many agents needs to be simulated. Modern computer systems nowadays, commonly offer multi-core architectures, thus ARGoS was designed to take advantage of that feature. This is a major breakthrough since most simulation engines are still single-threaded.

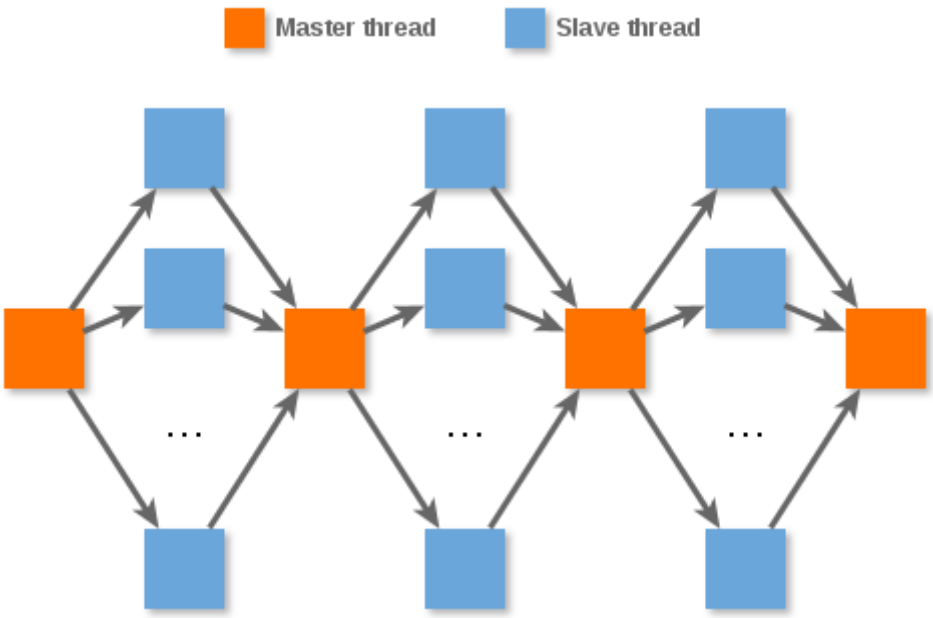

***Figure 10**: Multithreaded architecture. Reprinted with permission from ARGoS website.*

Users developing their simulations on ARGoS, can choose the number of threads they want to use when executing an experiment. The simulator core separates threads into master and slave, with the master thread responsible for distributing the tasks to the slave threads. Each slave thread is responsible for a single plug-in (sensor, actuator, entity, physics engine etc.). There are two methods available for task allocation among the threads. The first one, called scatter-gather, provides better results when we have tasks with similar computational costs, such as a swarm of homogeneous robots. It divides the tasks evenly and it's the default method used. Task assignment is calculated before the experiment and ARGoS recalculate it only when robots are added or removed from the simulation. The second, called h-dispatch, is used when we have diverse computational costs, for example an experiment with different robots (heterogeneous swarm). This method allocates a task to a thread as soon as the thread is idle. The main thread the simulation is running is responsible for task allocation across multiple threads. Since the thread allocation is implemented in the core of ARGoS the user doesn't have to change his implementation in order to take advantage of multi-threaded execution.

2.3.4 ARGoS Loop functions - Controlling and Monitoring the Experiment World.

Loop functions are a very important aspect of ARGoS simulator. They are function hooks, defined by users, in important simulation points, a common approach in many simulators. Those functions enable the developers to add extra functionality in their experiments. Code on a loop function can be executed at the initialization and the end of the experiment as well as before and after each simulation step.

Loop functions provide access to the entire simulation and allow any modification of the former. Users can use these functions to extract statistics and information about the simulation, or write data to a file for future reference/usage. In addition to that it is possible to remove or add entities to the simulation or change their state (i.e. changing the color of the LED lights a robot is equipped with etc.).

2.4 ARGoS Simulation - Experiment configuration

When an experiment is executed in ARGoS two inputs must be provided. The first is the compiled user code specifying the swarm's behavior and the second is the experiment configuration xml format file (.argos file extension). The configuration file includes all the necessary information for the simulator in order to set up the simulated environment and add all the entities. Configuration file can be broken up into different sections based on whether they are optional or mandatory. Mandatory sections are the framework section, controllers, arena, physics engine and media. Optional sections are visualization and loop functions.

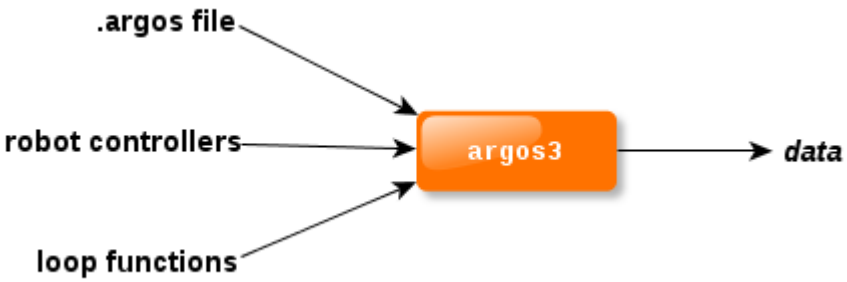

***Figure 11**: Input need to run an experiment. Reprinted with permission.*

2.4.1 Framework Section

The framework section contains the internal parameters ARGoS needs for the experiment, such as the number of threads, the duration of the simulation etc. The layout of the section can be seen below (**Figure 12**: Framework section from XML configuration file.).

```
<framework>
  <system threads="0" />
  <experiment length="0"
              ticks_per_second="10"
              random_seed="124" />
</framework>
```

***Figure 12**: Framework section from XML configuration file.*

The tag named system accepts two parameters, named threads and method (both optional). The thread attribute defines how many threads will be used in the simulation. Specifying 0 threads or leaving it unspecified assumes a single thread of execution (default value). The method attribute specifies the method we use to assign tasks in our threads. ARGoS supports two methods balance_quantity for using the scatter-gather method and balance_length when the h-dispatch method is preferred. We should mention that the system tag is optional and if it's not specified the simulator will only use one thread.

The experiment tag is a required section, as it contains all the information needed to run an experiment. It has three attributes, length, ticks_per_second and random_seed. Length is self-explanatory -it specifies the time the experiment will run in seconds. A zero value will keep our experiment running until the simulator GUI is closed or terminated through loop functions. Ticks_per_second specify how many ticks -or simulation steps- ARGoS executes every second. A value of 5 for example, will execute 5 steps a second and if the experiment runs for 60 seconds (length = 60), 300 ticks will be executed in total. It should be noted that physics engines can have different number of steps every second (defined in the physics engine section) allowing physics updates between the simulation steps. The random_seed attribute is optional and specifies the seed used by the random number generator. If not defined or given the value 0 the internal clock time will be used as random seed. This attribute is important when running simulations since the simulation

conditions can be the same across many runs of the experiment, thus allowing reproducibility

Lastly, the profiling tag allows ARGoS to keep information regarding memory and CPU. Information can be written into two different forms: human readable or machine readable. Each time the experiment runs the file can be truncated, if specified.

```
<framework>
  ...
  <profiling file="profile.txt"
             format="human_readable"
             truncate_file="true" />
</framework>
```

***Figure 13**: profiling tag. Reprinted with permission.*

2.4.2 Controllers section

This section contains all the robotic controllers used in the experiment. In the following snippet you can see the controllers defined for this thesis. There is no limit in the number of controllers one can declare.

```
<!-- *************** -->
<!-- * Controllers * -->
<!-- *************** -->
<controllers>

  <epuck_controller id="fdc" library="build/controllers/burning_forest/libburning_forest.so">
    <actuators>
      <leds implementation="default" medium="leds" />
      <range_and_bearing implementation="default"/>
    </actuators>
    <sensors>
      <range_and_bearing implementation="medium" medium="rab" show_rays="false"/>
    </sensors>
    <params spread_time="50" max_burn="100" spread_distance="12"/>
  </epuck_controller>
.......

</controllers>
```

***Figure 14**: Controllers section. An epuck robot is declared with the appropriate sensors, actuators and controller class.*

The tag for each controller is defined by the user and it has to be registered in the system as part of the controller code using the REGISTER_CONTROLLER macro. The

id and library attributes are mandatory. The first one declares a unique identifier for the controller allowing users to use the same code with different controllers. The second specifies the full path to the compiled controller code for the robot.

As already mentioned, every robot interacts with the environment using sensors and actuators. The sensors and actuators used must be declared on the appropriate sections. Each of them has a different configuration and different attributes must be specified. Same sensors or actuators may have multiple implementations which have to be specified using the implementation attribute. For example, an omnidirectional camera may have a generic implementation, an implementation for a wheeled foot-bot and one for flying eye-bot.

Last subsection of the controllers section is the params section. This section is optional and contains everything needed to configure our controller. A completely custom xml schema can be used or only xml attributes under the params tag - it's up to the user to choose based on his needs.

2.4.3 Arena section

Arena section contains all the needed information to set up the simulated environment. Users specify the objects that will populate the arena and how they are distributed.

```
<!-- ********************** -->
<!-- * Arena configuration * -->
<!-- ********************** -->
<arena size="5, 5, 2" center="0,0,1">
  <floor id="floor"
         source="loop_functions"
         pixels_per_meter="50" />
  <box id="wall_north" size="4,0.1,0.5" movable="false">
    <body position="0,2,0" orientation="0,0,0" />
  </box>
  <box id="wall_south" size="4,0.1,0.5" movable="false">
    <body position="0,-2,0" orientation="0,0,0" />
  </box>
  <box id="wall_east" size="0.1,4,0.5" movable="false">
    <body position="2,0,0" orientation="0,0,0" />
  </box>
  <box id="wall_west" size="0.1,4,0.5" movable="false">
    <body position="-2,0,0" orientation="0,0,0" />
  </box>

  <!--
      Randomly distributing entities
  -->
  <distribute>
......
  </distribute>

</arena>
```

***Figure 15**: Arena section*

In the above snippet an arena is declared bounded by four walls. The distribute tag is used to scatter entities around but they can also be placed in specific positions if needed.

The distribute tag contains three child elements: entity, position and orientation. The entity section contains the type of entity to distribute (in our case the e-puck entity). The two attributes must be given values, the quantity that specifies the number of entities ARGoS will try to distribute and the max_trials attribute. Max trials value is the maximum number of times the engine will attempt to place the entity in the world. If that value is reached without success, the engine will give up with an error. Inside the entity section we specify the added entities giving an id for each one and specifying the appropriate controller. In the following code snippet, 100 e-puck robots are placed in the arena with ids fdc_0, fdc_1, fdc_2…. fdcf_100. The base number used in the id can be changed by supplying the optional attribute base_num.

```
<distribute>
  <position method="uniform" min="-1,-2,0" max="2,2,0" />
  <orientation method="gaussian" mean="0,0,0" std_dev="360,0,0" />
  <entity quantity="100" max_trials="100">
    <e-puck id="fb">
      <controller config="fdc" />
    </e-puck>
  </entity>
</distribute>
```

***Figure 16**: e-puck distribution code*

Orientation and position sections specify how those two attributes are calculated for each entity placed in the environment. The method used to distribute them is given using the attribute with the same name. There are four methods available to the users, according to ARGoS website:

- **Uniform**: The values used are chosen from a uniform distribution. We use the min and max values to specify the range of the values.
- **Gaussian**: A Gaussian distribution is used to choose the values. The attributes mean and std_dev are used for the mean and standard deviation.
- **Constant**: This method allows us to set the value to a constant. We represent the value using the values attribute, as a triplet x,y,z.
- **Grid**: This method is used to place entities in a grid. The center of the grid is specified using the center attribute, the distances attribute declares the distances between the robots on the x,y,z axis and the layout value sets the number of robots per side on the axes.

Summarizing the above, ARGoS multi-robot simulator, capable of simulating large robotic swarms was presented. The design choices made on while developing the simulator are explained as well as how each of them separate ARGoS from other robotic simulators.

ARGoS modular approach allows users to freely modify the simulation. Adding new robot models, sensors or actuators is easily achieved since every one of them is a single module that researchers can use on their experiments or exchange them, promoting cooperation. Global space is responsible for combining the different modules and update their state on every simulation step.

ARGoS can use multiple physics engine on a single simulation, dividing the environment into sub-spaces each controlled by a single engine. This ability is the most

unique feature of ARGoS separating it from the rest of the robotic simulators available. Each physics engine can be tuned separately, according to the experiments needs improving the speed of the simulation.

Multi-threaded approach takes advantage of modern processors multi-core architecture. Thread allocation method differs based on whether the simulated swarm is homogeneous or heterogeneous. The former can use the scatter-gather method to evenly distribute the tasks between threads. The latter uses the h-dispatch method allocating task on a thread as soon as it's idle. According to Pinciroli (2012), ARGoS can simulate a swarm of 10.000 robots in 60% of real time on 2D simulation and in real time when 3D simulation is used.

Additionally, the ARGoS experiment configuration is presented. The most important sections of the configuration file are explained. The framework section sets up all the internal parameters for ARGoS, such as the number of threads. Controllers contains a list of user-defined controllers and they configuration for each one of them. Lastly arena contains the entities to add at the simulated environment and their position and orientation.

For all the reasons stated above, the ARGOS platform was selected as the tool of choice to simulate the wildfires scenario.

3. The Problem: Wildfires

3.1 Problem overview

Wildfires are catastrophic events that burn down millions of acres of forests every year around the world, according to data from NOAA and EFFIS (European Forest Fire Information System). A wildfire differs from other fires in its size, the fast speed it is spreading, and the ability to pass inflammable terrain (such as roads, fire breaks and rivers) as well as the unexpected direction change. Various prevention, detection and suppression techniques have been developed over the years, with international wildfire experts insist that further research and development is needed.

Prevention systems, are the techniques used to prevent the start of a fire or reduce its size and spread. The most common prevention method, is controlled burn: vegetation is burned under control to reduce the risk of a wild fire. In addition to that, old trees are cut down in order to reduce the fuel material available. Lastly fireproof materials are used for construction of houses and shelters in an area that a wildfire can occur. This can help reduce a wildfire occur due to human caused source, such as accidental ignition.

Detection is a key factor to fight wildfires. Main focus of detection systems is accurate results, early response, consistence between day, night and different weather conditions as well as prioritizing dangers. Detection systems include fire watch towers, satellite surveillance, cameras and smoke detectors. Human detection has been proven the most efficient method in terms of early detection and danger classification, although human resources are limited and hard to use during nighttime.

Suppression depends on the area and the technologies available at place. Most common method on undeveloped countries and places where firefighting supplies are limited is sand and sticks to beat the fire. More advanced techniques include fire retardants and water dropped by aerial vehicles (sometimes unmanned). It should be mention that complete suppression is not always the goal, instead the aim of firefighters is the suppression of the fire before it gets out of control. Firefighting wildfires is considered a dangerous task for humans. Not only the fire can change direction unexpectedly, leading to a dangerous situation for personnel, but heat and smoke can cause disorientation leading to entrapment between the flames with possible injuries or even death. Additionally, extreme conditions can be found in a burning forest. Those conditions can be dangerous and sometimes lethal for human forces. Robots can navigate through those conditions

without any issues. Also, the swarm is waiting in the forest for deployment in predesignated hubs, reducing the time needed to reach the fire drastically.

3.2 The Proposed solution

The solution proposed on the wildfire problem is using a robotic swarm to take out the fire or contain it until land or aerial forces arrive, thus minimizing the damage. The approach considers that a number of robots, equipped with fire extinguishers and a tank that carries fire retardants is used. There are two kinds of robots that differ in the amount of fire retardants their tank can carry as well as their top speed. The first variation is a fast moving robot, carrying a smaller tank thus less amount of the substance used to put out a fire. The second variation is equipped with a larger tank allowing to stay on the field more time and spray larger quantities but a drawback is its reduced speed. That variation allows fast intervention preventing a wider spread while more heavy robots approach the fire.

Initially robots are located in a hub placed inside the forest. That hub serves as a staging area for the swarm while it's idle. Robots located inside the hub are also refilling their tanks with fire retardant. One of the assumption made is that robots can be anywhere in the hub in order to refill their tank - there is no specific position for them.

Upon the event of a fire a detection system already in place will activate the swarm and provide the coordinates of the initial point of fire. Immediately after activation, the swarm starts navigating towards the fire. When a robot arrives either at the initial spot, or encounters a fire on its way, it will start firefighting. On firefighting mode, a robot is immobile and starts spraying fire retardant around it. That way not only the fire spot is sprayed but also the surrounding vegetation preventing further spread.

When on firefighting state there are two possible outcomes: either the fire is taken out on that spot or the tank is empty. If the first case is true (meaning there is still fire retardant on the tank) the robot continues towards the initial spot. If the second case is true, the robot returns to the hub in order to refill and start over - if the fire is still burning.

The mission is considered accomplished in two cases. The first case is when the fire is completely taken out. The detection system informs the robots that no fire is burning, and the robots return to the hub. Second case is when the swarm manages to put the fire under control, either letting the area on fire completely burn out or in case reinforcements

arrive take it out. Robots return to the hub using the GPS coordinates already stored in each robot.

The behaviour of each robot is controlled by reactive architecture. There are no local or global communication, thus every robot acts based on its own sensory input and perception of the environment. Robots move towards their goal until their sensors detect an obstacle, reacting on that input and adjusting their course in order to avoid it. Additionally, when sensors detect a tree on fire, robots react by starting their firefighting sequence.

Considering that a forest is hard to map, with vegetation developing over time, trees falling or ground shifting due to rain navigation had to be done without knowledge of the area. A hill climbing approach has been adopted to find the path to the fire location. Robots always try to move closer to the fire, and if an obstacle is detected they will adjust their course aiming for the lowest derail from the course. In order to escape valleys and ridges, a random value, based on the sensor readings, is used to adjust the course when avoiding the obstacle.

The swarm is designed as a finite state machine. The states and the transitions can be seen on the following table:

Current State	Input	Next State	Output
Idle	Fire detected -tank full	On Duty	Start navigation towards fire
	Fire not started	Idle	none
	Fire detected - empty tank	Idle	Refill tank
On Duty	Fire found	Fire Fighting	Starts spraying fire retardant
	Fire not found	On Duty	Navigating towards fire
	Fire taken out	Returning	Navigate back to the hub
Fire Fighting	Fire burning	Fire Fighting	Spraying fire retardant
	Local fire taken out	On Duty	Navigating towards fire
	Global fire taken out	Returning	Navigate back to the hub
	Tank empty	Returning	Navigate back to the hub
Returning	Arrived at hub	Idle	none
	Not at hub	Returning	Navigate back to the hub

***Table 1**: State and transitions of finite state machine*

There is no memory of any previous states and there is no planning of the next movement. Every transition is done based on the sensory input of each individual, its own perception of the environment and the proximity to the goal. That approach can lower the manufacturing and replacement cost of a robot since the memory requirements are low.

3.2.1 Advantages and limitations

The majority of advantages and disadvantages of swarm robotics that were presented on the second chapter, can be found in the context of the specific experiment. Firstly, the proposed solution can adapt to different environments and any changes that modify it. A forest area can change over time. Flora can develop or cleared changing the landscape thus making harder to create a clear path to an area. Moreover, a forest on fire can experience massive changes over a small period of time. The swarm proposed can handle those changes without any issues. GPS coordinates are a point of reference that remains intact and local navigation with obstacle avoidance can ensure that we will eventually reach the target.

The foot-bot swarm is also resilient. A single individual robot lost in the process is not preventing the system from completing its task. Foot-bots are autonomous entities operating inside the swarm and they do not rely on each other. Definitely a robot failure will have an impact - affecting the speed the fire will be taken out, unable to completely take out the fire etc.- but the main goal will not be affected.

The swarm designed and developed in this thesis, has the characteristic of distributed sensing as described on the previous chapter. Each robot has its own set of sensors that allow it perceive the world around it. There is only one centralized sensor, the fire detection system, alerting the robots for the fire and informing them of the corresponding the GPS coordinates. Every robot is acting on the same fire, approached from a different side based on the path chosen to reach it during the navigation step. Multiple fires or multiple hubs would allow the swarm to act on multiple places at the same time.

The forest fires swarm shares the limitations common to most swarm systems stated in section 2.1.1, such as uncertainty of other robots intentions and interference between each other. One important limitation in this particular system, is that robots interfere with each other, since every robot acts as an obstacle to all others Thus, the larger in number the swarm is, the more difficult is for robots to reach their target, i.e. the fire's

initial point is difficult to reach not only due to environment obstacles, but every stationary robot fighting the fire is adding one more obstacle to the path towards the fire. That can lead to situations where only a small amount of foot-bots is spraying fire retardant while the rest of them try to navigate closer, i.e. a congestion situation. The scenario described can occur in bottleneck areas as well - i.e. areas with the same narrow entrance and exit.

Another limitation is that there is no local or global communication introduced. The ability to send and receive messages would allow robots to coordinate and overcome obstacles or avoid areas that lead to a dead end. Current design does not support communication, forcing robots to interact with each other as obstacles. That can lead to potential entrapment, taking longer routes or not being able to reach the goal at all. However, adding communication capabilities would interfere with the reactive behavior of the robots. Robots will cease to react on the input from their sensors and will require planning of their actions based on actions of nearby robots. Also, communication hardware could potentially increase the manufacturing and maintenance cost of the swarm. Considering that robots will operate in dangerous and harsh environment that could lead to an economically non-viable solution.

3.3 ARGoS Simulation of Wild Forest Fires

This section presents the implementation of the system described in the previous section. The presentation begins with a description of a tool created to assist users and guide them through the creation of the simulation configuration file. The simulated entities used on the research are also presented. Furthermore, the simulated environment and the simulation control logic are explained in detail. Lastly an extensive analysis of the swarm implementation and the experiment configuration.

Implementation is made with the assumption that sensory input is noise free. Although ARGoS supports artificial noise generation for all the sensory input in the current research it is not used. That results in clean input data, easy to interpret and understand. In a real life scenario sensors return their data with environmental noise. More specifically in case of a wildfire in a real forest, smoke can interfere with the infrared or ultrasound sensors used to avoid obstacles. Additionally, low hanging leaves or tall grass can be perceived as an obstacle forcing the robot to adjust its course. That noise must be identified and cleared out of the input signal. Considering this is a simulation only, researching

whether a swarm of robot is a viable solution, noise is not take into account. On the same context the assumption that ground is flat with no cliffs is made.

3.3.1 ARGoS simulation file creator

One of the core files needed for the simulation is the configuration file, an xml formatted file. Creation of that file can be a time consuming task, especially on experiments with complex environments, due to the amount of properties we can use in each tag. In order to speed up the experiment configuration file creation, as well as reduce any schema errors, a tool was designed and developed using java.

The ARGoS simulation file creator tool guides the user through the entire process allowing him to enter values for all pre-defined tags and attributes as well as define its own. At the end of the process the file is saved on a user defined location with the correct extension. The class diagram is shown on the following image.

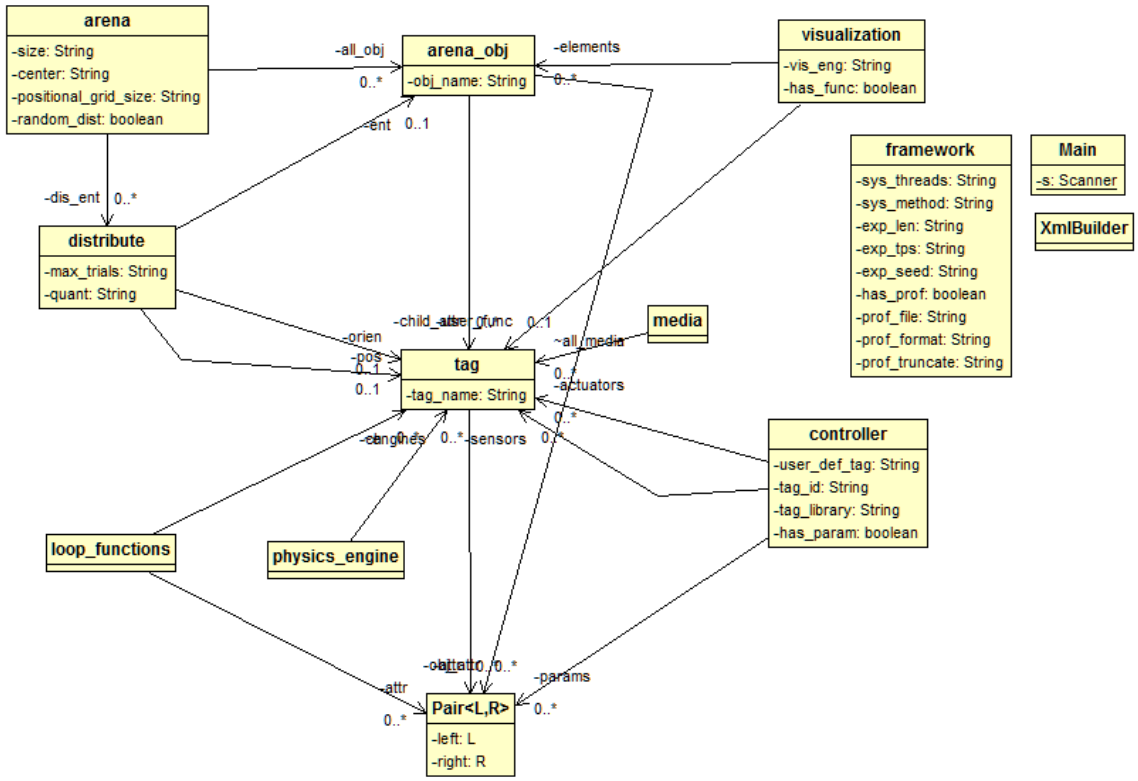

***Figure 17**: Class diagram*

A crucial part of the design of the mentioned tool, is that users can define their own xml tags and attributes. For instance, an experiment can have a structure defined, were the

various fields are populated with values specified in the xml configuration file. Since that xml schema is user defined, a class was needed to allow the users define their own schema. The classes tag and pair were developed to address that problem. The pair class holds an attribute and its value while the tag class holds a custom tag name and a list of all the attributes. An image of the program execution can be seen below.

```
************Argos Configuration Creator************
*                                                   *
*          Configuration file creator for           *
*                      ARGoS                        *
*                                                   *
*****************************************************
Configuring framework section........
**************************************
Configuring system tag.......
Insert threads number:
1
Select method(type correct number):
1. balance_quantity (default)
2. balance_length
2
Configuring experiment tag......
**************************************
Insert expirement length
0
Insert ticks per second
10
Insert random seed
4
Do you want to include profiling (Y/N)
n
```

***Figure 18**: Execution instance. The application guides the user through the configuration file creation*

The rest of classes are self-explanatory. Each section of the configuration file has its own class. After the initialization of a section, the latter is added to the xml document handled by the XmlBuilder class. At the end of the procedure the file is saved on a specified location.

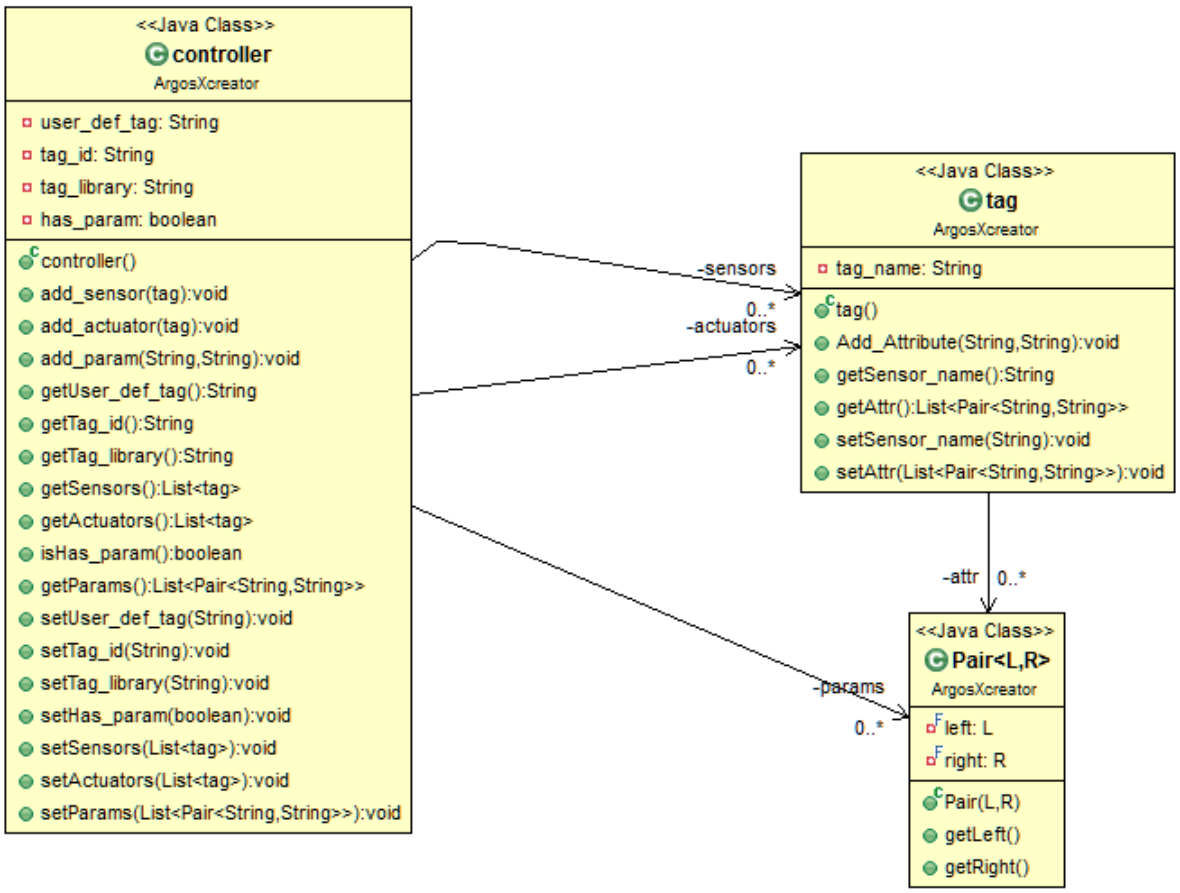

***Figure 19**: The controller class, responsible for fetching and storing the information needed on the controllers section. A fine example of the tag and pair class usage.*

3.3.2 Description of the entities in the environment

In order to simulate the forest fire system, we need to provide an environment that contains the following entities:

Trees, are simulated using e-puck robots, with each one representing a tree. In our initial efforts, collision between robots belonging to the swarm and e-puck robots, resulted on a trees moving away from their initial location. The reason for that is that both robots have similar mass, thus the force applied by the physics engine in case of collision dislocated the e-puck. That behavior is not acceptable, considering that trees can't be moved in case of collision. In order to solve that problem, the mass of the e-puck is greatly increased, resulting in a non-noticeable movement of the tree. The magnitude of the force generated by the impact is not big enough to move an object with that big mass around, but still has a small effect since forces can't simply disappear.

Robots, are simulated using the foot-bot model, already implemented in ARGoS. The foot-bot supports a variety of sensors and actuators. Actuators simulated include a differential steering actuator used for navigation and leds actuator needed to light the leds onboard of foot-bot. Simulated sensors include a proximity sensor used on obstacle avoidance, a ground sensor reading the color of the ground under the robot and a light sensor detecting light sources in the environment. The ground sensor is used to let the robot know whether or not is inside the hub. The light sensor is needed to achieve phototaxis and help the robot return to the hub.

Hub, is simulated by simply changing the color on the area designated as staging point. The color of the ground is changed to gray while the color on the ground outside of the hub is white, allowing clear detection using the ground sensor.

Obstacles, present on the simulation are 4 rectangles used as walls around the arena. Those walls prevent the robots from getting out of bounds while navigating. Each wall is declared as immovable object in order to avoid dislocation of the wall in case of collision.

Fire spreading model, is implemented using a heat variable declared on each tree. The initial value is set to 0 and the loop functions controlling the experiment ignite the tree. The incrementation of the heat on each simulation step, after the initial ignition, is done on the tree level.

3.3.3 The simulated environment

The simulated environment in ARGoS can be created using the objects already implemented in the simulator, including rectangles, cylinders etc. During the design phase of the environment it was made clear that provided objects can populate the space and simulate the trees in a forest, but the interaction between them could not be implemented easily. Interaction between objects is needed to start the fire and spread it around the forest.

For that reason, the e-puck robot provided was used to simulate the trees. The e-puck is an educational robot that, apart proximity sensors, is equipped with local communication device. Local communication is important while implementing a fire simulation. E-puck robot can inform the robots around that it’s on fire and they should start burning too.

The algorithm used to simulate a wildfire is pretty straightforward. The first robot placed in the environment is chosen as the initial point of the fire. This is done during the initialization phase of the trees and uses the id of the entity to identify the initial location the fire starts. Afterwards, a model is used to spread the fire to nearby trees. Spreading is

achieved using local communication with the range and bearing sensor/actuator. The pseudocode of the algorithm is the following:

```
Initialize;
On every simulation step {
    if (burning){
        current_temperature++;
        if (current_temperature >= spread_temperature){
            Announce(spreading_fire);
            leds.color(red);
        }
        else if (current_temperature >= burnout_temp){
            leds.color(black);
            burning=false;
            burned=true;
        }
    }
    else if (nearbytree.burning){
            leds.color(yellow);
            burning=true;
            current_temperature++;
        }
    }
```

***Figure 20**: Heat model pseudocode*

The model used to simulate the fire spread is based on the heat model originally presented by Rothermel (1972). Each simulation tick increases a burning tree's temperature. When the temperature reaches a specific point, fire spreads to trees within a specific distance and when a certain threshold is reached the tree stops burning as it is completely burned down. The described model is a simplified heat model since a number of real world parameters are not taken in consideration. There are more complex models proposed taking in account the effect of the wind, the slope of the ground, moisture etc.

In order to put nearby trees on fire, the range and bearing sensor/actuator is used, that is provided by Argos. The former is used -with the corresponding media- to receive messages while the later to receive messages. Messages transmitted using the range and bearing also transmit the position of the sender in relevance with the receiver. Using that position the distance is calculated and the appropriate trees are set on fire. During the initialization of the environment the message each e-puck transmits is initialized to "Not-Burning". Every robot keeps transmitting that message on each simulation step. When the

spread threshold is reached, the e-puck on fire transmits a “Burning” message. When a robot receives a “Burning” message, checks the distance between itself and the sender as well as if it’s already burning or it’s burned out. If it’s not on fire, it starts burning and follow the model described above.

The visualization of the burning trees was achieved using the leds placed around the e-puck. When the robots are on the initial state the leds are turned on green. Catching fire will result on yellow leds while reaching the spreading point will change the color of the leds to red. Finally, when a robot is burned out its led lights are illuminated black - a color available on simulation - as an indicator. Leds are the only visual representation of the fire on the simulated environment. A snippet of the simulated burning forest can be seen below.

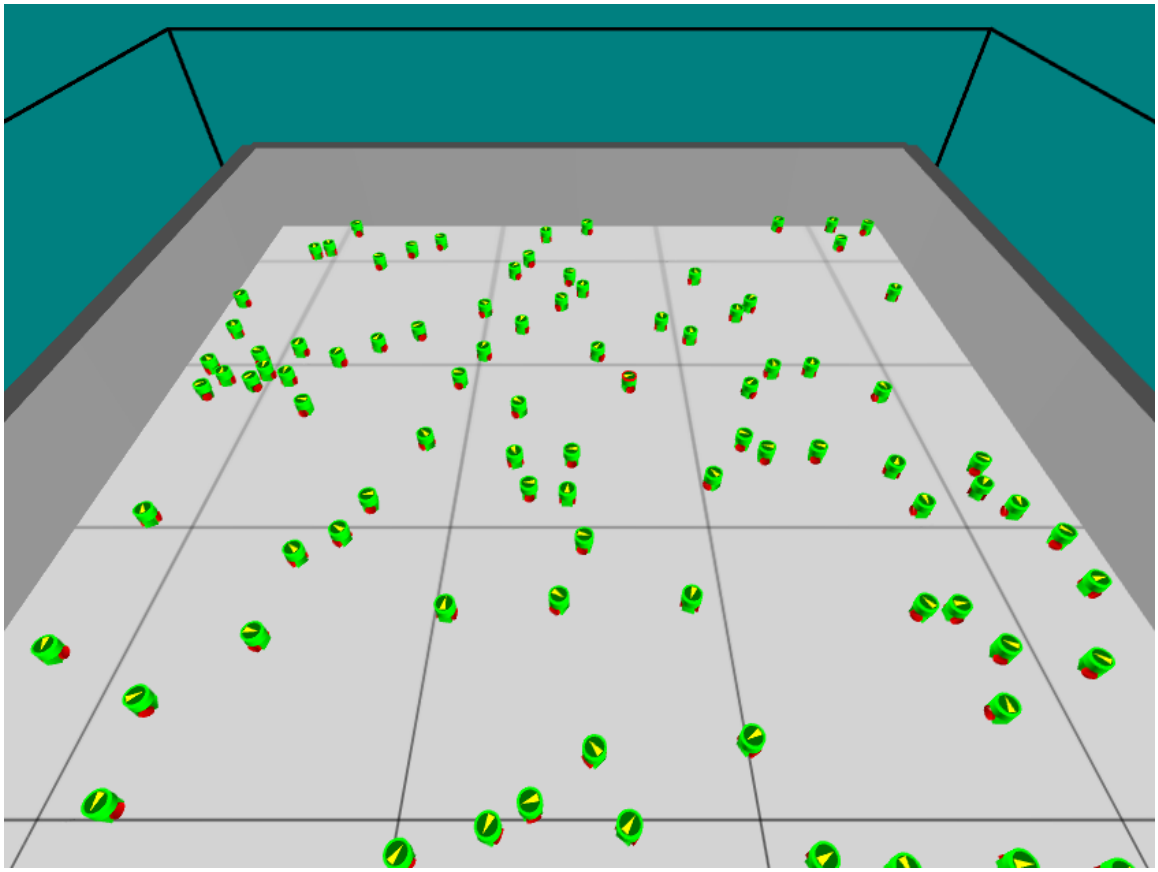

***Figure 21**: Simulated forest on the first simulation tick. The ignited tree can be seen in the middle with its led lights having red color*

3.3.4 Controlling the experiment

The loop function developed is used to control the flow of the experiment and contains the logic of our simulation. Loop function logic is executed in three points on the simulation. First one is during the initialization phase. The other two are executed before the simulation step and after the simulation step.

In the simulation, during the initialization of the environment space, the simulated floor color is set. In order for our swarm to be aware when it's inside the hub or outside of the hub, different floor colors were used. A pointer to the floor entity is passed to the GetFloorColor function. That function is using the position on plane property to find the correct color of the pixel. White color represents the forest and a gray pixel represents the hub.

```
CColor Firefight_loop_functions::GetFloorColor(const CVector2& c_position_on_plane) {
   if(c_position_on_plane.GetX() < -1.0f) {
      return CColor::GRAY50;
   }
   return CColor::WHITE;
}
```

***Figure 22**: Setting the color of the hub floor to identify it using the ground sensor*

The pre step method is where the main interaction with the simulation is happening. This method acts as the fire detection system. An array containing a pointer to each e-puck entity is iterated to find an e-puck that has the burning flag set on true. When a burning tree is found, the swarm is activated and the GPS coordinates of the fire initial point are passed on each robot of the swarm.

Lastly during the after step, executed at the end of simulation step, searches for at least one tree burning. If at least one e-puck is retrieved with a burning state, the experiment continues. In case no trees are burning the swarm is informed that the fire is out and navigates back to the hub. It is worth mentioning that fire is ignited on the first simulation tick on the pre step, thus preventing the after step from setting the swarm to return to hub state.

```
void Firefight_loop_functions::PreStep(){
    CSpace::TMapPerType& m_cEpucks = GetSpace().GetEntitiesByType("e-puck");
    CSpace::TMapPerType::iterator iter = m_cEpucks.begin();
    while(iter != m_cEpucks.end()){
        /*get handle to e-puck entity*/
        CEPuckEntity& cEpuck = *any_cast<CEPuckEntity*>(iter->second);
        Cburning_forest& epController = dynamic_cast<Cburning_forest&>(cEpuck.GetControllableEntity().GetController());
        if (epController.Burning() && !RobotsAreActivated){
            CVector2 ePos;
            ePos.Set(cEpuck.GetEmbodiedEntity().GetOriginAnchor().Position.GetX(), cEpuck.GetEmbodiedEntity().GetOriginAnchor().Position.GetY());
            CSpace::TMapPerType& m_FootBots = GetSpace().GetEntitiesByType("foot-bot");
            for(CSpace::TMapPerType::iterator i = m_FootBots.begin(); i != m_FootBots.end(); ++i){
                CFootBotEntity& cFootBot = *any_cast<CFootBotEntity*>(i->second);
                footbot_firefight& FbController = dynamic_cast<footbot_firefight&>(cFootBot.GetControllableEntity().GetController());
                FbController.CallOfDuty(ePos);

            }
            RobotsAreActivated = true;
            break;
        }
        else if(RobotsAreActivated) {
            RobotsAreActivated = false;
        }
        ++iter;
    }

}
```

***Figure 23**: Pre step function. If a robot is found burning, we set the swarm to "activated"*

3.3.5 The Firefighting swarm

The core of the current thesis is the design and the development of the firefighting swarm. The swarm consists of foot-bots, a robot already implemented in ARGoS. **Figure 24** depicts the robot used.

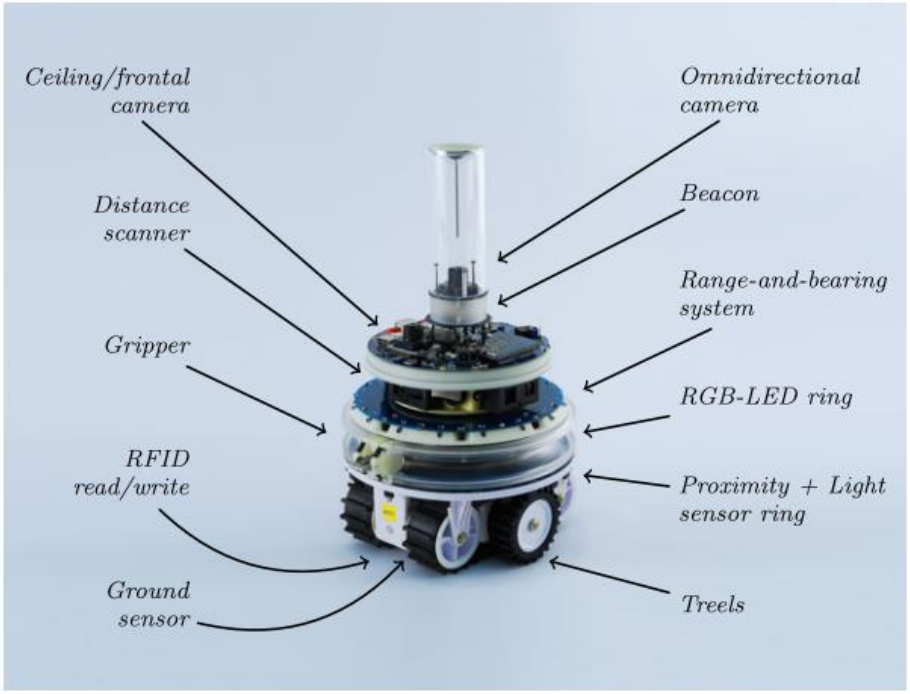

***Figure 24**: Foot-bot model used. Reprinted form ARGoS presentation paper by Pinciroli et al (2012)*

An assumption was made that the foot-bot is modified to carry a water or fire retardant tank as well as a spray system around its external ring. No changes were on the actual implementation of the foot-bot entity to simulate the firefighting system. The needed system was programmatically added to the simulation as a custom structure. Foot-bot comes with a full set of sensors and actuators. Of all those we use the proximity sensor, the light sensor, the ground sensor, the differential steering actuator and the leds actuator.

The differential steering actuator is used to move the robot around the simulated environment. The actuator allows users to set the speed on left and right wheels independently. Utilizing that ability, foot-bot can take turns that can be either soft or sharp, by applying force only to one wheel or opposite forces to both wheels respectively. A structure was created to store all the values needed for navigation. An enumerator was declared to indicate the state of the robot, going straight, soft turning or hard turning. Additionally, the max speed value is stored and the angle thresholds to take a turn - and how sharp it's going to be - as well as the threshold to keep a straight course. The values for those properties are specified on the experiment configuration file, allowing further testing when speed and angle threshold are changed, without recompiling the source code.

```
<diffusion straight_angle_range = "-5:5"
           delta = "0.1" />
<robot_turning Hard_Turn_Angle_Threshold = "70"
               Turn_Angle_Threshold = "30"
               Straight_Angle_Threshold = "10"
               max_speed = "30" />
```

***Figure 25**: Diffusion and turning parameters declared on the configuration file.*

Obstacle avoidance while navigating is achieved using the proximity sensor equipped on foot-bot. On every simulation step the readings of the sensor are fetched in a vector and their sum is calculated. If the angle of the vector is small enough and the closest obstacle is far enough, the vector is ignored and the robot goes straight, otherwise the vector is returned and the turn angle is calculated. The vector angle is compared against a user defined angle.

The returned vector is used to turn the robot in order to avoid any obstacles found. A boolean variable is used as flag to indicate whether an obstacle has been detected or not. If an obstacle is not detected the method that sets the wheel speed is called with the GPS

coordinates of the fire's initial point. On the other hand, if an obstacle is detected, the vector used to set the speed of the robot is the sum of the vector containing the gps coordinates and the vector returned from the obstacle detection method, both multiplied by the max speed value. That value ensures the robot will avoid the obstacle while moving towards the fire.

As already mentioned robots are not provided with a map of the operation area. On each simulation tick, foot-bots try to move closer to their goal. Entrapment on areas surrounded by trees, is one of the issues the swarm is facing. The vector obtained from obstacle avoidance algorithm, will assist robots escape when adjusting their course. Since the vector is not static but it is calculated from the readings, the trajectory of the robot is adjusted with a random value escaping such conditions.

Robots entering firefighting mode have their leds turned blue to visually indicate its state. When firefighting the amount of remaining substance in the tank is reduced and the heat value of the fire around the robot also decreases. Depending on the type of robot - as described previously - different total amount can be sprayed. The total amount a robot can carry is defined in the experiment configuration file.

After their mission the robots return to their hub. The navigation back to the base can be done using the same method already described. For the purpose of investigating further capabilities, the robot can have, in this thesis it's achieved using a guidance light. A light source is placed on top of the hub and the swarm will start following the light back to the nest. This technique is called phototaxis and can be seen in many species in nature - such as moths or even bacteria. Additionally, while at hub their load is slowly refilled with fire retardant.

3.3.5 Experiment configuration file

The core of an ARGoS simulation is the experiment configuration file, containing all the needed information to initialize and populate the simulated space. That file is parsed not only from the simulation engine but from user defined code also. The entities developed for this simulation parse the necessary user defined data.

The general configuration section seen below, defines basic settings of the simulation. The experiment is single threaded and runs until the user terminates it through the user interface. Also every second we execute ten simulation steps, resulting in a rather smooth simulation. Lastly, the random seed generator value, can be changed to generate

various environment setups. Keep in mind that the same value will generate the exact sequence of random numbers thus the exact same environment and conditions every time the experiment runs.

```
<framework>
  <system threads="0" />
  <experiment length="0"
              ticks_per_second="10"
              random_seed="124" />
</framework>
```

***Figure 26**: Framework section used on thesis*

The controllers section can be considered the most important part of the configuration file. It is where the entities are defined along with their properties. The first controller defined is the e-puck. Its attributes specify the id and the location of the library the controller will use. Also the actuators and sensors used from e-puck are declared. E-puck accepts 3 parameters defined by the user: spread_time, max_burn and spread_distance. Their purpose is to set the thresholds needed to simulate the fire, the first sets the heat needed to start spreading, the second the maximum heat before the tree burns completely and the third sets the maximum distance the fire can spread from each tree.

```
<epuck_controller id="fdc" library="build/controllers/burning_forest/libburning_forest.so">
  <actuators>
    <leds implementation="default" medium="leds" />
    <range_and_bearing implementation="default"/>
  </actuators>
  <sensors>
    <range_and_bearing implementation="medium" medium="rab" show_rays="false"/>
  </sensors>
  <params spread_time="50" max_burn="100" spread_distance="12"/>
</epuck_controller>
```

***Figure 27**: e-puck configuration file section. The relevant sensors and actuators as well as the path to controller file are specified.*

The foot-bot controllers declared after the e-puck section, handle the robots that form the swarm. As with the previous entity the actuators and sensors are added. Additionally, the parameters section contains all the user defined data our swarms needs to operate. The diffusion tag specifies the angle threshold that is used to detect and obstacle and decide whether to turn or not. Robot turning section contains the angle limits that

define the type of turn, soft or hard, the robot will take as well as the max speed. The last tag -named water tank - contains the maximum capacity a robot can carry and how much fire retardant it can disperse on every simulation tick.

```
<footbot_firefight_controller id="ffc" library="build/controllers/footbot_firefight/libfootbot_firefight.so">
    <actuators>
        <differential_steering implementation="default" />
        <leds implementation="default" medium="leds" />
    </actuators>
    <sensors>
        <footbot_light implementation = "rot_z_only" show_rays="false" />
        <footbot_proximity implementation="default" show_rays="false" />
        <positioning implementation = "default" />
        <footbot_motor_ground implementation="rot_z_only" />
    </sensors>
    <params>
        <diffusion straight_angle_range = "-5:5"
                   delta = "0.1" />
        <robot_turning Hard_Turn_Angle_Threshold = "70"
                       Turn_Angle_Threshold = "30"
                       Straight_Angle_Threshold = "10"
                       max_speed = "30" />
        <water_tank max_load = "100" />
    </params>
</footbot_firefight_controller>
```

***Figure 28**: Foot-bot declaration on configuration file. All sensors and actuators are declared and the initial values of structures are specified.*

In the arena section, all the information necessary for setting up the environment the swarm is operating are placed. The size of the arena is specified on the appropriate tag as well as the center of arena. Those values are used to get the GPS coordinates and the position of each entity - arena is Cartesian plane. Since the floor entity has to be accessible programmatically, a floor tag is declared in the arena section. Also the box entity is added to the arena to represent/simulate the walls around our environment. Each box is specifically declared as non-movable to prevent a possible movement in case of collision. Four light entities are added to the simulated space. All of them are placed on top of the hub and on equal distance to achieve phototaxis. It’s worth mentioning that when the floor entity is accessed by the loop functions, the lights are disabled to prevent the light from interfering with the floor coloring.

```
<arena size="5, 5, 2" center="0,0,1">
  <floor id="floor"
         source="loop_functions"
         pixels_per_meter="50" />
  <box id="wall_north" size="4,0.1,0.5" movable="false">
    <body position="0,2,0" orientation="0,0,0" />
  </box>
  <box id="wall_south" size="4,0.1,0.5" movable="false">
    <body position="0,-2,0" orientation="0,0,0" />
  </box>
  <box id="wall_east" size="0.1,4,0.5" movable="false">
    <body position="2,0,0" orientation="0,0,0" />
  </box>
  <box id="wall_west" size="0.1,4,0.5" movable="false">
    <body position="-2,0,0" orientation="0,0,0" />
  </box>
```

***Figure 29**: Arena configuration. The size and walls are defined as well as the center of the arena. Size can be modified to create larger or smaller areas for the experiments*

The distribute tag is responsible for distributing the entities across the simulated environment. As already described on chapter 2, there are different methods that can be used to distribute the objects. Both the e-puck and the foot-bot robots are using the uniform distribution to find their position in the environment and the Gaussian to decide their orientation. The min and max values ensure that the foot-bots will only be placed inside the hub and the e-puck will be placed outside the hub, leaving a small area free so the foot-bots can exit the hub.

```
  <distribute>
    <position method="uniform" min="-1,-2,0" max="2,2,0" />
    <orientation method="gaussian" mean="0,0,0" std_dev="360,0,0" />
    <entity quantity="100" max_trials="100">
      <e-puck id="fb">
        <controller config="fdc" />
      </e-puck>
    </entity>
  </distribute>
```

***Figure 30**: e-puck distribution section. ARGoS will try to distribute 100 robots in the environment choosing values from a uniform distribution with min and max values specified.*

The rest of the configuration file specifies the media required by some sensors and actuators, the graphics and physics engine and the loop functions that are required. The graphics engine used is the OpenGL engine, specified on the visualization section. The physics engines section contains the physics engine used. The dynamics 2D engine is used and it is already implemented in ARGoS. Those tags can be seen below:

```
<!-- ******************* -->
<!-- * Physics engines * -->
<!-- ******************* -->
<physics_engines>
  <dynamics2d id="dyn2d" />
</physics_engines>

<!-- ********* -->
<!-- * Media * -->
<!-- ********* -->
<media>
  <range_and_bearing id="rab" />
  <led id="leds" />
</media>

<!-- ****************** -->
<!-- * Visualization * -->
<!-- ****************** -->
<visualization>
  <qt-opengl />
</visualization>
```

***Figure 31**: Physics, Media and Visualization section of the configuration file. Media includes the led and communication media needed to ensure the data will be available across multiple engines and modules.*

3.4 Simulation Experiments

The robotic swarm was tested in several randomly generated simulated environments. Some of those environments were also generated to have some specific setup, such as closed/bottleneck areas or completely blocked initial point of fire. Also in different experiments the spread speed was modified to investigate different types of burning materials. An example of the simulated experiment can be seen on the following figure:

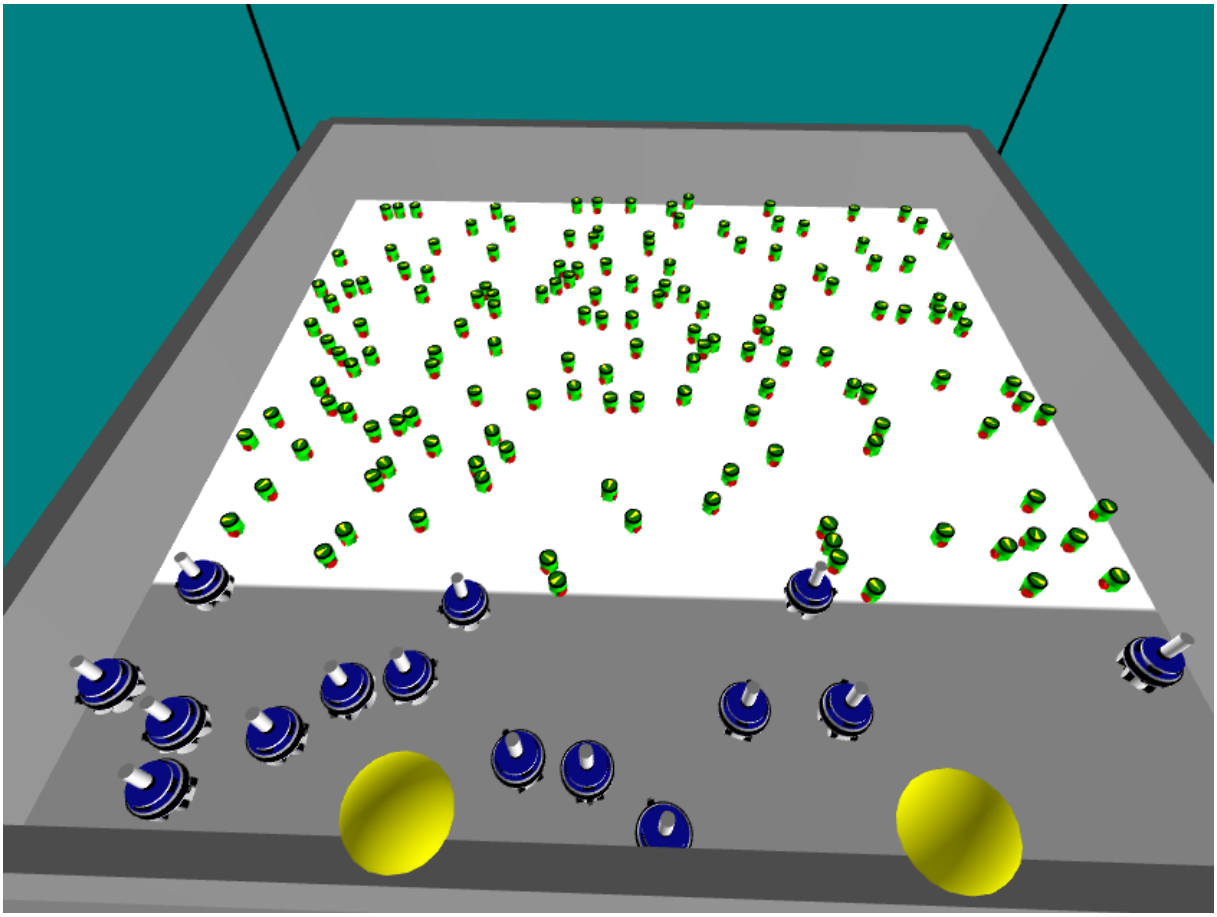

***Figure 32**: Simulated experiment ready to start. Yellow balls are the lights placed on top of the hub marked with gray color.*

3.4.1 Performance

Overall the performance of the swarm is acceptable for most of the experiments. In more than 50% of the scenarios simulated the fire was contained or taken out within reasonable amount of time preventing a wide fire. On unsuccessful experiments the swarm either failed to complete its task or a big amount of forest was burned by the time the robots arrived and start firefighting. Mission success depends on various factors that either make it easier, harder or even impossible to accomplish.

One of the main factors that affect the result is the topology of the forest. Since the swarm has no previous knowledge of the area nor mapping the forest while navigating, it is not always ensured that the closest path will be chosen or a path that does not lead to a dead end. Considering that hill climbing was employed entrapment on local optimal or

ridges and valleys can be an issue. As already mentioned robots may be able to escape but with a significant delay to reach their target.

Geographical location of the initial point is also a factor that has an effect on the performance of the swarm. Points closer to the hub are faster to reach and the fire is taken out or contained effectively. On the other hand, points that require traversal of a bigger are to reach them tend to spread in more trees around them, making harder to put them under control. That issue can be addressed by introducing multiple hubs inside the forest, placed in strategic positions.

It is also observed that the location of the hub exit can affect the robot's path. Hub's exit is not specifically defined in any of the experiments, robots can exit from any point that is not blocked by an obstacle. Although e-puck robots are placed in the simulated environment leaving the area around the hub free, it is possible for a robot to exit from a side leading to a dead end or an area that can trap it. A scenario like that can be seen on the following image.

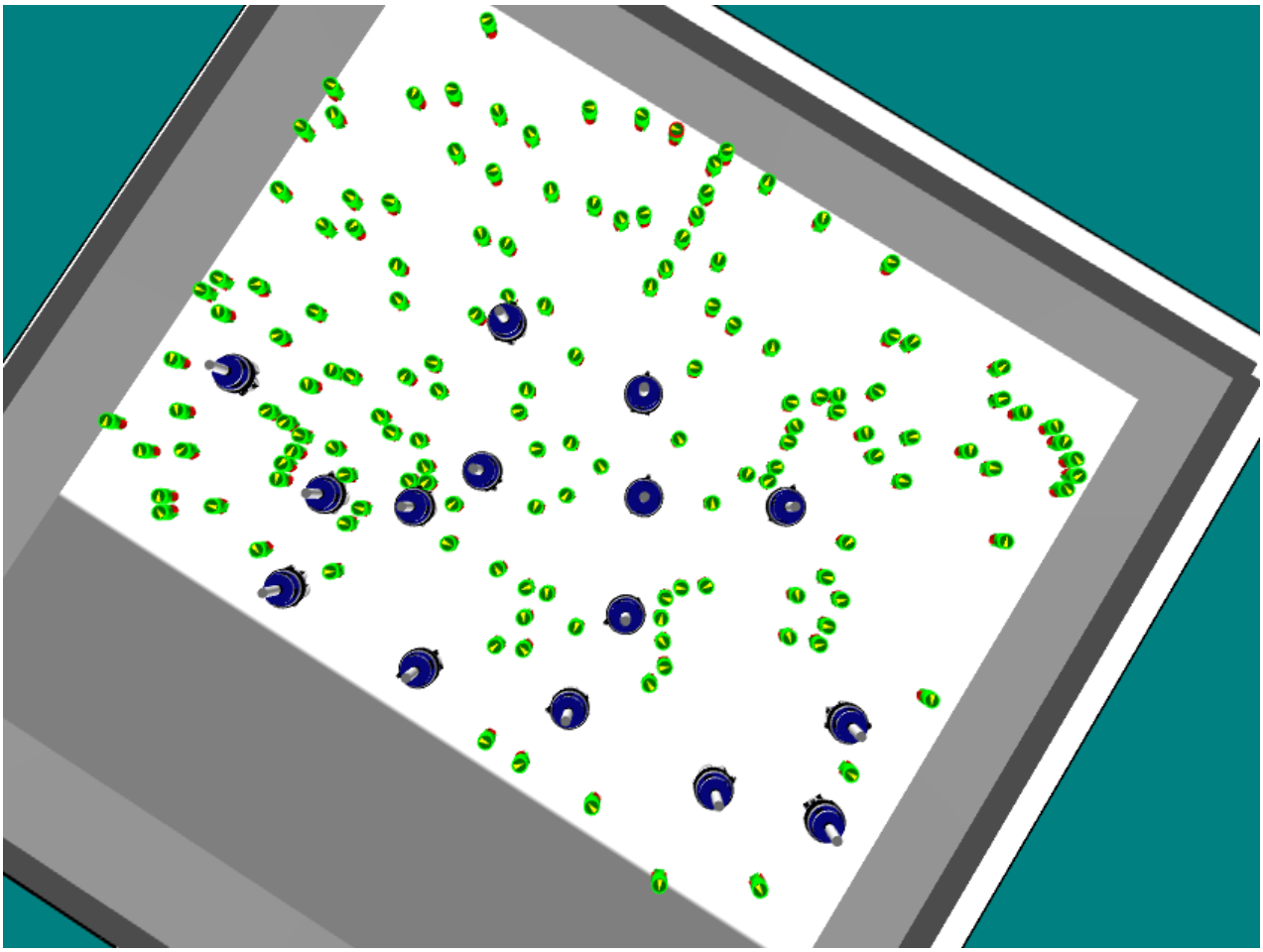

***Figure 33**: Closed area entrapment. The lower left side of the forest has many areas that robots can be trapped.*

Additionally, the number of robots forming the swarm is seriously affecting the overall performance. Congestion points can be formed in narrow passages or close to the goal location. Exits and entrances to the hub are also points that robots can block each other's way. This performance issue derives from the already mentioned disadvantage of robotic swarms. In order to address it, the most obvious approach is to use a trial and testing phase to determine the optimal size of the swarm, based on the total area of operation. A big and open space area can accommodate a larger amount of robots than a smaller area with a lot of trees.

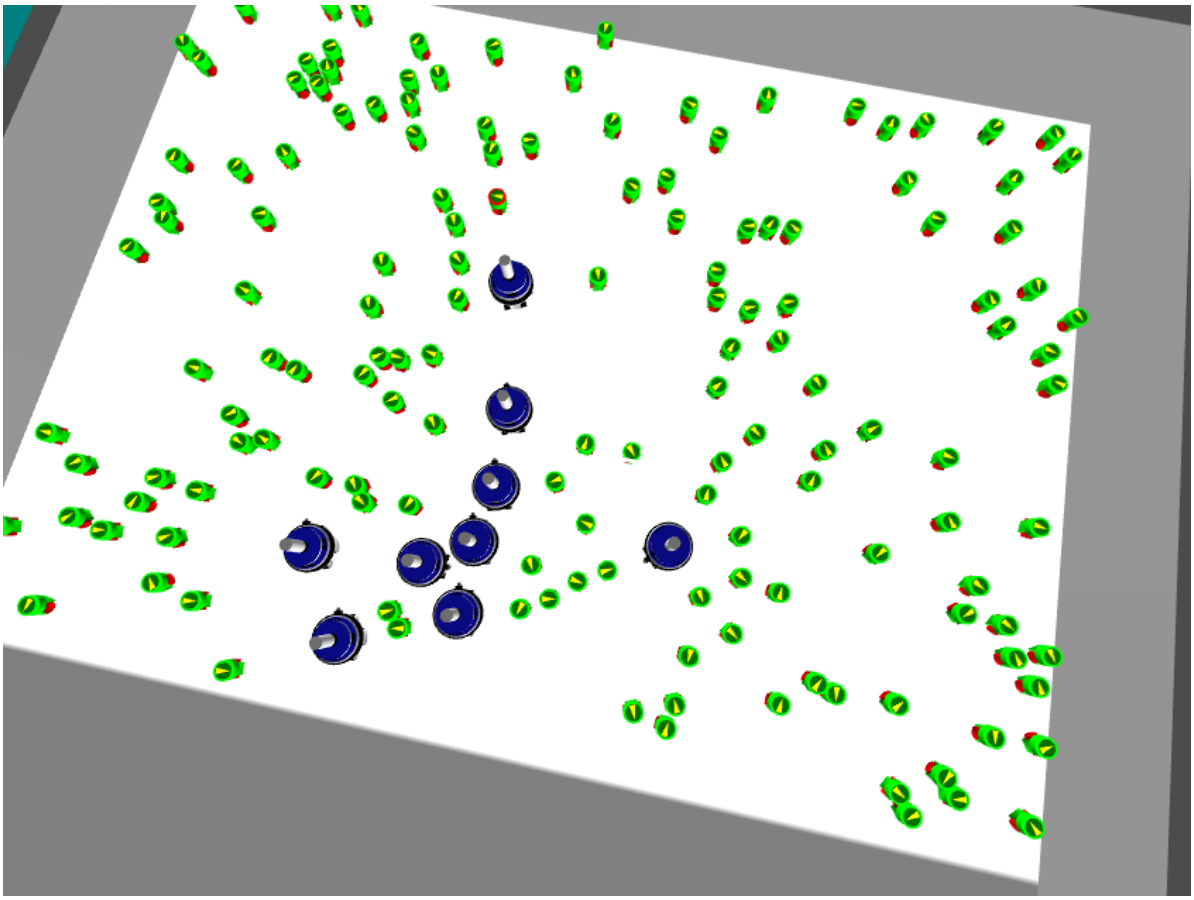

***Figure 34**: Congestion example. Robots gather around a narrow area leading to the fire*

As expected performance is not affected when individuals are lost or get trapped in the process. There is of course a specific limit as to how many robots can fail. When the number of robots drop below that number, the mission is either harder or impossible to accomplish. That threshold is hard to define since it depends on the specific scenario. A bigger fire with a significant amount of fuel material around, thus easier to spread, has a

smaller tolerance on the number of individual failures, whereas a smaller one with not many trees around the initial point has a higher tolerance.

In scenarios where the robots had to navigate through maze like areas with a lot of dead ends, resulted in large areas completely burned. While trying to escape dead ends, it was observed that robots usually collide with each other, especially when one was entering a bottleneck space and another one was trying to escape. As a result, the fire was spreading through the forest and the swarm only entered firefighting mode when the fire came close to them. There are some scenarios with a maze configuration that a few robots of the swarm managed to reach the fire on time. The reason for that is that robots exit the hub on different sides, resulting in different paths chosen.

Additionally, the swarm was tested in scenarios where there was only one path leading to the fire in order to observe the behavior on possible traffic congestion. Multiple robots entered the path and were blocked by the first robot that stop to spray fire retardant. When the fire reached the blocked robots the robots put the fire under control, something that could have been done earlier with less loses.

4. Conclusions

Wildfires is one of the major physical disasters (can occur naturally or due to human causes) that destroy large areas of forests every year. There are many techniques developed over the years, divided in three categories based on the time of intervention: prevention, detection and suppression techniques. Current thesis assumes there is a detection system in place and focus on the suppression of the fire.

The swarm consists of simple robots equipped with a tank able to carry fire retardant and a system to spray it. The mission of the swarm is to either completely take out the fire or contain it until ground forces arrive reducing the overall damage caused. The swarm is autonomous with no centralized control and it's depended on the detection system to provide the GPS coordinates of the fire.

The ARGoS platform was used for the simulation. ARGoS is a swarm robotics simulator with exceptional performance on multi robot systems. The simulator includes a physics and graphical engine as well as the robot models. Additionally, it's architecture is based on a modularity approach - everything is a plugin- allowing to add extra features easily without affecting simulation speed.

4.1 Synopsis and results

This thesis is focused on the simulation of the swarm. In order to effectively simulate a wildfire, the entities that represent trees had to be developed. ARGoS simulated entities that can be used to create the environment (boxes, cylinders etc), have limited usage when users need to programmatically access and interact with them. For that reason, the trees were simulated using the e-puck robots.

E-puck, a robot manufactured mainly for educational and teaching purposes, is mounted with all the necessary sensors and actuators needed to simulate a wildfire. Visualization, necessary for monitoring the experiment was achieved through color coding provided by leds mounted on the robot.

Fire spreading is based on a simplified heat model. Each tree (an e-puck robot) has an internal temperature value, a threshold that tells when to start spreading the fire to nearby trees. When the specified threshold is reached, a signal to nearby robots to allow fire spread simulation. The mass of the e-puck has been increased so that collision with foot-bots will not move the trees out of their original position.

The swarm consists of foot-bot robots. The model was already implemented in the ARGoS simulator. The swarm is activated when the first tree catches fire. The loop functions -responsible for controlling the simulated environment- inform the robots that a fire has started and provide them with the GPS coordinates of the initial point. After activation each robot starts to move towards the coordinates provided avoiding any obstacles found on the way, including robots of the swarm. Navigation is based on the hill climbing algorithm, the robot is setting its course in order to get to an unoccupied point closer to the target.

When a robot locates a burning tree, whether or not it's the initial point, it enters a firefighting mode starting to spray fire retardant. Firefighting simply modifies the heat value of trees in range, lowering it based on the robot's capabilities. Two types of foot-bots are implemented, a faster one with a small capacity tank and a slower one that can carry higher quantities of fire retardant.

After the fire fighting process is over the swarm is entering a state where it's navigating back to the hub. Returning to base can be done either by GPS navigation or phototaxis - a navigation based on lights inspired by nature. In this thesis the latter is used only to present the swarms capabilities.

There are cases where the robots are not able to completely take out the fire but instead they manage to contain it. If the fire is not spread for a set amount of time, and at least one robot entered firefighting mode, then the fire is considered under control.

The swarm is having an acceptable performance. Results from multiple experiments shown that it is a viable solution for wildfires. Although this thesis only focused on simulation in most scenarios the fire was taken out or put under control. Also most robots arrived at an area that was on fire within a reasonable amount of time allowing fast intervention.

More specifically, in scenarios where the initial point was close to the hub the swarm was located the fire was taken out completely, with only a very small on fire and only the initial point completely burned. Faster robots arrived at the spot fast enough to prevent further spreading while slower robots helped to take it out completely. It is worth mentioning that the smaller the fire the more effective the swarm was. A bigger fire required robots to unload all of the fire retardant they were carrying, thus forcing them to return to hub for reload.

Generally, the swarm's performance in scenarios with average to high distance between the hub and the fire, and random obstacles was more than acceptable. Most of the

fires were put under control and some of them were completely taken out especially when there was not a lot of fuel material around the initial point. There were cases though that the swarm didn't manage to perform as expected.

As a conclusion, a robotic swarm is a viable solution for wildfire firefighting. Proposed approach is a novel concept with further research required to achieve better results and make it viable in real life scenarios. Swarms can't be considered as a panacea that will solve wildfire problems. It is a measure that can help contain the damage until reinforcements arrive. There are many cases that require attention and problems that derive from the nature of the problem or the swarm system itself, and need to be addressed. In general, it is an area that can provide a wildfire suppression solution with low cost and high efficiency.

4.2 Limitations of the research

Current approach is a novel concept proposed and has several limitations either by design or due to the simulation nature. This research was not tested in any real life scenarios but only on simulated areas. Simulation is a good way to test a robotic swarm but can't fully resemble a wildfire or a physical robot.

Sensory input is clear without any noise from the environment. That resulted in easier navigation and communication. In the real world, all data coming from sensors is corrupted from noise (i.e. obstacle detection can detect insects or obstacles that the robot can ignore such as leaves or tall grass). That require further processing, something that it's not handled in this thesis.

Fire spreading model is simple heat model. There are many factors not taken into account that can affect a wildfire. Weather conditions can help the fire spread or slow it down as well as change the direction it's spreading. In all of the experiments the fire was spreading around each tree uniformly. Fuel material also can speed or slow the spreading rate. Areas with dry trees or easy to burn flora can reduce the time needed to reach the heat threshold. There are also some kind of trees that can spread the fire in high distances (i.e. pine cones can explode reaching high distances and help the fire spread). None of the above is taken into account in this simulation.

4.3 Future work

Further research can be done on the firefighting swarm to increase its potential and allow real life application. Future work can be focused either on improving the robots or improving the behavior and the intervention method of the swarm.

At the moment the swarm is homogenous, made completely with foot-bots. As already described navigating inside a forest can trap robots in dead ends. Additionally, the fire can be unreachable by ground (something that is not taken in consideration in this research). A heterogeneous swarm can be introduced consisting of both ground and aerial robots. Drones, being able to reach the fire faster, can spray fire retardant on top of it before it spreads. Also drones can help the ground robots navigate -as already demonstrated on the swarmanoid project.

Another important area that the swarm can be improved is path planning. Currently the hill climbing method is used and although it is effective, more efficient methods can decrease the time needed to reach the fire. There are several approaches proposed to path planning in forests of random trees that could be integrated. Super linear speedup is one of the most recent solutions presented by M. Otte (2014).

One of the drawbacks the proposed solution is having is the finite amount of fire retardant each individual can carry. In large fires, robots can empty their tank before the fire is out or more reinforcements arrive. Further research can examine a solution where the robots form a "chain" carrying a water hose attached to a fire hydrant. Work should be focus on effectively planning the path to the fire and the distances between each robot holding the hose. When the robot at the start reach the fire led signaling can inform a robot at the base to start the flow.

Lastly another major update to improve the swarm's performance will be local communication usage. Taking advantage of range and bearing sensors/actuators equipped on the foot-bot, robots can take advantage of individual knowledge of the area. Robots with memory of their previous state can inform the rest of the swarm of the path they took when they arrive to target, allowing them to adjust their course accordingly. Also individuals can inform others about dead ends or ask them to move when they block their way. Lastly they in cases where only one robot can approach the fire, it can inform the rest of the swarm and have them transfer their load to its tank, making a chain of robots able to stay on duty for more time.

5. Bibliography

Bachrach, J., Beal, J., & McLurkin, J. (2010). *Composable continuous-space programs for robotic swarms*. Neural Computing & Applications, vol. 19, no. 6, 825–847.

Beal, J. (2004). *Programming an amorphous computational medium*. In Lecture notes in computer science: Vol. 3566. Proceedings of the international workshop on unconventional programming paradigms (UPP), pp. 97. Berlin: Springer.

Beer, R. D., & Gallagher, J. C. (1992). *Evolving dynamic neural networks for adaptive behavior*. Adaptive Behavior, vol. 1, no. 1, 91–122.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. New York: Oxford University Press.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). *Swarm robotics: a review from the swarm engineering perspective*. Springer Swarm Intelligence Magazine, vol. 7, no. 1, pp. 1-41., Berlin: Springer.

Brooks, R. (1990). *Elephants don't play chess*. Robotics and Autonomous Systems, vol. 6, no. 1–2, pp. 3–15.

Brooks, R. A. (1986). *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation, vol. 2, no. 1, pp. 14–23.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., & Dorigo, M. (2012). *ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems*. Swarm Intelligence Magazine, vol. 6, no 4, pp. 271-295., Berlin: Springer.

Dorigo, M., Floreano, D., Gambardella, L.M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A.,

Christensen, A.L., Decugnière, A., Di Caro, G.A., Ducatelle, F., Ferrante, E., Förster, A.,Guzzi, J., Longchamp, V., Magnenat, S., Martinez Gonzalez, J., Mathews, N., Montes de Oca, M.A., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stuetzle, T., Trianni, V., Tuci, E., Turgut, A.E., & Vaussard, F. (2013). *Swarmanoid: a novel concept for the study of heterogeneous robotic swarms*. IEEE Robotics & Automation Magazine, vol. 20, no. 4, pp. 60-71.

Elman, J. L. (1990). *Finding structure in time*. Cognitive Science, vol. 14, no. 2, pp. 179–211.

Fine, T. L. (1999). *Feedforward neural network methodology*. Berlin: Springer.

Khatib, O. (1986). *Real-time obstacle avoidance for manipulators and mobile robots*. The International Journal of Robotics Research, vol. 5, no. 1, 90–98.

Matarić, M. J. (1997). *Reinforcement learning in the multi-robot domain*. Autonomous Robots, vol. 4, no. 1, pp. 73–83.

Matarić, M. J. (1998). *Using communication to reduce locality in distributed multi-agent learning*. Journal of Experimental and Theoretical Artificial Intelligence, vol. 10, no. 3, pp. 357–369.

Minsky, M. (1967). *Computation: finite and infinite machines*. Upper Saddle River: Prentice-Hall

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D. and Martinoli, A. (2009). *The e-puck, a Robot Designed for Education in Engineering*. Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, pp. 59-65.

Mondada, F., Floreano, D., Guignard, A., Deneubourg, J., Gambardella, L., Nolfi, S., & Dorigo, M. (2002, September). *Search for Rescue: An application for the SWARM-*

BOT Self-Assembling Robot Concept. Technical report LSA2 - I2S - STI, Swiss Federal Institute of Technology, Lausanne, Switzerland.

Mondada, F., Pettinaro, G., Guignard, A., Kwee, I., Floreano, D., Deneubourg, J.L., Nofli, S., Gambardella, & L.M., Dorigo, M. (2004). *Swarm-Bot: A New Distributed Robotic Concept*. Autonomous Robots, vol. 17, pp. 193-221, Netherlands: Kluwer.

Navarro, I., & Matia, F. (2013). *An Introduction to Swarm Robotics*. ISRN Robotics, vol. 2013.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics*. Intelligent robots and autonomous agents. Cambridge: MIT Press.

Otte, M., & Correll, N. (2014). *C-FOREST: Parallel Shortest Path Planning with Superlinear Speedup.* IEEE Transactions on Robotics, vol. 29, no. 3, pp. 798-806.

Rothermel, R. (1972). *A mathematical model for predicting fire spread in wildland fuels*. Research paper INT-115. USA: Intermountain forest and range experiment station, Forest service, Department of Agriculture.

Sahin, E., & Spears, W.M. (2005). Swarm Robotics: *From Sources of Inspiration to Domains of Application*. In Lecture notes in computer science: Swarm robotics, Vol. 3342, pp. 10–20, Berlin: Springer.

Soysal, O., & Şahin, E. (2005). *Probabilistic aggregation strategies in swarm robotic systems*. In Proceedings of the IEEE swarm intelligence symposium, pp. 325–332. Piscataway: IEEE Press.

Soysal, O., & Şahin, E. (2007). *A macroscopic model for self-organized aggregation in swarm robotic systems*. In Lecture notes in computer science: Swarm robotics. Vol. 4433, pp. 27–42. Berlin: Springer.

Soysal, O., Bahçeci, E., & Şahin, E. (2007). *Aggregation in swarm robotic systems: evolution and probabilistic control*. Turkish Journal of Electrical Engineering and Computer Sciences, vol. 15, pp. 199–225.

Stirling, T., & Floreano, D. (2010). *Energy Efficient Swarm Deployment for Search in Unknown Environments*. Proceedings of the Seventh International Conference on Swarm Intelligence, ANTS2010, Berlin: Springer.

Theraulaz, G., Goss, S., Gervet, J., & Deneubourg, J.-L. (1990). *Task differentiation in polistes wasp colonies: a model for self-organizing groups of robots*. In Proceedings of the first international conference on simulation of adaptive behavior on from animals to animats, pp. 346–355. Cambridge: MIT Press.

Wolpert, D. H., & Tumer, K. (1999). *An introduction to collective intelligence.* Technical Report NASA-ARC-IC-99-63. NASA Ames Research Center.

Wooldridge, M., & Jennings, N. (1995). *Intelligent agents: theory and practice*. The knowledge Engineering review, vol. 10, pp. 115-152. Cambridge: Cambridge university press.