ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

# RVR
# A new robot platform for swarm robotics
## The building blocks of new horizons

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en Informatique

**Raffaele Todesco**

Directeur
Professeur Mauro Birattari

Superviseur
Miquel Kegeleirs

Service
IRIDIA

DEMIURGE
Automatic Design of Robot Swarms
An ERC Consolidator Project

Année académique
2021 - 2022

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme: DEMIURGE Project, grant agreement No 681872

# ACKNOWLEDGMENT

First, I would like to thank my thesis director, Mauro Birattari, for trusting me and stimulating me to push this project forward as much as possible.

I also express a huge gratitude towards my supervisors, Miquel Kegeleirs and David Garzon Ramos, for their deepest dedication into the accomplishement of this thesis, even at the latest hours of the day. Their advice and opinion on my work have been of an invaluable help.

I also want to thank the researchers and students at IRIDIA, especially Jonas Kuckling, Guillermo Legarda Herranz, Muhammad Salman, Arthur Pierrot and Ilyes Gharbi, for the welcoming atmosphere at the lab and their help at crucial points of this thesis.

Finally, I thank my parents for convincing me to pursue these studies, when nothing was less sure, and for having been an inexhaustible source of motivation in these five years of study. I also address a special thanks to my friends, which have been key in keeping faith in my work when the end of the tunnel did not seem to come.

Thank you all for having been part of this work.

Raffaele Todesco

# Résumé


RVR : A new robot platform for swarm robotics

**Raffaele Todesco**

La robotique en essaim est un approche de systèmes multi-robots qui, depuis quelques années, a prouvé que des comportements complexes émergent de règles et interactions locales dans un groupe d'agents qui se coordonnent. En particulier, le robot e-puck a été largement utilisé en robotique en essaim, tout particulièrement concernant les récentes avancées en conception automatique de logiciel de contrôle.

Cependant, l'e-puck a aussi montré des défauts qui, au mieux, réduisent la facilité d'usage du robot, et, au pire, ont un impact négatif sur la performance de l'essaim. L'autonomie réduite ou la qualité plus basse de certains capteurs sont des exemples de telles faiblesses.

L'objectif de ce mémoire est d'étendre les capacités de la robotique en essaim, via une nouvelle platforme : le robot RVR. Le RVR présente d'intéressants capteurs, tels que la couleur du sol ou l'odométrie, qui pourraient constituer les premières pierres de nouvelles possibilités en robotique en essaim. En premier lieu, je présente comment cette nouvelle platforme a été intégrée dans un environnement de simulation, et comment le logiciel de contrôle conçu en simulation peut être transposé de manière transparente vers la réalité. Un nouveau processus de conception automatique, AutoMoDe-*Watermelon* – inspiré du logiciel de pointe AutoMoDe – est introduit et évalué sur un ensemble d'expériences qui soulignent l'efficacité d'une méthode de conception automatique avec cette platforme en particulier, à la fois en simulation et en réalité. Ceci montre que cette approche est prometteuse pour le développement de nouvelles tâches et capacités en robotique en essaim.



**Mots-clés** : robotique en essaim · RVR · conception automatique

*Mémoire présenté en vue de l'obtention du diplôme d'Ingénieur civil en Informatique à finalité spécialisée*
*Raffaele Todesco, 2021-2022*

# Abstract

RVR : A new robot platform for swarm robotics

**Raffaele Todesco**

Swarm robotics is an approach of multi-robot systems that, in the past years, has proven that complex behaviours emerge from local rules and interactions in a pool of coordinating agents. In particular, the e-puck robot has been widely used in swarm robotics, especially in automatic design of control software.

However, the e-puck has also shown some flaws that, at best, reduce the ease of use of the robot, and, at worst, have an negative impact on the performance of the swarm. The reduced autonomy or the lower quality of some sensors are examples of such weaknesses.

The goal of this thesis is to extend the capabilities of swarm robotics, by introducing a new robot platform : the RVR robot. The RVR presents some interesting sensors, such as ground color or odometry, that could constitute the building blocks of new swarm robotics behaviours. First, I discuss how this new platform has been integrated in a simulation environment, and how the control software designed in simulation can transparently be transposed into reality. A new automatic design process, AutoMoDe-*Watermelon* – inspired by the state-of-the-art AutoMoDe software – is introduced. It is evaluated on a set of test experiments that highlight the efficiency of an automatic design method with this particular platform, both in simulation and reality. This showed that is a promising approach for the development of new swarm robotics tasks and capabilities.

**Keywords** : swarm robotics · RVR · automatic design

# CONTENTS


**1 Introduction** **7**

1.1 Objectives . . . . . 7

1.2 Main contributions of the thesis . . . . . 8

1.3 Structure of the thesis . . . . . 8

**2 Related work** **9**

2.1 Swarm intelligence & swarm robotics . . . . . 9

2.2 Swarm robotics simulators . . . . . 10

2.3 Swarm robotics platforms . . . . . 10

2.4 RVR microcontrollers . . . . . 12

2.5 Interfacing & communication . . . . . 12

2.6 Experiments . . . . . 13

2.7 Automatic design . . . . . 14

2.7.1 Artificial evolution of neural networks . . . . . 14

2.7.2 AutoMoDe . . . . . 14

**3 The Sphero RVR** **16**

3.1 Hardware architecture . . . . . 16

3.2 Software architecture . . . . . 17

**4 Simulation** **20**

4.1 Robot model . . . . . 20

4.2 Simulating sensors . . . . . 20

4.2.1 Ground color sensor . . . . . 20

4.2.2 Velocity sensor . . . . . 21

4.2.3 Accelerometer . . . . . 22
4.2.4 Gyroscope . . . . . 23
4.2.5 IMU . . . . . 23
4.2.6 Light sensor . . . . . 24
4.2.7 Locator . . . . . 24
4.2.8 Quaternion . . . . . 24
4.2.9 Proximity sensors . . . . . 25
4.2.10 Lidar . . . . . 25
4.3 Simulating actuators . . . . . 26

**5 Control software** **27**
5.1 The reference model . . . . . 27
5.2 AutoMoDe-*Watermelon* . . . . . 28

**6 Experiments** **32**
6.1 Experimental environment . . . . . 32
6.2 Modules assessment . . . . . 32
6.3 Aggregation . . . . . 33
6.4 Grid exploration . . . . . 33
6.5 Color selection . . . . . 34
6.6 Protocol . . . . . 35

**7 Results** **36**
7.1 Assessment . . . . . 36
7.2 Aggregation . . . . . 37
7.3 Grid exploration . . . . . 38
7.4 Color selection . . . . . 40

**8 Conclusion** **44**

**A Finite State Machines** **45**
A.1 Aggregation . . . . . 45

A.2 Grid exploration . . . . . 49
A.3 Color selection . . . . . 53

# LIST OF FIGURES


3.1 The base RVR robot and the current prototype. . . . . 17

3.2 Base RVR architecture . . . . . 17

3.3 RVR - ARGoS architecture with ROS . . . . . 18

3.4 Complete robot architecture . . . . . 19

4.1 RVR 3D model. Black cylinders are visual representations of the RGB LEDs. The green part of the upper side highlights the front of the robot. . . . . 21

4.2 Noise of the color sensor for several ground colors. Each histogram color represents the corresponding color channel. . . . . 22

4.3 RVR axis system . . . . . 23

4.4 Proximity sensors noise distribution . . . . . 25

4.5 Lidar noise distribution . . . . . 26

5.1 An example of a PFSM control software that could be designed with AutoMoDe-*Watermelon* . . . . . 31

6.1 Assessment FSM for the black floor transition . . . . . 33

6.2 The simulated experimental setup for all three missions. . . . . 34

6.3 The real experimental setup for all three missions. . . . . 35

7.1 Control software performance on the aggregation mission . . . . . 38

7.2 Performance distribution for the aggregation experiment . . . . . 39

7.3 Top view of the extended RVR . . . . . 40

7.4 Control software performance on the grid exploration mission . . . . . 41

7.5 Control software performance on the color selection mission . . . . . 42

7.6 Performance distribution for the color selection experiment . . . . . 43

A.1 FSM 1 for the aggregation mission . . . . . 45
A.2 FSM 2 for the aggregation mission . . . . . 46
A.3 FSM 3 for the aggregation mission . . . . . 46
A.4 FSM 4 for the aggregation mission . . . . . 46
A.5 FSM 5 for the aggregation mission . . . . . 47
A.6 FSM 6 for the aggregation mission . . . . . 47
A.7 FSM 7 for the aggregation mission . . . . . 48
A.8 FSM 8 for the aggregation mission . . . . . 48
A.9 FSM 9 for the aggregation mission . . . . . 48
A.10 FSM 10 for the aggregation mission . . . . . 49
A.11 FSM 1 for the grid exploration mission . . . . . 49
A.12 FSM 2 for the grid exploration mission . . . . . 49
A.13 FSM 3 for the grid exploration mission . . . . . 50
A.14 FSM 4 for the grid exploration mission . . . . . 50
A.15 FSM 5 for the grid exploration mission . . . . . 51
A.16 FSM 6 for the grid exploration mission . . . . . 51
A.17 FSM 7 for the grid exploration mission . . . . . 52
A.18 FSM 8 for the grid exploration mission . . . . . 52
A.19 FSM 9 for the grid exploration mission . . . . . 53
A.20 FSM 10 for the grid exploration mission . . . . . 53
A.21 FSM 1 for the color selection mission . . . . . 53
A.22 FSM 2 for the color selection mission . . . . . 54
A.23 FSM 3 for the color selection mission . . . . . 54
A.24 FSM 4 for the color selection mission . . . . . 54
A.25 FSM 5 for the color selection mission . . . . . 55
A.26 FSM 6 for the color selection mission . . . . . 55
A.27 FSM 7 for the color selection mission . . . . . 55
A.28 FSM 8 for the color selection mission . . . . . 56
A.29 FSM 9 for the color selection mission . . . . . 56

A.30 FSM 10 for the color selection mission . . . . . 57

# 1 INTRODUCTION

Swarm robotics is an exciting field, straddling the border between artificial intelligence and robotics, gathering simple agents into a collective intelligence. The technical challenge of assembling such a functional party of robots, however, brings the development and use of dedicated simulators to the forefront, to explore possibilities beyond physical limitations.

The robot addressed in this thesis is the Sphero RVR, an education-tailed device packed with a large sensor set. Its modularity makes it an interesting candidate for robotics experiments, especially in swarm robotics. These native new capabilities, such as odometry – the ability for the robot to locate itself into space – and the extended autonomy might be a qualitative addition to the existing robot pool.

The purpose of this work is to build an ecosystem for RVR experimentation in swarm robotics, through the development of a simulator, and of a cross-platform control software interface that enables transparency between simulated and real-life behaviours of the robot. Moreover, this thesis aims to show that existing swarm robotics techniques, such as AutoMoDe, also work on this new platform.

This chapter is structured as follows : Section 1.1 defines the objectives of this thesis; Section 1.2 summarizes the main contributions of this thesis to the field; Section 1.3 lists the subsequent chapters of this work.

## 1.1 Objectives

This work aims to design and implement an architecture for swarm robotics development with the RVR, as well as to design the robot itself, augmented with new sensors.

This cross-platform architecture allows researchers to :

- Realize virtual experiments with multiple extended RVRs through the simulator, and design control software in simulation.
- Transpose transparently this control software to the real robot and achieve a similar behaviour.
- Exploit the features they need, or extend them, thanks to the modularity it offers.

More importantly, this work will also evaluate the quality of this transposition with control softwares automatically designed, by comparing the performance in simulation and in reality.

## 1.2 Main contributions of the thesis

This thesis aims to contribute to swarm robotics research.

First, I created a baseline of hardware and software architectures for the RVR, including 2 new sensors : the lidar and the proximity sensors, with the modularity and compatibility in mind.

Secondly, I developed a simulation environment that aims to mimic the real robots as well as possible; this environment is used to design control software, either by human experts or via an automatic process.

Finally, I assess the quality of automatic modular design with this platform, both in simulation and reality; the reality gap, i.e. the performance drop between simulation and reality, is estimated on 3 experiments and tracks of its sources are evoked.

## 1.3 Structure of the thesis

Chapter 2 summarizes the state-of-the-art of topics related to this thesis.

Chapter 3 describes the RVR robot, and its current software and hardware architecture implementations.

Chapter 4 presents the implementation of the simulation environment of the robot, as well as the virtual recreation of sensors.

Chapter 5 explains how the automatic design process takes place and the information it accesses.

Chapter 6 describes the experiments that will be conducted as a proof-of-concept for this robot platform.

Chapter 7 exposes the simulation and reality results of the said experiments.

Finally, Chapter 8 concludes this thesis with a brief summary of the work and insights of the possible future improvements.

# 2
# RELATED WORK

Robotics and artificial intelligence (AI) are evolving fast in past years, leading to an explosion of applications and models, and more and more complex robots emerged, especially with the help of machine learning.

This is where the swarm robotics particularity stands out : the point is to build AI from a collection of simple agents, and not from a single efficient brain. Although, this does not mean that they progress completely in parallel : the growth of deep learning for image recognition, for example, can totally be useful for a swarm element.

Developing for robots also means wisely picking hardware; especially with numbers of simple swarm members that can have to run costly algorithms. The evolution in microcomputers performance, and possibly energy consumption, is thus a key metric and bottleneck for this field.

This chapter is structured as follows : Section 2.1 examines swarm intelligence, swarm robotics and their principles; Section 2.2 compares the existing simulators designed for swarm robotics; Section 2.3 lists the other robot platforms used in swarm robotics; Section 2.4 discusses the different microcontrollers possibilities for the RVR; Section 2.5 details the communication software components and libraries; Section 2.6 highlights some common experiments of the field; Section 2.7 addresses the automatic design approaches followed in swarm robotics.

## 2.1 Swarm intelligence & swarm robotics

Swarm intelligence is defined as "*any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies*"[10]. It is often applied for solving combinatorial problems through the Ant Colony Optimization algorithm[10, 22], and can reach state-of-the-art performance in network optimization[4, 43] or other routing problems[86].

However, swarm robotics is a different topic. It is defined as "*the study of how large numbers of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment*"[72], which is yet inspired from natural ants as well.

Usually, swarm robots[11, 73, 6, 59] :

- are autonomous;
- can modify the environment in which they are situated;

- have local sensing and communication capabilities;
- do not have access to a centralized control or global knowledge;
- cooperate to achieve a given task.

This lack of control center is the major difference with other multi-robot approaches, that, on the other hand, can be strongly coordinated, or take advantage of an external source of information.

These properties also imply that robots are adaptive and independent from the group[11, 73, 6, 59]:

- they are able to cope with a missing peer, a faulty sensor, etc. : they are *fault-tolerant*;
- they stay efficient with swarms of different sizes : they are *scalable*;
- and they are *flexible* with the environment they are placed in.

## 2.2 Swarm robotics simulators

In order to efficiently prototype swarm behaviours, a powerful, full-fledged simulator, has to be developed for the RVR.

Gazebo[32] is a free, open-source simulator, designed for multi-robot and swarm robotics experiment. However, implementing custom sensors and actuators to mimic RVR's within this software seems complicated.

Webots[50] is a commercial open-source alternative, which is intended to realistically simulate multiple robots with accurate physics and collision. However, its performance decreases fast with a large number of robots[59], which is not desirable for a swarm robotics experiment.

Microsoft Robotics Studio[39] is a multi-robot simulator that can only run on Windows platforms, which can be a problem since the rest of the architecture of this project is Linux-based.

SwarmBot3D[64] is a tailor-made simulator for the s-bot; its lack of modularity hinders the addition of a new robot.

Finally, ARGoS[66] is an open-source, highly customizable simulator, dedicated to swarm robotics. Its main advantage is that it supports large scale experiments, by focusing less on high-precision physics, which is particularly relevant in our case. Moreover, it supports plugins for robots, sensors and actuators, which helps implementing new entities properly.

## 2.3 Swarm robotics platforms

Many different types of robots have been used in the past to conduct swarm robotics experiments[59, 73, 82].

1. Khepera[54] : developed for educational and research purposes, Khepera has been losing interest in past years, after having been widely used[60, 88, 16, 69], for example in exploration[53];

2. Khepera IV[76] : upgraded version of Khepera, by K-team. This circular robot comprises WiFi, Bluetooth, an accelerometer, gyroscope and camera, as well as proximity sensors. It is mainly education-oriented[24, 62, 23], dealing with usual swarm robotics problems such as path following and obstacle avoidance[24].

3. E-puck[33] : circular educational robot including proximity sensors, ambient light sensor, accelerometer, LEDs, Bluetooth, and optional Zigbee communication module. Numerous extension boards allow various usages[1] and communication possibilities[17, 35, 49];

4. Alice[13] : smaller rectangular robot, with high modularity (supports additional modules such as camera, RF or gripper);

5. Jasmine[42] : another small rectangular robot that implements proximal communication on top of the usual sensor set, that can face navigation problems[36, 80, 74] well.

6. s-bot[55] : versatile robot, with lots of actuators. Its self-reconfiguring and self-assembling abilities can help in tasks that involve objects interaction[20, 21].

7. Kobot[82] : interference-resistant, bigger robot that can distinguish other robots headings and obstacles. It is adapted for flocking purposes[83, 15].

8. Kilobot[18] : small low-cost robot, designed for large-scale swarms. Ideal for herding[61] and self-assembly tasks[71].

This abundance of swarm-compatible robots shows that the needs and experimental environment shape the choice of the platform. The common requirements and selection criteria that stand out of these robots are[73]:

- *Sensing and signalling* : the direct or indirect interaction between robots is the only way for them to communicate, and thus its quality is a primordial constraint :
  - The interference between the sensors and the external sources of noise should be minimized;
  - The robots should be able to recognize each other;
  - The robots should be able to communicate information to their peers directly or via their environment, for example via pheromones. This is called stigmergy.
- *Communication* : a wireless ad-hoc communication between the robots is highly recommended. Moreover, a connection to a development console is ideal for monitoring and debugging purposes;
- *Physical interaction* : robots can have the ability to physically interact, for example to self-assemble[72];
- *Autonomy* : the battery life of these should be long enough to allow the collective behaviour to emerge;

- *Cost* : tens of robots are required, which often means that a small individual robot cost is preferable.
- *Size* : multiple robots standing in the same test arena implies that their size should be taken into account. A reasonable size is a trade-off between compatibility with the arena size, and having an expandable robot that is not too expensive due to components miniaturization.

These criteria make the RVR a very good candidate for swarm robotics. At a reasonable price point, you obtain a well-sized robot with a large sensor set, that includes odometry, which greatly help to locate itself and other mates. It has a very good base autonomy compared to others, and, with additional sensors and a LIDAR, proximity detection is granted.

## 2.4 RVR microcontrollers

Robots cited earlier usually have their own microcontroller. In RVR's case, robot control is done through an UART interface, leaving the choice of the microcontroller to the user, among :

1. Raspberry Pi[26] : in this case, you have access to a relatively powerful microcomputer that can run Linux distributions. Sphero designed a Python API specifically for this platform[79].
2. Arduino UNO[2] : a lower-level microcontroller that uses Sphero's SDK as an interface to communicate with the robot.
3. micro:bit[51] : cheaper alternative that is more education-oriented, with a block-coding option.

In order to run computationally intensive programs and maximize compatibility, using a Raspberry Pi is the usual choice[58, 63, 3].

## 2.5 Interfacing & communication

Communication between robots, or between different components of the software architecture, is a key feature, especially in swarm robotics where inter-robots communication is a major source of information. A global framework that could bundle together high-level applications and lower-level control software is highly desirable to improve the modularity and the evolution capacity of the robot.

There are several frameworks that offer such abstraction.

ROS[70], the Robot Operating System, is a popular set of libraries for robot applications. The communication process of ROS works similarly to a RSS feed. Software components, called *nodes*, can *publish* information on given channels, named *topics*. These components can also retrieve information from other channels by *subscribing* to them. Every application is then using only the information it requires to work.

ROS is widely used in swarm robotics projects, for various applications, from Particle Swarm Optimization (PSO)[46, 48] to swarm organization[85, 38]. It has been more and more considered as a standard in robotics[89].

This popularity also created a large and active community, that designs packages which can be easily implemented due to the "plug-and-play" approach of the architecture, similarly to other frameworks such as the Mobile Robot Programming Toolkit (MRPT)[57]. MRPT is another open-source set of libraries that aims to help robotics research, especially in the mapping and navigation fields. Despite some successful applications[41, 56, 84], MRPT suffers the comparison with ROS due to its narrower community and applications. Moreover, MRPT libraries are wrapped into a ROS package, granting access to them if using ROS[5]. Other specific alternative exist : the Microsoft Robotics Developer Studio (MRDS)[52], a discontinued proprietary option thought for research; or CARMEN[14], a navigation toolkit.

Even if, in particular cases, these alternatives might be viable, the range of ROS applications and its broad, growing community turns it into the go-to option. Its logging, visualization and monitoring capabilities are great tools for development as well.

However, in swarm robotics, the use of ROS as inter-robot communication is extremely rare, because it then requires the use of a local network, infringing the rules presented in section 2.1 by using a central source of information. The borders of the use of the network and its limitations must be clearly defined to avoid altering the swarm property of the group.

Fortunately, ROS can be as restrictive as desired, by simply defining upstream the constraints the robots will face, which will translate into accessibility – or not – to some transmission topics. Robots can then fall back to computer vision or range-and-bearing devices to handle communication.

## 2.6 Experiments

In order to evaluate this new robot platform in comparison with the previous ones, a set of experiments has to be designed and the performance of the swarm should be evaluated on the said experiments.

The most common tasks represented in swarm robotics are [59]:

- *Aggregation* : The goal of the swarm is to gather on a defined or undefined place of the arena. Some variants include forbidden areas or ambient cues [27]. This task is often a subtask of more complex missions.
- *Dispersion/coverage* : Conversely, in dispersion missions, the robots have to cover the biggest area they can, while staying connected – or close enough – to their neighbours. Again, there exists variants with forbidden areas for example [27].
- *Pattern formation* : This problem consists of forming a global pattern with the robots, which is interesting due to the local information and communication constraints of swarm robotics.

- *Collective motion* : The aggregated swarm has to coordinate on a movement, similarly to a school of fish. Either the position and orientation of the robot is enforced within the group (formations) or not (flocking).
- *Foraging* : The robots have to transport objects from a source to their base, called *the nest*. This can lead to interesting experiments, where too heavy objects need to be carried by several robots [34], and can also introduce the division of labor concept, which splits the swarm into robots having to perform different tasks[40].

## 2.7 Automatic design

Creating robot swarms control software can rely on human expertise to design a proper, well-thought and understandable code. However, this is very time consuming, and, moreover, it has been shown that automatic design methods can perform better than human experts [27].

There exists 2 main approaches in the automatic design of swarm robotics [27] :

### 2.7.1 Artificial evolution of neural networks

A neural network maps an input (for example, the readings of the sensors) to an output (the actuators values, i.e., what the robot has to do). In order to obtain the best neural network controller, in terms of topology (number of nodes, hidden layers...) and parameters, an evolutionary algorithm will apply natural selection and eventually produce the best controller he could find. This has been the go-to option for several years[11].

In particular, the *EvoStick* algorithm is a constrained version of NEAT, where the neural network has no hidden layers and is feed-forward[29].

However, this approach has a hard time to cross the reality gap : even though it produces good results in simulation, with real robots the performance of the software drops heavily[29]. The source of this performance drop is the specialization of the control software to the simulation characteristics; the controller poorly generalizes to reality. This low-bias/high-variance problem has been tackled by introducing automatic modular design with AutoMoDe, which is described in subsection 2.7.2.

### 2.7.2 AutoMoDe

AutoMoDe generates control software automatically in the form of a probabilistic finite state machine (PFSM)[29]. Each of this state machine is then composed of 2 types of modules :

1. *Behaviours* : These are the states in the PFSM and represent a core behaviour the robot can accomplish : exploration, stop, attraction/repulsion to neighbours or light, etc.
2. *Transitions* : The transitions are the conditions to transition from one state to the other. These can be fixed probability, color of the floor, neighbour coun, etc.

AutoMoDe will then optimize the combination of modules to produce a control software represented as a PFSM. This process can be handled by various optimization algorithms.

This limited set of modules injects enough bias to reduce the impact of the reality gap significantly[29]. This method also allows a better understandability of the code compared to neuroevolution.

Several variants of the original version of AutoMoDe, AutoMoDe-*Vanilla*, exist, and are called flavours. Each flavour changes components of the automatic design process: modules, form of the control software, optimization algorithm, etc.

The first important variant is AutoMoDe-*Chocolate*, that improved the performance of *Vanilla* by using a different optimization algorithm, iterated F-race [29].

Afterwards, many variants appeared. Some examples are *Coconut*[77] that implements several exploration modules, *Tuttifrutti*[30] which allows interaction via colors or *Gianduja*[37] that adds explicit communication.

# 3
# THE SPHERO RVR

The RVR[78] is a differential treaded robot, developed by Sphero in Greenville, TX, for education purposes. It has a very attractive price point, which makes it an interesting choice for robotics experimentation. Unfortunately, it is fairly recent, which explains the very few current uses of the robot in research publications[58, 7, 75].

This chapter is structured as follows : Section 3.1 describes the inherent capabilities of the robot, the additional sensors, and the current prototype; Section 3.2 details the modular software architecture enabling cross-components communication.

## 3.1 Hardware architecture

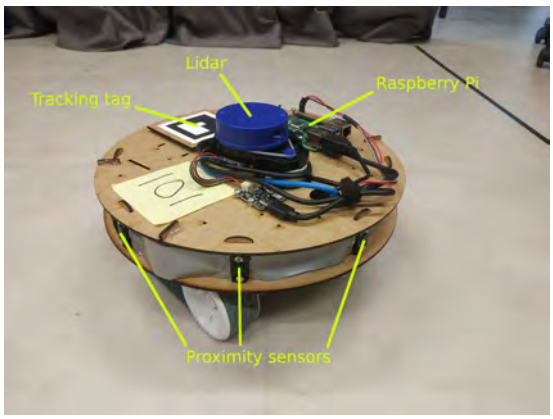
The factory version of the RVR, paired with a Raspberry Pi 4[26], provides the following main sensors :

- an RGB ground color sensor that detects the color of the floor;
- a velocity sensor;
- a 3-axis accelerometer;
- a 3-axis gyroscope, measuring angular velocities;
- an IMU sensor, that measures the robot pitch, roll and yaw;
- an ambient light sensor;
- a locator odometry sensor, that allows the robot to locate itself.

It also offers 5 RGB LEDs and its 2 distinct treads as actuators. An extended version of the robot, designed at IRIDIA, proposes 8 proximity sensors (Terabee Teraranger Multiflex[81]) and a lidar (YDLIDAR X4 [87]).
Communication between the robot and the Raspberry Pi is ensured through the UART protocol.

As shown on Figure 3.1b, the additional components (lidar, proximity sensors, and Raspberry Pi) are mounted on a cardboard "hat" on top of the RVR. Consequently, the ground and odometry sensors are still fully working, while the robot is able to detect obstacles with the proximity sensors. The lidar is the only element at its height level in the experimental environment, in order to enable neighbour detection between the robots. The only downside is that, currently, the robot is not able to detect ambient light. This will be fixed in future improvements as the opaque cardboard will be replaced by translucent plexiglass.

(a) Base RVR




(b) Extended RVR

Figure 3.1: The base RVR robot and the current prototype.

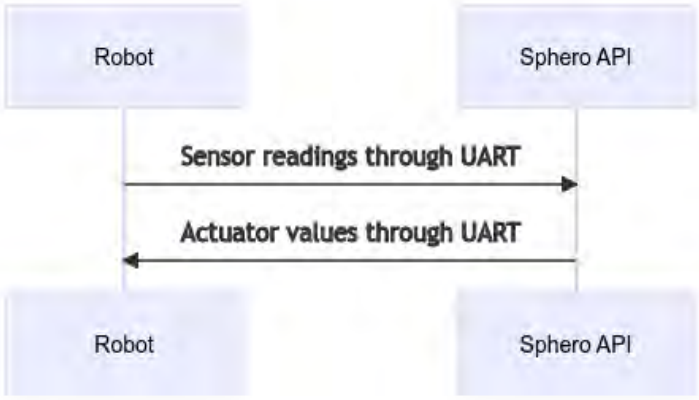



Figure 3.2: Base RVR architecture

## 3.2 Software architecture

Sphero developed a Python API[79] to enable interaction between the Rasperry Pi and the RVR through the UART protocol. Thus, any Linux distribution such as Ubuntu[12] or Raspberry Pi OS[25](previously Raspbian) supports this API. This creates the first level of abstraction presented in Figure 3.2.

This base architecture allows to design control software for the robot, but only within this Python environment which, as such, can not be connected to ARGoS and enable translation from simulation to reality.

This leads to the introduction of ROS in the ecosystem. As described in section 2.5, ROS connects multiple software components through *topics* to which these components can *subscribe* (declare that they want to get access to a given topic information) or *publish* (write information on a given topic). This will allow the ARGoS C++ control software and the Python RVR driver to transfer information to each other. In particular, the control software will subscribe to sensor topics and publish to actuator values, and the driver will do the opposite.

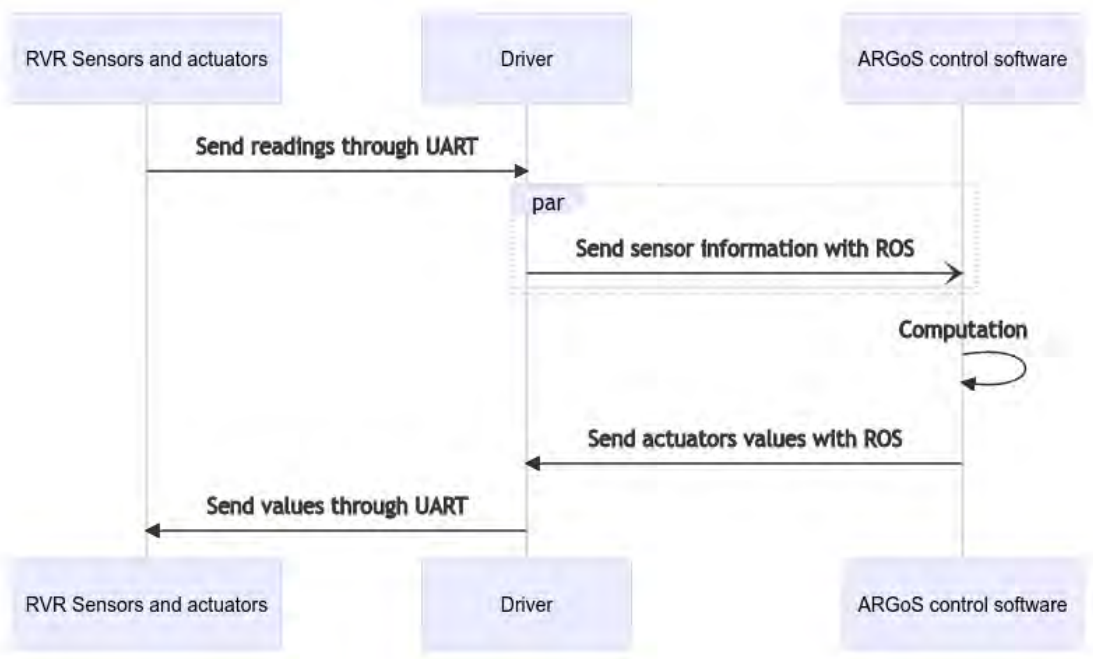



Figure 3.3: RVR - ARGoS architecture with ROS

The topics themselves will be split into the following dedicated ones :

- The color detected by the ground sensor;
- The ambient light perceived;
- IMU information, that groups :
  - The orientation (pitch, roll and yaw) provided by the IMU sensor;
  - Angular velocities from the gyroscope;
  - Linear acceleration from the accelerometer.
- Odometry information, which means:
  - The *pose*, including the position estimated by the locator and the orientation provided by the so-called quaternion sensor;
  - The *twist*, i.e., the angular velocities of the gyroscope and the linear velocities from the velocity sensor.

This architecture is presented in Figure 3.3.

Finally, the flexibility and modularity of this architecture is highlighted when adding the additional hardware modules : the lidar and the proximity sensors. Indeed, these sensors both have their own ROS nodes (software components), which means that they can connect to the same ROS master system as well. In order to integrate these sensors with the rest of the architecture, they will publish their information on dedicated ROS topics, which can be read by the ARGoS control software, showing that no additional work on the architecture is required to incorporate new sensors or actuators to the robot.

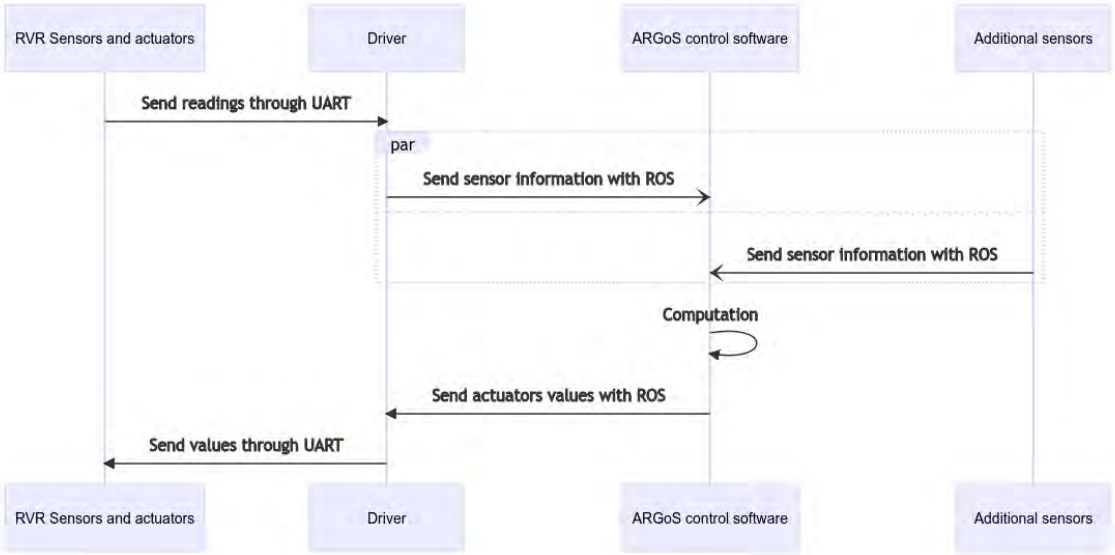



Figure 3.4: Complete robot architecture

This final architecture is schematized in Figure 3.4.

This modular architecture also helped to attenuate the ground sensor calibration problem. Depending on the robot, the color sensed by the robot beneath is highly biased, either towards darker or lighter tones. As no official calibration procedure is provided by Sphero, a color labeler ROS node has been integrated, in order to reliably detect tiles for the color selection experiment (see section 6.5). This additional node maps the raw reading to the RGB value of the closest label, defined by :

$$\text{label} = \text{argmin}_i \, d_i \tag{3.1}$$

$$d_i = (R_{sensor} - R_i)^2 + (G_{sensor} - G_i)^2 + (B_{sensor} - B_i)^2 \tag{3.2}$$

The result of this labeling is output on its own topic which is provided as an input to controllers.

# 4 SIMULATION

The simulation is a key point of robotics research. It allows important features such as offline design, repeatability, or easy testing. The simulator used here is ARGoS[66], due to its high modularity, in terms of robots and sensors, and to its compliance with swarm robotics needs. Its multi-engine capabilities allow the user to prioritize physical accuracy or experiment scalability, and its parallel computations improve performance on modern machines. It has already proven itself[68] with other robots such as the Kilobot[67] or the e-puck[66].

Several extensions of the simulator, named *plugins*, already exist and integrate new robot platforms into ARGoS, such as the Khepera IV[65] or the Kilobot[44].

This chapter is structured as follows : Section 4.1 presents the 3D model of the robot integrated in the simulation; Section 4.2 exposes the implementation of each sensor in simulation and the noise estimation for the main sensors; Section 4.3 explains the use and operation of the actuators.

## 4.1 Robot model

A 3D model of the RVR has been realized, according to robot measures and pictures. This 3D model can be seen in Figure 4.1. Regarding physics, a 2D physics model is implemented, using a cubed colliding box around the robot shape, similarly to the e-puck implementation[19]. The treads are undergoing a classical differential-drive model, with 2 distinct speeds $v_l$ and $v_r$.

## 4.2 Simulating sensors

Due to the large sensor set of the RVR, simulating its capabilities accurately was a substantial part of this work.

The implementation of each sensor in simulation will be described in the following section. As accurate simulation of the sensor noise is a key feature to mitigate the reality gap[47], I will then describe the noise evaluation and generation for the sensors that will be in use in the experiments of this thesis.

### 4.2.1 Ground color sensor

The ARGoS simulator already includes grayscale ground sensor in some robot implementations, because it is a common sensor in swarm robotics. However, the RVR is able to detect the color underneath it and provide it as an RGB (Red - Green - Blue) value. In ARGoS, this is simulated by checking the color of the virtual floor, at the position of the projection of the

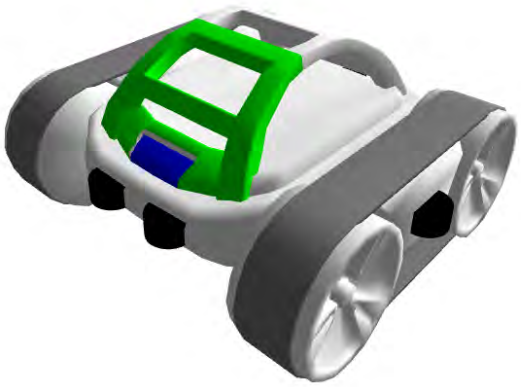


Figure 4.1: RVR 3D model. Black cylinders are visual representations of the RGB LEDs. The green part of the upper side highlights the front of the robot.

virtual color sensor on it. This virtual floor can be set as any image by the user.

The noise of the sensor has been evaluated for 4 different benchmark color measurements : red, green, blue and yellow, over 300 samples each. The measured noise histograms, as well as their kernel density estimation, are presented in Figure 4.2. Accordingly, a noise following a Gaussian distribution $N(0,5)$ for each color channel has been recommended in the experiment configuration. The $\mu$ and $\sigma$ parameters of the Gaussian distribution can be tweaked by the user to better simulate its own experimental environment.

### 4.2.2 Velocity sensor

The velocity sensor provides the robot speed along the $X$ and $Y$ axis "absolute" axis, in $m/s$. The $X$ and $Y$ axis here are defined as the axis according to Figure 4.3, at the boot time of the robot. However, ARGoS rigidbodies only provide the position and orientation of the robot in the virtual space, meaning that we have to compute the speed from wheel speeds $v_l$ and $v_r$. The idea here will be :

1. Compute the velocity $v$ of the robot by averaging $v_l$ and $v_r$.

$$v = \frac{v_l + v_r}{2} \tag{4.1}$$

2. Find the relative orientation $Q_{rel}$ from the original absolute orientation $Q_i$ and the current orientation $Q_c$, where all orientations are represented by quaternions.

$$Q_{rel} = Q_i^{-1} * Q_c \tag{4.2}$$

3. Compute the Euler angle $\alpha$, representing the rotation around the $Z$ axis, from $Q_{rel}$.

4. Return the speed vector as

$$\vec{v} = (v\cos\alpha, v\sin\alpha) \tag{4.3}$$

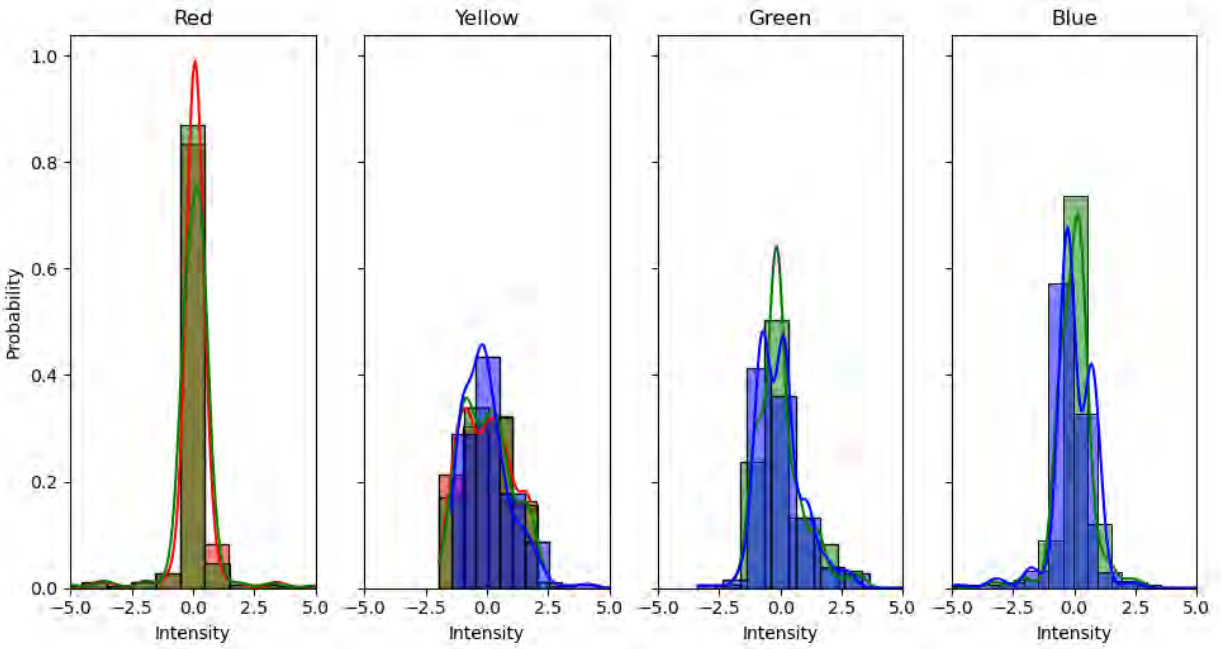



Figure 4.2: Noise of the color sensor for several ground colors. Each histogram color represents the corresponding color channel.

### 4.2.3 Accelerometer

The 3-axis accelerometer provides the robot acceleration around its 3 axis (see Figure 4.3), in $g$. The axis are defined as in the velocity sensor, which means that we will apply the same method to find the relative orientation $Q_{rel}$. To be able to simulate acceleration, we need to compute the evolution of speed over time. Again, the only speeds we have access to are $v_l$ and $v_r$. The process will be :

1. Compute the velocity $v$ of the robot by averaging $v_l$ and $v_r$, as earlier.

$$v = \frac{v_l + v_r}{2} \tag{4.4}$$

2. Compute the relative orientation $Q_{rel}$ and find the Euler angle $\alpha$ from it.

$$Q_{rel} = Q_i^{-1} * Q_c \tag{4.5}$$

3. Compute the current velocity vector $\vec{v}_t$ from $\alpha$.

$$\vec{v}_c = (v \cos\alpha, v \sin\alpha) \tag{4.6}$$

4. Retrieve the time spent between 2 speed measurements, which is the clock tick of ARGoS physics simulator $\Delta t$.

5. Compute the acceleration vector $\vec{a}$, with the previous speed vector $\vec{v}_{t-1}$, and convert it to $g$ ($G = 9.81\ m/s^2$).

$$\vec{a} = \frac{\vec{v}_t - \vec{v}_{t-1}}{G\Delta t} \tag{4.7}$$

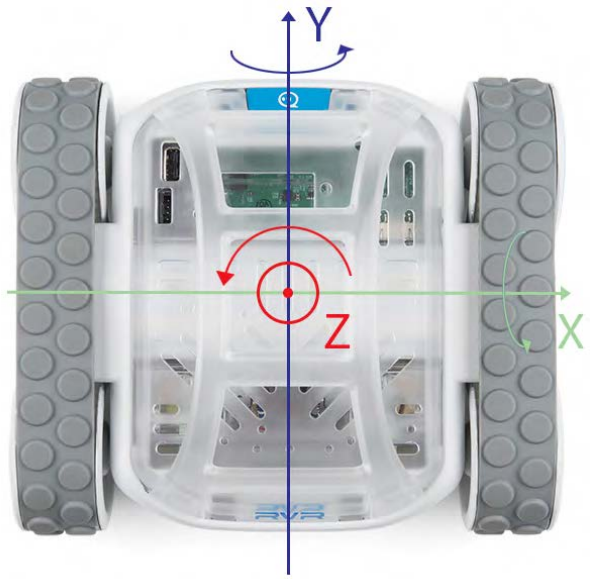



Figure 4.3: RVR axis system

6. Update the value $\vec{v}_{t-1}$ to $\vec{v}_t$ for the next accelerometer reading.

### 4.2.4 Gyroscope

The gyroscope provides the angular velocity of the robot around each axis. Similarly to the accelerometer, we only have access to the current orientation of the robot, and not its angular speed, which requires to compute the difference between 2 timesteps, which is done as :

1. Retrieve the differential orientation $Q_t$, the rotation between the previous timestep orientation $Q_{t-1}$ and the current one $Q_c$.

$$Q_t = Q_c * Q_{t-1}^{-1} \tag{4.8}$$

2. Convert $Q_t$ to Euler angles $\alpha_X$, $\alpha_Y$ and $\alpha_Z$, which represent the relative angular rotation and constitute the angular vector $\vec{\alpha}$.

3. Retrieve again the time spent between 2 measurements $\Delta t$.

4. Compute the angular velocity vector $\vec{v}$ :

$$\vec{v} = \frac{\vec{\alpha}}{\Delta t} \tag{4.9}$$

5. Update $Q_{t-1}$ to $Q_t$.

### 4.2.5 IMU

The Inertial Measure Unit (IMU) will abstract information from the magnetometer, accelerometer and gyroscope of the robot to provide orientation as pitch, roll, and yaw angles. As the IMU measurement is based on an external reference (with the magnetometer) that all

robots share, they will also share the same referential system for the IMU. For that reason, the simulated IMU will provide the orientation that comes from the absolute referential of the simulator, considering it thus has a virtual North pole that robots can align to.

### 4.2.6 Light sensor

The single ambient light sensor is placed at the front of the robot and can perceive the ambient light in lux. To simulate it, it is needed to sum the contribution of each light source in the scene and take occlusion into account.
The idea is the following :

1. Go through every light source in the scene
2. Check that light is not occluded by raycasting
3. Distribute the reading to the sensor according to its distance to it and its intensity
4. Clamp the value to the range the sensor can detect

The sum of all light sources contributions then constitute the sensor reading.

### 4.2.7 Locator

The locator can provide odometry information as $(X, Y)$ values inside an axis system that is defined as the robot is aligned to North pole, similarly to the IMU. Although the axis systems of each robot are aligned, the origin of the axis is the original position of each robot. The simulated locator thus works as the following :

1. At the beginning of the experiment, save the position of the robot as the origin of the locator axis $(X_o, Y_o)$
2. Provide $(X, Y)$ readings as the current coordinates of the robot in the absolute simulator axis system $(X_c, Y_c)$, translated to the locator origin.

$$(X, Y) = (X_c, Y_c) - (X_o, Y_o) \tag{4.10}$$

### 4.2.8 Quaternion

The quaternion sensor provides the relative orientation of the robot with respect to its booting position. The approach is similar to the locator's :

1. At the beginning of the experiment, save the orientation of the robot $Q_i$
2. Find the relative orientation $Q_{rel}$ from the original absolute orientation $Q_i$ and the current orientation $Q_c$

$$Q_{rel} = Q_i^{-1} * Q_c \tag{4.11}$$

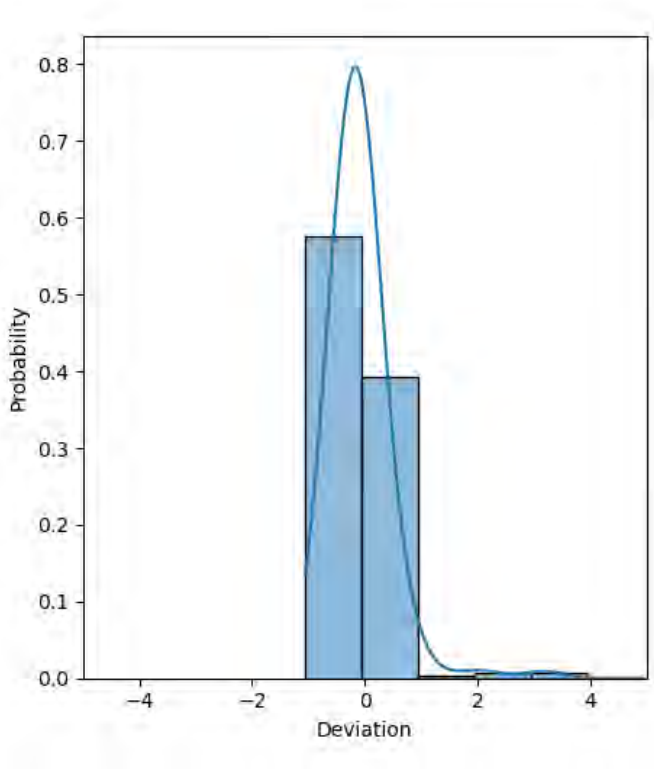



Figure 4.4: Proximity sensors noise distribution

### 4.2.9 Proximity sensors

The proximity sensors allow to detect obstacles between 0.05 and 2 meters. To detect obstacles in simulation, a raycasting approach has been applied.

1. For each sensor, check the first occluding object in the direction of the sensor
2. If the distance between the sensor and the obstacle is in the range $[0.05, 2]$, the sensor reading is that distance. Otherwise, it is $-\infty$ or $+\infty$ if the object is too close or too far, respectively.

The noise of the proximity sensors is normally-distributed with a low deviation, as shown in Figure 4.4.

### 4.2.10 Lidar

The lidar, due to its very similar functionality, has been implemented in the same way as the proximity sensors, with different sensor placements, many more readings (719 against 8), and a broader range : [0.10, 12] meters.

The lidar noise is distributed as in Figure 4.5. Accordingly, a uniform noise has been integrated, to which the user can tweak the range to better fit its observations and environment.

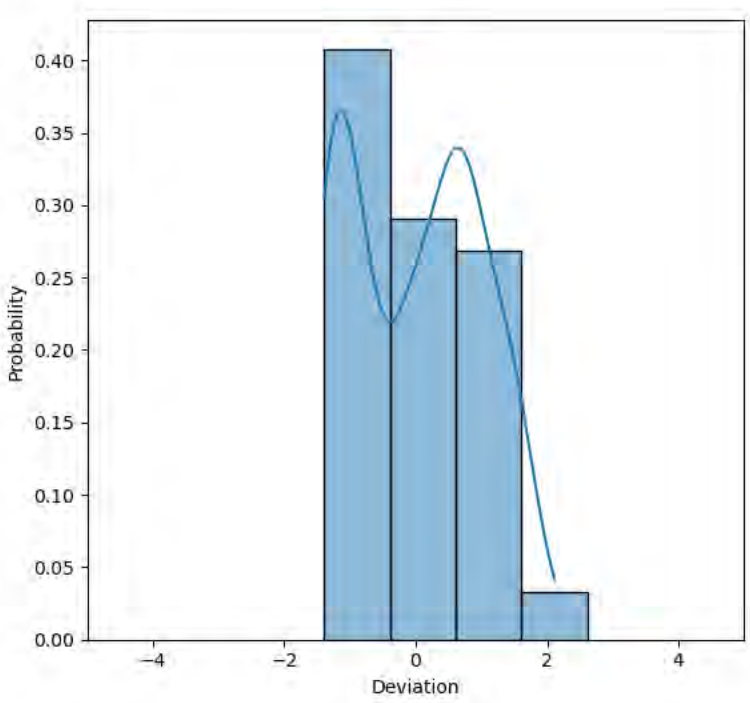



Figure 4.5: Lidar noise distribution

## 4.3 Simulating actuators

The current actuator set comprises :

1. 4 wheels grouped into 2 treads (connecting 2 wheels each);
2. 5 RGB LEDs (2 in the front of the robot, one on each side, one in the back).

In simulation, the wheels are considered as a differential drive to which 2 different speeds can be submitted (left and right), which matches the actual robot implementation. The LEDs are represented with black cylinders (see Figure 4.1) and can display any color from its RGB encoding.

# 5 CONTROL SOFTWARE

The previous chapters showed how the robots capabilities are transposed into a simulation environment, namely, ARGoS. In order to assess its quality and drive experiments with the robots, a control software that dictates the actions of the robot must be designed. As addressed in section 2.7, the human design approach will be complex and extremely time-consuming. AutoMoDe-*Chocolate* has been chosen as the base automatic design software for this thesis, as it crosses better the reality gap between the simulation and the actual swarm[29].

This chapter is structured as follows : Section 5.1 describes how an additional layer of abstraction is added to interface AutoMoDe and the robot; Section 5.2 presents the automatic modular design approach applied to the RVR.

## 5.1 The reference model

In AutoMoDe, every robot needs to be described as a *reference model*, a set of input/output variables that will be used by the different modules of the state machines produced by AutoMoDe. They will provide another layer of abstraction between the control software and the sensors and actuators. The reference model variables for the RVR are presented in Table 5.1.

The proximity sensor provides 8 readings included in $[0, 1]$, 1 meaning that an obstacle is less than 0.05 meters away from the sensor, and 0 signalling no obstacles in range. The maximum range of the proximity sensor can be tuned by the user to face different use cases. The proximity sensor also indicates the angle of the sensor with respect to the head of the robot.

The light sensor provides the illuminance it perceives, in lux. However, as the latest robot prototype does not have access to ambient light, this sensor is currently not in use.

The ground color sensor, as expected, provides the color output by the color labeler as an RGB value triplet.

The neighbour locator is an abstraction that needs to be made because of the difference between the robot implementation and the simulation. To detect its neighbours, the real robot uses the lidar readings and clusters them into groups it will consider as a single robot. In that way, each robot can detect its neighbours in a 0 to 10 meters range by detecting the lidar of other robots which is the only object at the same height. However, due to the physics simulation model, simulating inter-lidar detection is not possible. The simulation uses an omnidirectional camera that is able to detect virtual, invisible LEDs placed on the top of other robots. Practically, these two ways of detecting neighbours aim at the same objective, which are grouped under the neighbour localization sensor of the reference model.

| Sensor/actuators | Variables | Range of values |
|---|---|---|
| Proximity | $prox$ | $[0,1]^8$ |
| | $\angle q$ | $[0,2\pi)^8$ |
| Light | $light$ | $[0,16000]$ |
| Ground color | $R,G,B$ | $[0,255]^3$ |
| Neighbour localization | $d$ | $[0,10]^m, m \in \mathbb{N}$ |
| | $\angle q$ | $[0,2\pi)^m, m \in \mathbb{N}$ |
| Wheels | $v$ | $[0,155]^2$ |

Table 5.1: The reference model variables for the RVR

To extract the position of neighbours from lidar readings, they are clustered in the following way :

1. The algorithm goes through every of the 719 lidar readings.
2. If a reading lies outside of the [0.10, 0.75] range, it is discarded because it can not belong to another robot. The upper bound of 75 cm is thought to avoid considering walls of the room in which the arena lies as neighbours.
3. If 2 consecutive readings have a similar distance and angle (with arbitrary thresholds set at 2 cm and 10 degrees), they are clustered together and considered as belonging to the same robot.

This algorithm eventually outputs the number of neighbours, their distance and their heading with respect to the current robot.

The wheel variables are trivially the speed of each tread.

## 5.2 AutoMoDe-*Watermelon*

AutoMoDe-*Chocolate* control software takes the form of a probabilistic finite state machine (PFSM), which is composed of 2 types of modules :

1. *Behaviours* : These are the states in the PFSM and represent a core behaviour the robot can accomplish : exploration, stop, attraction/repulsion to neighbours or light, etc.
2. *Transitions* : The transitions are the conditions to transition from one state to the other. These can be fixed probability, color of the floor, neighbour count, etc.

In particular, AutoMoDe-*Chocolate* contains the same behaviour modules as the original AutoMoDe-*Vanilla*, which are the following [29]:

1. *Exploration* : the robot moves straight. If any of the proximity sensors positioned in front senses an obstacle, that is, if $prox_i \geq 0.1$ for any $i \in 1,2,7,8$, the robot turns on itself for a random number of control cycles chosen in 0,1,...,$\tau$,where $\tau$ is an integer parameter in 1,2,...,100. The robot turns away from the direction faced by the proximity sensor that returned the highest value.
2. *Stop* : the robot stays still.

3. *Phototaxis* : the robot moves towards the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where $k$ is a hard-coded parameter whose value has been *a priori* fixed to 5 and $\mathbf{w}'$ and $\mathbf{w}_o$ are vectors defined as:

$$\mathbf{w}' = \begin{cases} \mathbf{w}_l = \sum_{i=1}^{8} \left(\text{light}_i, \angle q_i\right), & \text{if light is perceived.} \\ (1, \angle 0), & \text{otherwise;} \end{cases}$$
$$\mathbf{w}_o = \sum_{i=1}^{8} (prox_i, \angle q_i), \tag{5.1}$$

4. *Anti-phototaxis* : the robot moves away from the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} -\mathbf{w}_{l,} & \text{if light is perceived} \\ (1, \angle 0), & \text{otherwise:} \end{cases} \tag{5.2}$$

   and $k$, $\mathbf{w}$', and $\mathbf{w}_o$ are defined in phototaxis.

5. *Attraction* : the robot goes in the direction of the robots in neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} \mathbf{w}_n = \sum_{m=1}^{n} \left(\frac{\alpha}{d_i}, \angle q_i\right), & \text{if robots are perceived} \\ (1, \angle 0), & \text{otherwise;} \end{cases} \tag{5.3}$$

   and $\alpha$ is a real-valued parameter in [1,5], and where $\mathbf{w}_o$ and $k$ are defined in phototaxis.

6. *Repulsion* : the robot moves away from the other robots in its neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} -\mathbf{w}_n, & \text{if robots are perceived} \\ (1, \angle 0), & \text{otherwise;} \end{cases} \tag{5.4}$$

   and $\mathbf{w}_n$ is defined in attraction, while $\mathbf{w}_o$ and $k$ are defined in phototaxis.

However, the (anti-)phototaxis behaviours rely on the presence, for the e-puck, of multiple light sensors which allow to find the direction of the light with respect to the robot. As the RVR owns a single light sensor, these behaviours can not be implemented without regularly sampling the light intensity around the robot to find its direction. Moreover, as the current prototype obstructs the light sensor, these modules will not be used with the RVR. Exploration, stop, attraction and repulsion are compatible and working with this robot.

Nonetheless, these modules also had to be tweaked, as the original modules were designed for the e-puck. The reasons of these changes will be addressed in section 7.1. This leads to the introduction of AutoMoDe-*Watermelon*, the first AutoMoDe flavour designed for the RVR. The behaviour modules of AutoMoDe-*Watermelon* are the following :

1. *Exploration* : the robot moves straight. If any of the proximity sensors positioned in front senses an obstacle, that is, if $prox_i \geq 0.1$ for any $i \in 1,2,7,8$, or if there is a neighbour in front, the robot turns on itself for a random number of control cycles chosen in 0,1,...,$\tau$,where $\tau$ is an integer parameter in 1,2,...,100. The robot turns away from the direction faced by the proximity sensor that returned the highest value.

2. *Stop* : the robot stays still.

3. *Attraction* : the robot goes in the direction of the robots in neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = k_w\mathbf{w}' - k_o\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} \mathbf{w}_n = \sum_{m=1}^{n} \left(\frac{\alpha}{d_i}, \angle q_i\right), & \text{if robots are perceived} \\ (1, \angle 0), & \text{otherwise;} \end{cases} \tag{5.5}$$

   and $\alpha$ is a real-valued parameter in [1,5], and where $\mathbf{w}_o$ is defined in phototaxis; $k_w$ and $k_o$ are hard-coded hyperparameters that have been *a priori* set respectively to 1.2 and 5.

4. *Repulsion* : the robot moves away from the other robots in its neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} -\mathbf{w}_n, & \text{if robots are perceived} \\ (1, \angle 0), & \text{otherwise;} \end{cases} \tag{5.6}$$

   and $\mathbf{w}_n$ is defined in attraction, while $\mathbf{w}_o$ and $k$ are defined in phototaxis.

Regarding transition conditions, AutoMoDe-*Watermelon* contains :

1. *Black floor* : if the ground is black (ground reading in grayscale is lower than 0.05), the transition is enabled with probability $\beta$, where $\beta$ is a parameter.

2. *Gray floor* : same as black-floor but the prerequisite is that the ground reading in grayscale is around 0.5.

3. *White floor* : same as black-floor but the prerequisite is that the ground reading in grayscale is above 0.78.

4. *Neighbour count* : the transition is enabled with probability

$$z(n) = \frac{1}{1 + e^{\eta(\xi - n)}} \tag{5.7}$$

   where $n$ is the number of robots in the neighbourhood, $\eta \in [0,20]$ is a real-valued parameter and $\xi \in 0,1,...,10$ is an integer parameter. The transition is enabled with probability 0.5 if $n = \xi$. The parameter $\eta$ regulates the steepness of the function $z(n)$ at $n = \xi$.

5. *Inverted neighbour count* : the transition is enabled with probability $1 - z(n)$, where $z(n)$ is defined in neighbour count.

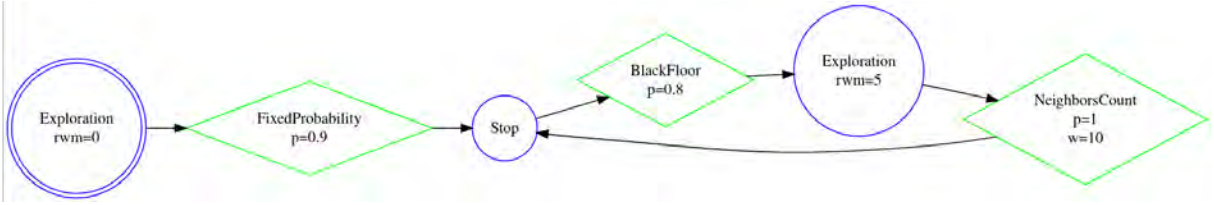



Figure 5.1: An example of a PFSM control software that could be designed with AutoMoDe-*Watermelon*

6. *Fixed probability* : the transition is enabled with probability $\beta$, where $\beta$ is a parameter.

All of these transitions are compatible and implemented with the RVR. Moreover, to take advantage of the ground color sensor, a $7^{th}$ transition has been created, inspired by AutoMoDe-*Tuttifrutti*[31]:

7. *Color detection* : if the robot perceives a given color $c$ among a fixed set (red, green, blue and yellow), the transition is enabled with probability $\beta$. $\beta$ and $c$ are parameters. To be resistant to noise, the hue of the color perceived by the sensor is compared to each of the hues of the expected labels. If it is close enough (under an arbitrary threshold of 15°), the closest label is considered as the perceived color.

An example of an AutoMoDe-*Watermelon* PFSM can be found in Figure 5.1.

In order to obtain a control software that highlights a desired collective behaviour of the swarm, the FSM choice problem is turned into an optimization problem that will be tackled by *iterated F-race*[9], as in AutoMoDe-*Chocolate*[28]. This algorithm will then maximize the performance of the swarm by evaluating different FSMs in the search space within a certain budget, i.e., the number of ARGoS simulations allowed. The performance of a given control software is defined by mission-specific indicators. Each FSM is restricted to a maximum of 4 states, and 4 outgoing transitions (which can not be self-transitions). Once the budget is depleted, iterated F-race outputs the best state machine it could find.

# 6 EXPERIMENTS

In order to assess the performance of the swarm, it will be evaluated on a set of test missions. The choice of missions to include aims to exhibit the general problem-solving capabilities of the robot compared to other robot platforms, as well as to highlight the new possibilities offered by the RVR in particular. Therefore, 2 missions are directly taken from other AutoMoDe variants, and the last one has been designed to emphasize the use of the ground color sensor, which is an exclusive capability compared to the e-puck, the only other platform currently supported by AutoMoDe. These 3 missions are aggregation, grid exploration, and color selection.

Before any actual experiment starts, each module of AutoMoDe-*Watermelon* should be evaluated individually. In this bottom-up approach, identifying potential strengths or weaknesses of the modules can help to mitigate or justify the reality gap. This is called the modules assessment experiment.

This chapter is structured as follows : Section 6.1 presents the environment in which the experiments take place; Section 6.2 defines the assessment protocol for single AutoMoDe components; Section 6.3, Section 6.4 and Section 6.5 describe respectively the aggregation, grid exploration and color selection experiment; Section 6.6 outlines the evaluation protocol for the said missions.

## 6.1 Experimental environment

Every mission is held in a dodecagonal arena of 4.91 $m^2$, which is delimited by walls high enough to ensure that the proximity sensors and lidar can not detect objects beyond them. The base color of the floor is gray, but some missions require some areas to be colored. All missions are performed by a swarm of 3 RVRs and last 120 seconds. 3 robots in the swarm ensure a reasonably good performance on the core behaviours, as the robots have to go at minimum 25 cm/s to avoid overheating the motors. Indeed, this speed in such a small arena with robots of 27cm diameter yields a lot of congestion and collisions with more than 3 robots.

## 6.2 Modules assessment

In this experiment, the goal is to examine the behaviour of the RVRs on the AutoMoDe-*Watermelon* modules. Each behaviour module is qualitatively evaluated 5 times with the swarm of 3 robots. If the robots manifest the expected behaviour, the behaviour module will be considered good enough to be included in the automatic design process.

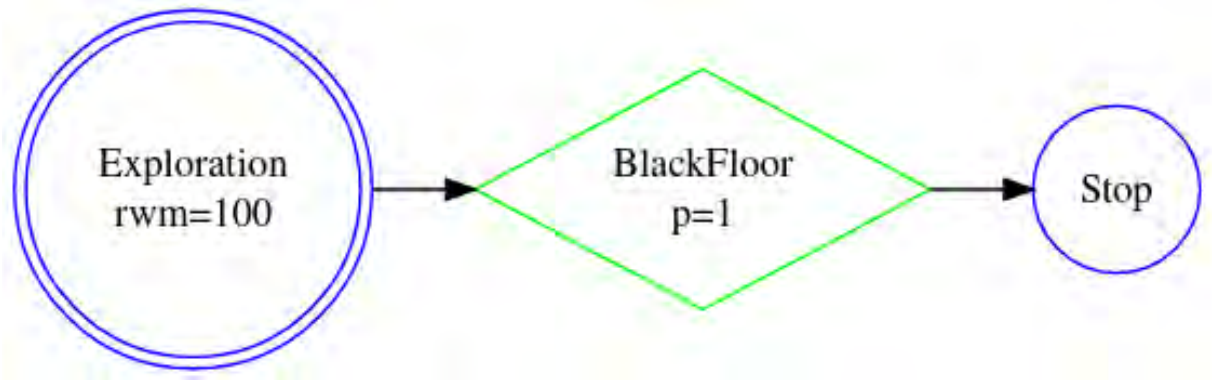



Figure 6.1: Assessment FSM for the black floor transition

Regarding transitions, each module will be tested with a 2-states FSM as presented in Figure 6.1. The robot will move from exploration to stop as soon as the transition is triggered (with $p = 1$ for *Black floor*, *Gray floor*, *White floor*, *Fixed probability* and *Color detection*, and with $\eta = 0$ and $\xi = 1$ for *Neighbour count* and *Inverted neighbour count*). If the robot successfully triggers the transition when it is expected to 10 times, it is considered good enough to be included in the design process.

## 6.3 Aggregation

This is inspired by the one-spot aggregation experiment from AutoMoDe-*Coconut*[77]. The robots have to aggregate as fast as possible on a black spot at the center of the arena. The floor is completely gray except for a black circular area of diameter 1m at the center of the workspace. At the beginning of the experiment, the 3 robots are randomly placed in the whole workspace. The workspace is shown in Figure 6.2a. The performance of the swarm is measured by the number of robots in the black area at the end of the mission. More formally,

$$F_{aggregation} = \sum_{i=1}^{N} b_i \tag{6.1}$$

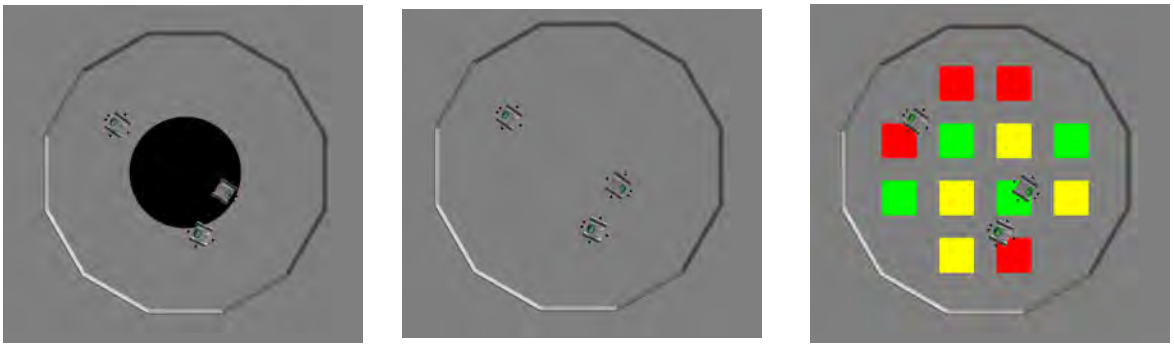
where $N$ is the number of robots and $b_i$ equals 1 if the robot $i$ lies in the black area at the end of the mission, 0 otherwise.

## 6.4 Grid exploration

This is the grid exploration experiment from AutoMoDe-*Coconut*[77]. The robots must explore and cover as much space as possible. The floor is completely gray. At the beginning of the experiment, the 3 robots are randomly placed in the workspace. The workspace is shown in Figure 6.2b. In order to measure the performance of the swarm, the arena is divided in a grid of 10 tiles by 10 tiles. For each tile, we retain the time $t$ elapsed since the last time it was visited by a robot. Each time the tile is visited by an robot, this time is reset to 0. The performance of the swarm is measured by the sum over all control cycles of the opposite of the average time $t$ over all the tiles. More formally,

| Color | Reward |
|---|---|
| Green | 0 |
| Red | 1 |
| Yellow | 2 |

Table 6.1: Color rewards for the color selection experiment



(a) Aggregation (b) Grid exploration (c) Color selection

Figure 6.2: The simulated experimental setup for all three missions.

$$F_{grid\ exploration} = -\frac{1}{N_{tiles}} \sum_{i=1}^{N_{cc}} \sum_{j=1}^{N_{tiles}} t_{ij} \tag{6.2}$$

where $N_{cc}$ is the number of control cycles for the whole experiment, $N_{tiles}$ is the number of tiles and $t_{ij}$ is the time, at the control cycle $i$, since the tile $j$ was crossed by a robot.

## 6.5 Color selection

The robots must stand as fast as possible on the color patches that reward the most. At the beginning of the experiment, the 3 robots are randomly placed in the workspace. The floor is completely gray except for 12 square color patches of different colors (green, yellow, and red) that are 30 cm of side length and shuffled randomly at the beginning of the experiment. A score is associated with each color, as shown in Table 6.1. The performance of the swarm is measured by the position of the robots at the end of the mission. The position is awarded by the reward associated with the said color. More formally,

$$F_{color\ aggregation} = \sum_{i=1}^{N} \sum_{j=1}^{N_{colors}} r_j b_{ij} \tag{6.3}$$

where $N$ is the number of robots, $N_{colors}$ is the number of colors, $r_j$ is the reward associated with color $j$ and $b_{ij}$ equals 1 if robot $i$ stands on a tile of color $j$ at the end of the mission, 0 otherwise.

Note that, given the robot size, a robot standing on a tile implicitly means that the center of the robot should lie above the tile. As the ground sensor is slightly offset towards the front, a behaviour that immediately stops as a good tile is detected might not always be accounted as a rewarding position, which should be handled by the control software.

(a) Aggregation (b) Grid exploration (c) Color selection

Figure 6.3: The real experimental setup for all three missions.

## 6.6 Protocol

AutoMoDe-*Watermelon* is executed 10 times on each of the 3 missions with a budget of 100k evaluations, generating 10 control software instances per mission. Each of the produced instances is then evaluated once on its respective mission [1]. The results of these evaluations are then presented mission by mission.

Then, each instance is uploaded on real RVRs and evaluated once in a real environment with the same geometry and features as in the simulation. The results of these evaluations are also presented for each mission.

[1] This protocol is the same as the one described in [29, 77, 28, 45] and discussed in [8].

# 7 RESULTS

As mentioned earlier, the goal of these missions is to evaluate how well a control software designed for the RVR, in particular with AutoMoDe-*Watermelon*, can transfer to reality. The state machines produced by AutoMoDe-*Watermelon* will be analyzed to see the effectiveness and utility of the modules, and the performance of the swarm in simulation and reality will be compared in order to evaluate the reality gap.

This chapter is structured as follows : Section 7.1 examines the functionality of each individual component of AutoMoDe-*Watermelon*; Section 7.2, Section 7.3 and Section 7.4 present the state machines and performance of the aggregation, grid exploration and color selection experiments.

## 7.1 Assessment

The 4 behaviour modules (*Exploration*, *Stop*, *Attraction* and *Repulsion*) pass the assessment test. Obstacle avoidance is effective most of the time with the proximity sensors. However, the robot presents a latency that could sometime lead to a too high reaction time when encountering another robot, and cause high velocity bumps. For this reason, the obstacle avoidance (in *Exploration* as well as in *Attraction* and *Repulsion*) also integrates the position of neighbours as detected by the lidar to improve its responsiveness. This permits a better anticipation and improved the overall obstacle avoidance of the robots.
The 7 transition modules also pass the assessment test.

This assessment experiment allowed to identify some discrepancies between simulation and reality, which can help to explain the potential reality gap.

- **Latency** : even though the simulation can act immediately on the wheels of the robot, it is not the case in reality. There is around 200 milliseconds of unconstant latency between the control software commanding a change in speed and the actual effect on the treads. This can have an impact on the overall behaviour of the swarm, especially regarding positioning and obstacle avoidance.
- **Obstacle detection** : the proximity sensors are very sensible to the glossiness of the surface against which their infrared rays collide. If the surface is too matt, it can induce errors in obstacle detection (either not detecting existing obstacles, detecting non-existing ones, or miscalculating the distance). Even with glossy surfaces, the proximity sensors are unpredictably unreliable. This is heavily affecting missions that rely on long exploration periods, such as grid exploration.
- **Neighbour detection** : some of the prototypes happened to sometimes show difficulties to detect their neighbours. This can bias the *Attraction* and *Repulsion* behaviours.

- **Timestep duration** : even with the same control software, the ROS software architecture slightly alters the control software step frequency, which can impact the spinning duration in the *Exploration* module. This has been fixed by enforcing a fixed time duration $T$ in seconds for the robot to turn on itself based on the random number of control cycles $r$[1]. It is computed as such :

$$T = r \times \Delta t \tag{7.1}$$

where $\Delta t$ is the duration of 1 clock cycle in simulation. In the context of this thesis, $\Delta t = 100$ ms. As long as the robot did not spin for $T$ seconds, it keeps spinning, which aligns better the simulation and reality behaviours.

## 7.2 Aggregation

The 10 states machines output by AutoMoDe-*Watermelon* for the aggregation experiment are presented in section A.1.

The overall presence of modules in the control softwares is summarized in Table 7.1. The occurrence rate is the proportion of control softwares including the module in question.

| Module | Average number of occurrences per control software | Occurrence rate |
|---|---|---|
| *Exploration* | 1.5 | 80% |
| *Stop* | 1.2 | 100% |
| *Attraction* | 0.2 | 20% |
| *Repulsion* | 0.4 | 40% |
| *Black floor* | 2.5 | 100% |
| *Gray floor* | 1.3 | 80% |
| *White floor* | 0.7 | 50% |
| *Neighbour count* | 0.6 | 40% |
| *Inverted neighbour count* | 0.4 | 30% |
| *Fixed probability* | 0.8 | 60% |
| *Color detection* | 1.2 | 50% |

Table 7.1: Module appearance statistics for the aggregation control softwares

As expected, the most present modules are *Stop*, *Exploration*, *Black floor* and *Gray floor*. Intuitively, they allow to place each robot in the black area and ensure it does not move away from it. *Attraction* and *Repulsion* are not very popular modules, because the robots manifest a tendency to aggregate at the perimeter of the black area and *Attraction* may lead to stuck a robot against its peers at the edge of the area. Conversely, *Repulsion* might reject the robot away from the rewarding area.

In some control softwares, *Color detection* is heavily represented. The hue comparison can apparently work as a gray/black comparison. This point will be investigated later in sec-

[1]This fix has been applied **after** running the 3 other experiments.

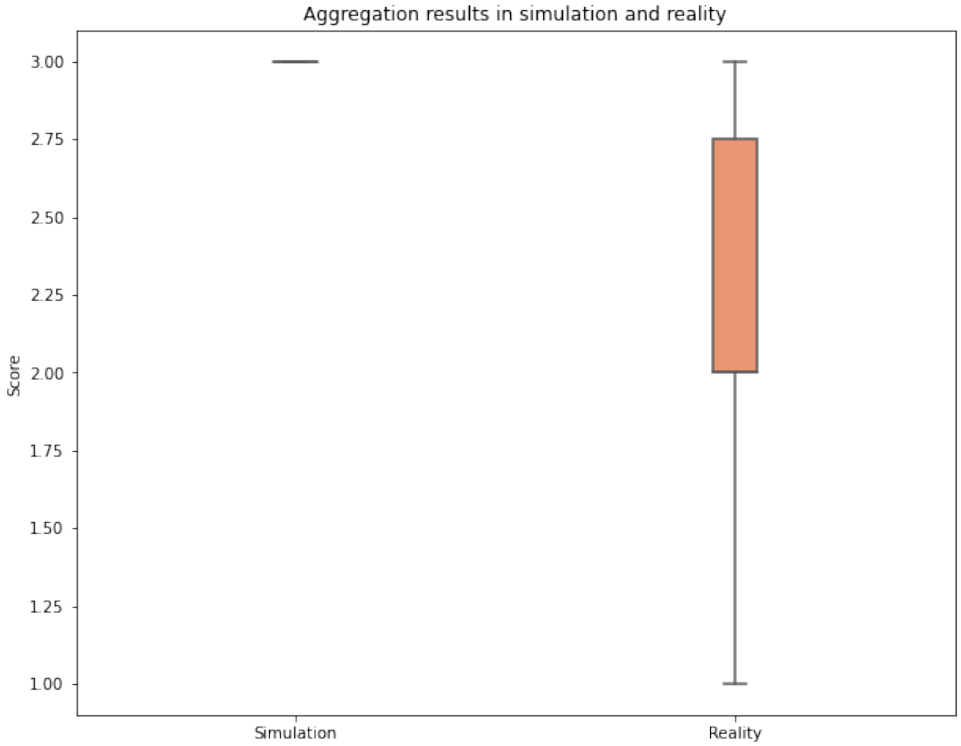



Figure 7.1: Control software performance on the aggregation mission

tion 7.4.

The performance of the control software in simulation and reality is compared in Figure 7.1. A more visual representation of the results is shown on Figure 7.2. There is effectively a performance drop. On average, 2.1 physical robots stand on the black spot at the end of the experiment. The divergence between the simulated and real timestep durations is probably a source of this difference. As the robots spin longer in simulation, when they encounter the walls of the arena, they can rotate longer and bounce back towards the center where the other robots and the objective lie. In reality, the robots tend to drive along the walls because they rotate for shorter periods of time, making it more difficult for them to reach the goal. However, we can notice that, on average, less than 1 robot misses the spot.

## 7.3 Grid exploration

The 10 states machines output by AutoMoDe-*Watermelon* for the grid exploration experiment are presented in section A.2.
The overall presence of modules in the control softwares is summarized in Table 7.2.

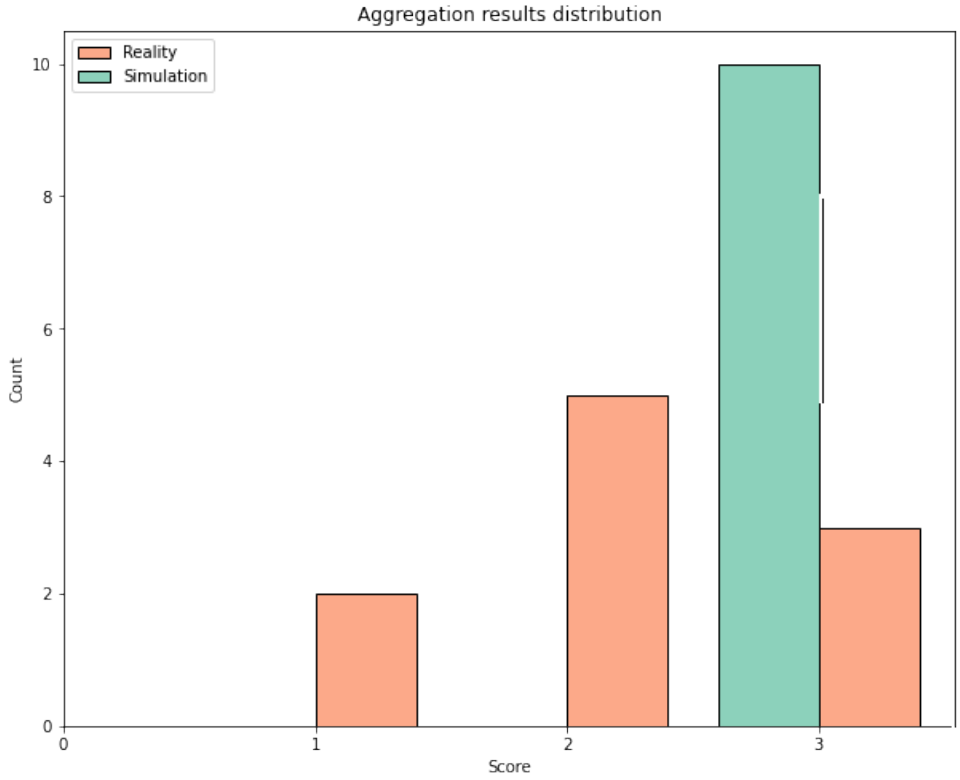



Figure 7.2: Performance distribution for the aggregation experiment

| Module | Average number of occurrences per control software | Occurrence rate |
|---|---|---|
| *Exploration* | 2.4 | 100% |
| *Stop* | 0.0 | 0% |
| *Attraction* | 0.0 | 0% |
| *Repulsion* | 0.1 | 10% |
| *Black floor* | 0.5 | 50% |
| *Gray floor* | 0.7 | 60% |
| *White floor* | 0.4 | 40% |
| *Neighbour count* | 1.0 | 70% |
| *Inverted neighbour count* | 2.1 | 90% |
| *Fixed probability* | 1.9 | 90% |
| *Color detection* | 0.7 | 40% |

Table 7.2: Module appearance statistics for the grid exploration control softwares

The behaviours here essentially consist of *Exploration* with different values for the $\tau$ parameter. The transitions are triggered either by *Fixed probability* – note that here *Gray floor* is equivalent to *Fixed probability* – or by the number of surrounding neighbours.

The performance of the swarm on the 10 evaluation runs is presented in Figure 7.4. Surprisingly, the swarm performs better in reality than in simulation. This might come from the fact that the score computation for reality is not exactly the same as in simulation. To compute the score in reality, robot positions are tracked with a tag that is considered at the

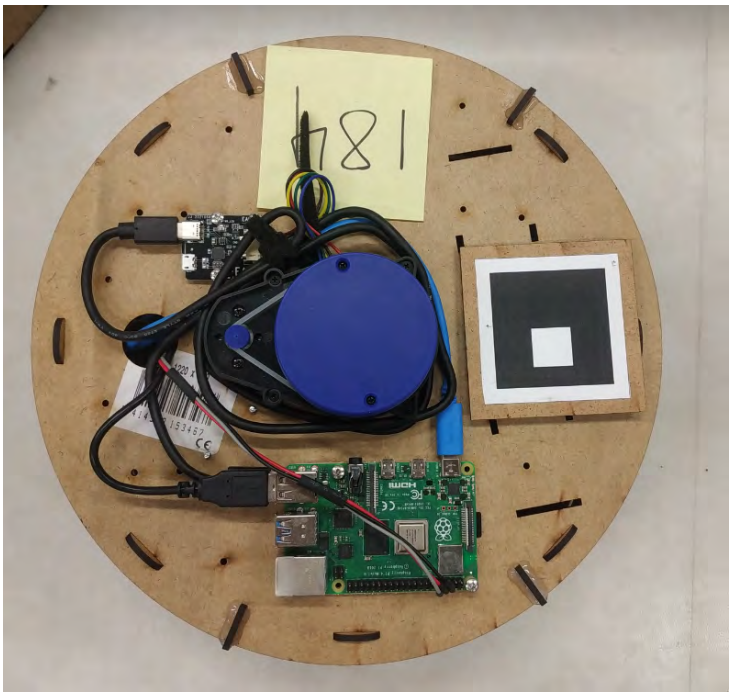


Figure 7.3: Top view of the extended RVR

center of the robot. However, this tag could not be placed at the center of the robot due to the presence of the lidar (see Figure 7.3). This creates an offset that is advantageous for the swarm in that experiment; indeed, when the robots turn on themselves, the tag is going over several tiles that the center of the robot actually does not cover, increasing the performance of the swarm. This might explain the better performance in reality. These results also show that the behaviour transferred effectively from simulation to reality.

The problem of faulty proximity sensors and timestep length has heavily been impacting this experiment. When the robots adopt an *Exploration* behaviour with a small $\tau$ value, in addition to the lack of reliable obstacle detection, they tend to simply go forward against obstacles, since, when they detect them, they spin for an extremely short period of time. This led to the crash of robots against the arena walls and them disassembling the arena. To be able to proceed with the experiment, heavy objects have been put against the walls outside of the arena, so that the robot are stuck against them without moving them.

## 7.4 Color selection

The 10 states machines output by AutoMoDe-*Watermelon* for the color selection experiment are presented in section A.3.

The overall presence of modules in the control softwares is summarized in Table 7.3.

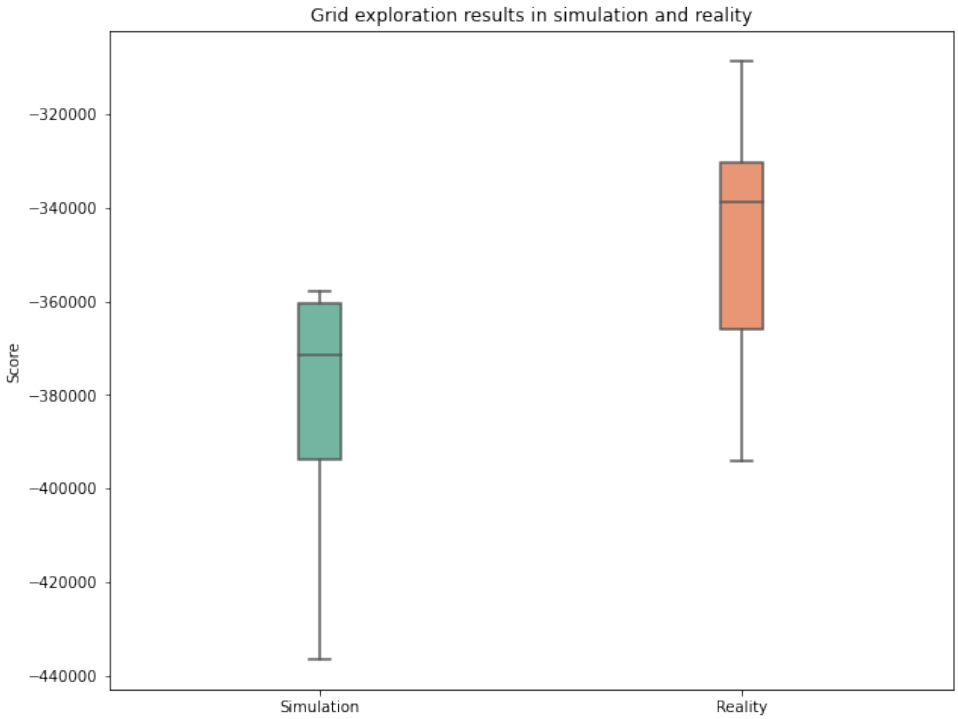



Figure 7.4: Control software performance on the grid exploration mission

| Module | Average number of occurrences per control software | Occurrence rate |
|---|---|---|
| *Exploration* | 1.6 | 100% |
| *Stop* | 1.1 | 100% |
| *Attraction* | 0.2 | 20% |
| *Repulsion* | 0.5 | 40% |
| *Black floor* | 0.9 | 70% |
| *Gray floor* | 1.9 | 100% |
| *White floor* | 1.6 | 100% |
| *Neighbour count* | 1.2 | 80% |
| *Inverted neighbour count* | 1.0 | 70% |
| *Fixed probability* | 0.7 | 70% |
| *Color detection* | 0.9 | 80% |

Table 7.3: Module appearance statistics for the color selection control softwares

We can summarize the use of each module in this mission :

- **Exploration** : in 8 of the 10 state machines, robots start in the *Exploration* state, and it appears in every control software. Indeed, it seems intuitive to be a good starting state to search for high reward spots. We can also notice that robots can switch from an *Exploration* state from another one with a different value for the $\tau$ parameter, to refine their behaviour.
- **Attraction** : the *Attraction* behaviour only appears once, in which it is the starting state. As the robots barely have room to fit in the colored tiles, it seems logical to avoid being attracted to other robots.

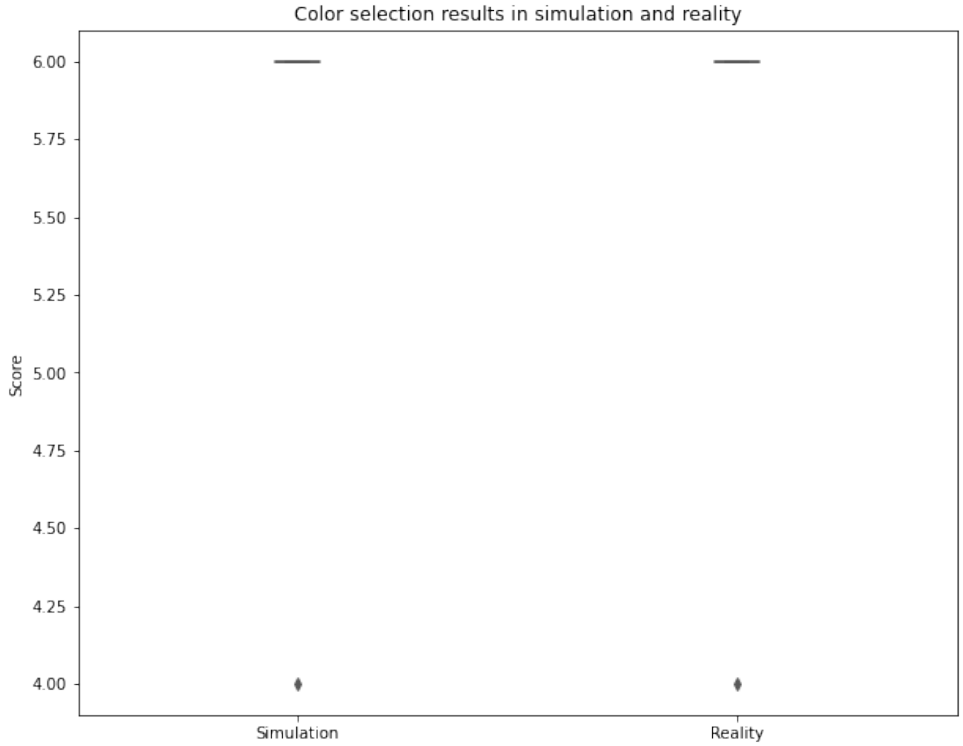



Figure 7.5: Control software performance on the color selection mission

- **Repulsion** : 3 of the control softwares integrate *Repulsion*. In Figure A.23, the initial state is *Repulsion* and another *Repulsion* state exist. Already more popular that *Attraction*, this behaviour allows to improve search within the arena when other robots have found – or not – a good place to stand.
- **Stop** : unsurprisingly present in every design, *Stop* allows the robots to stop when they detect a good spot.

Suprisingly, *Color detection* transitions are often replaced by *White*, *Gray* or *Black* floor. Due to the thresholding done in these transitions and the colors of the experiment, they can suffice to evaluate the quality of the current position of the robot. However, 80% of the control softwares also present the *Color detection*, meaning that they are still a useful transition; in this experiment they seem to be somewhat equivalent to grayscale comparisons. This is an indicator that specific new modules should be designed for the RVR. The AutoMoDe-*Watermelon* modules are based on modules designed for the e-puck robot, and this experiment highlights the need of platform-specific modules, to avoid this redundancy of features across multiple modules and improve the quality and abilities of the control software.

The performance of the control software in simulation and reality are compared in Figure 7.5. A more visual representation of the results is shown on Figure 7.6. The scores in simulation and reality are the same; in 9 out of 10 evaluations, the 3 robots ended up on yellow spots. In one of the runs, a single robot missed a rewarding tile. This shows that the control software is portable and works similarly across simulation and reality.

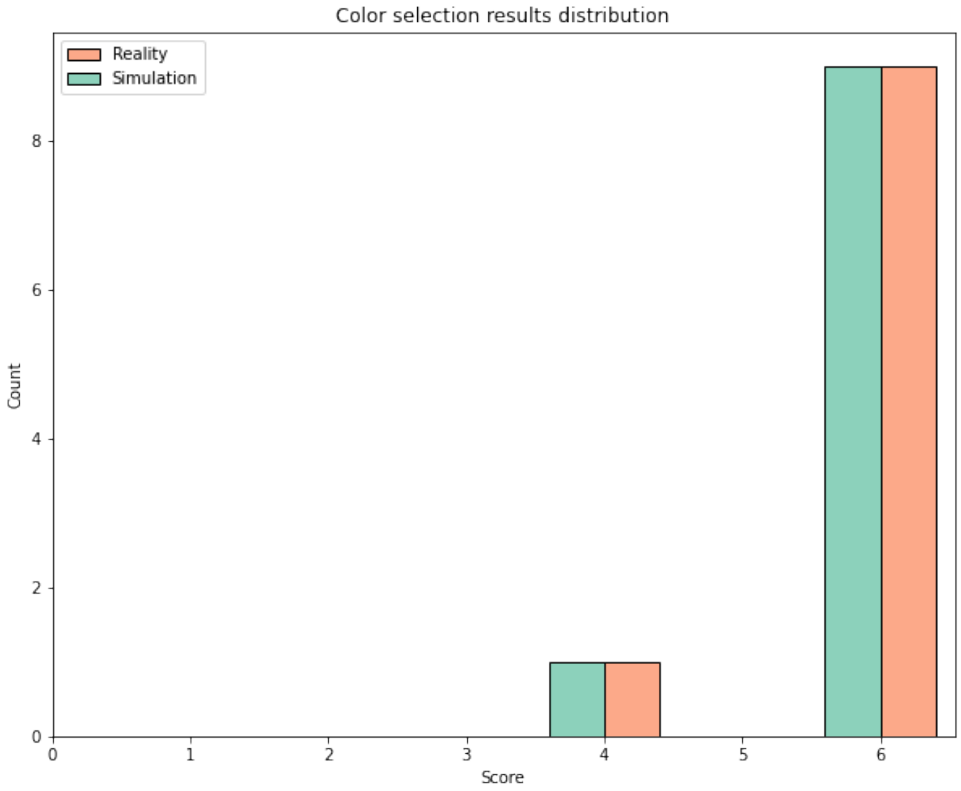



Figure 7.6: Performance distribution for the color selection experiment

# 8 CONCLUSION

In this master thesis, I have presented a software and hardware architecture for the RVR robot in swarm robotics. This enables the use of the robot within a simulation environment, as well as the transparent transfer of control software between simulation and reality. The addition of new sensors, i.e. the lidar and the proximity sensors, allowed to develop common swarm robotics features such as obstacle avoidance and peer detection.

In order to conceive the architecture, I reviewed literature about swarm robotics, robotic platforms for swarms, simulators, microcontrollers, and inter-component communication. The choice of the microcontroller and software libraries has been thought to be cross-compatible, portable and highly modular.

I then developed a unified platform that could be transposed easily from simulation to reality, thanks to the ARGoS software. Special care has been given to reality fidelity, with noise evaluation and imitation.

Thanks to this architecture, any control software designed in simulation can be directly transposed to reality. As a validation procedure, this new platform has been implemented as AutoMoDe-*Watermelon*, an automatic modular design of control software framework. After assessing the effectiveness of each individual component of AutoMoDe-*Watermelon*, the design process has been evaluated through 3 experiments.

The results showed that this robot is a promising platform for swarm robotics. The modular control software was able to achieve the objectives fixed in the missions, with some unexpected (but efficient) design choices, such as an equivalence between grayscale and color comparison. This highlighted the need to (re)design hardware and software components, thought for this platform, that would exploit its capabilities better, even though the current ones are already able to make a coordinated behaviour emerge.

Real-life experiments also showed that the control software automatically designed offline in simulation can be effective in reality. The reality gap manifested itself in the results, leading to a divergence between the simulation and reality results. The discrepancies between simulation and reality are addressed in this work and reducing them could be the objective of a future work.

As a conclusion, this work constitutes the foundations of a new window of opportunities for swarm robotics. The new capabilities of this robot, as well as its modular architecture, could lead to improvements in swarm robotics real-world applications such as disaster area mapping, with the included odometry. I strongly believe that this platform can achieve interesting new tasks and unlock a new potential for swarm robotics.

# A
# FINITE STATE MACHINES

This appendix contains all the control softwares produced by AutoMoDe-*Watermelon* for this work.

## A.1 Aggregation

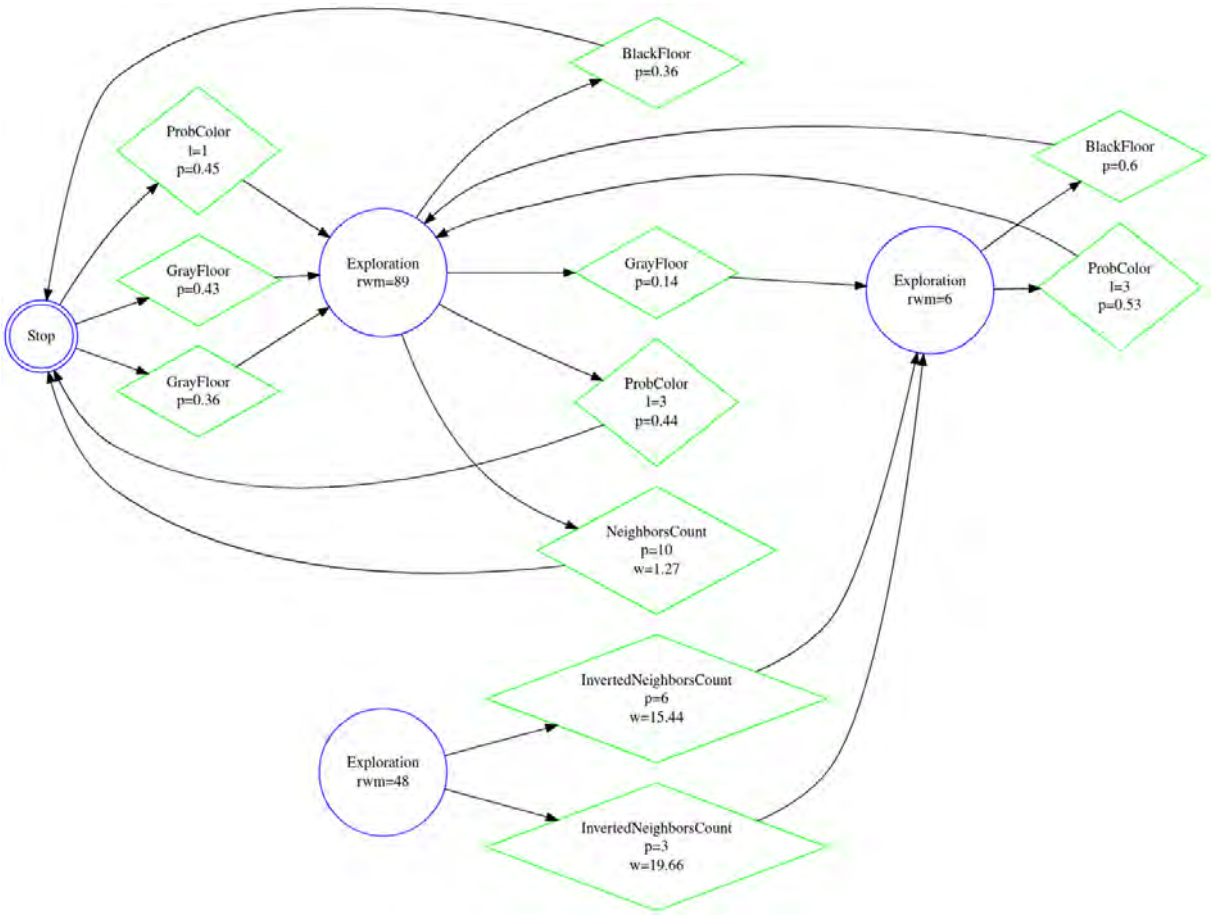



Figure A.1: FSM 1 for the aggregation mission

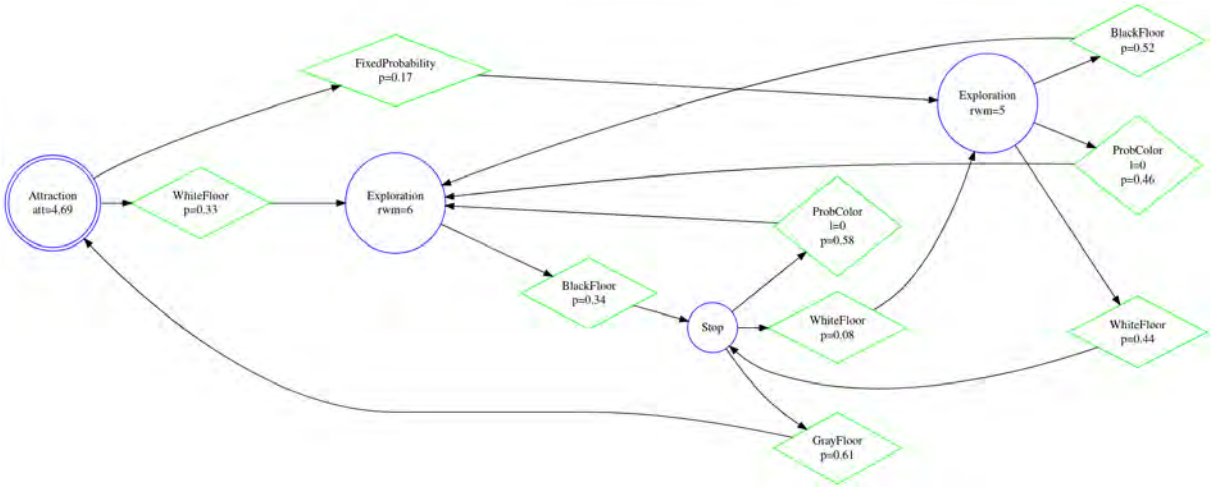



Figure A.2: FSM 2 for the aggregation mission

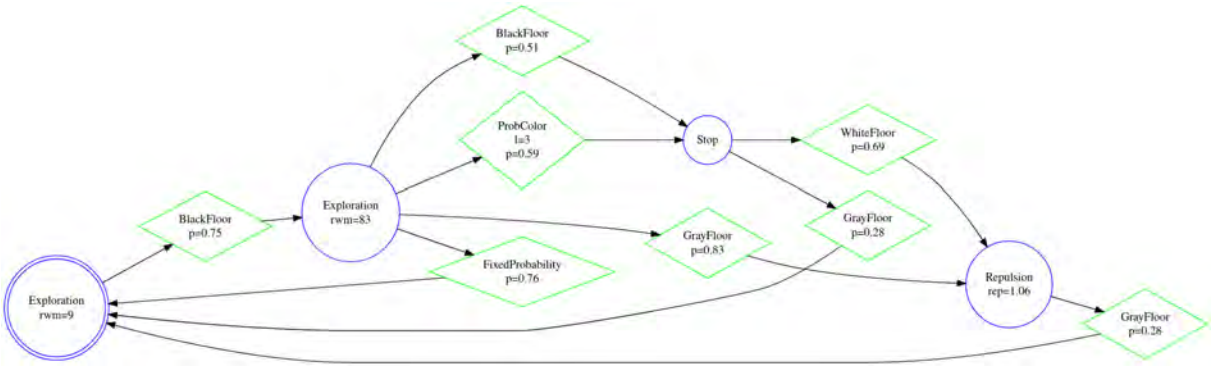



Figure A.3: FSM 3 for the aggregation mission

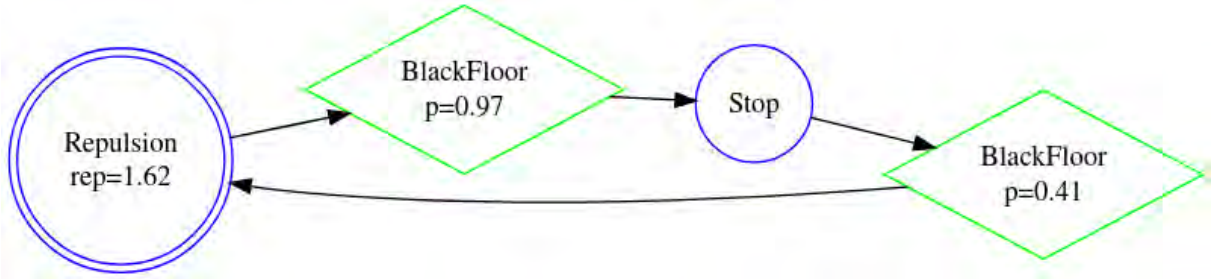



Figure A.4: FSM 4 for the aggregation mission

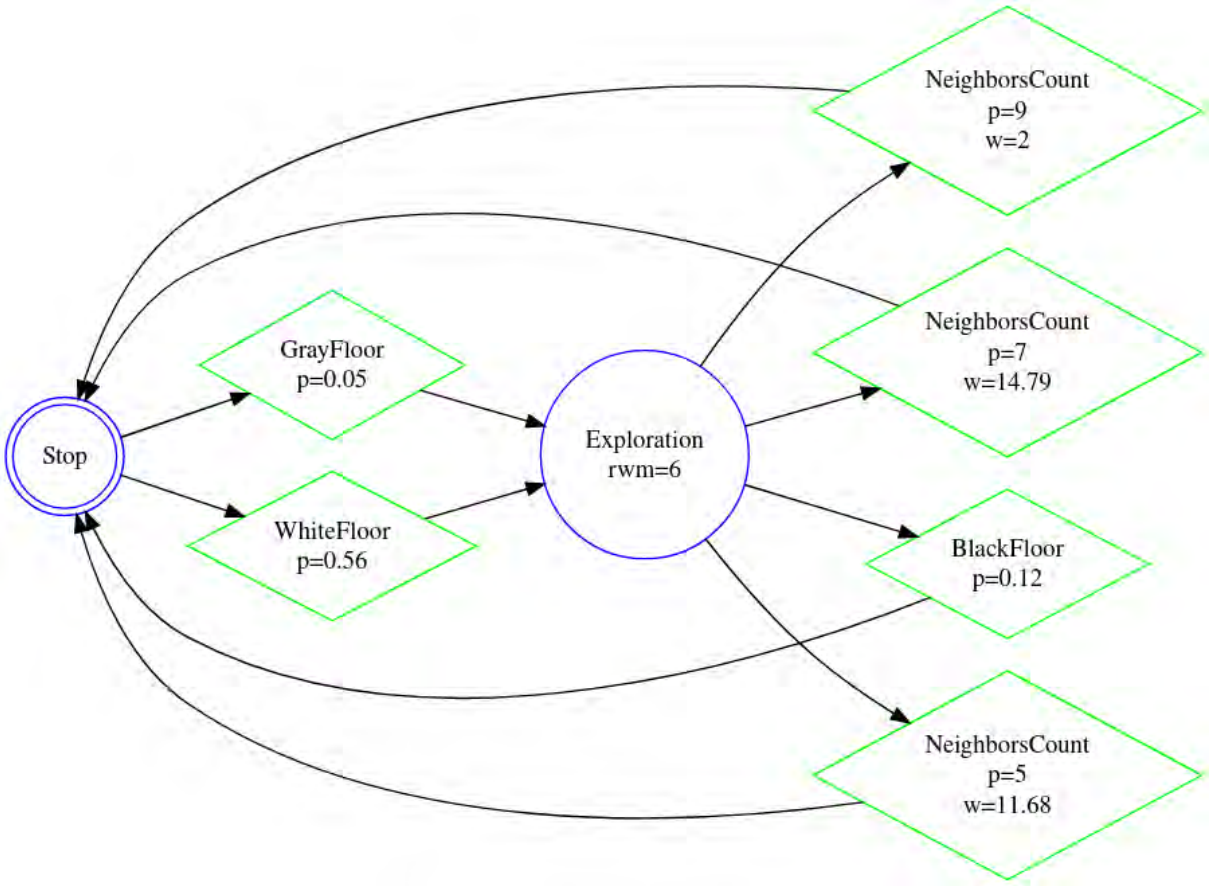



Figure A.5: FSM 5 for the aggregation mission

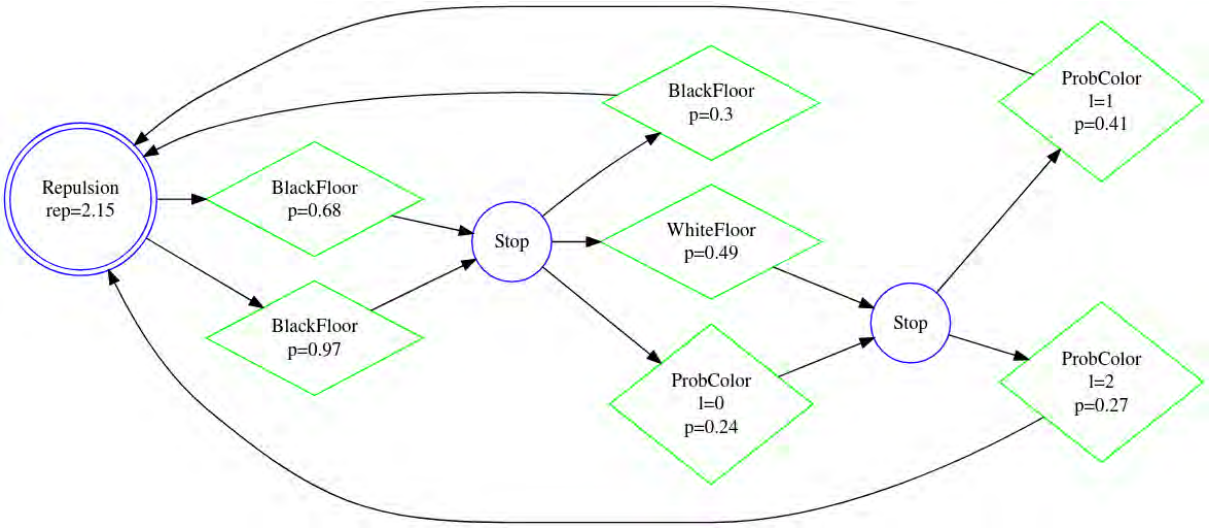



Figure A.6: FSM 6 for the aggregation mission

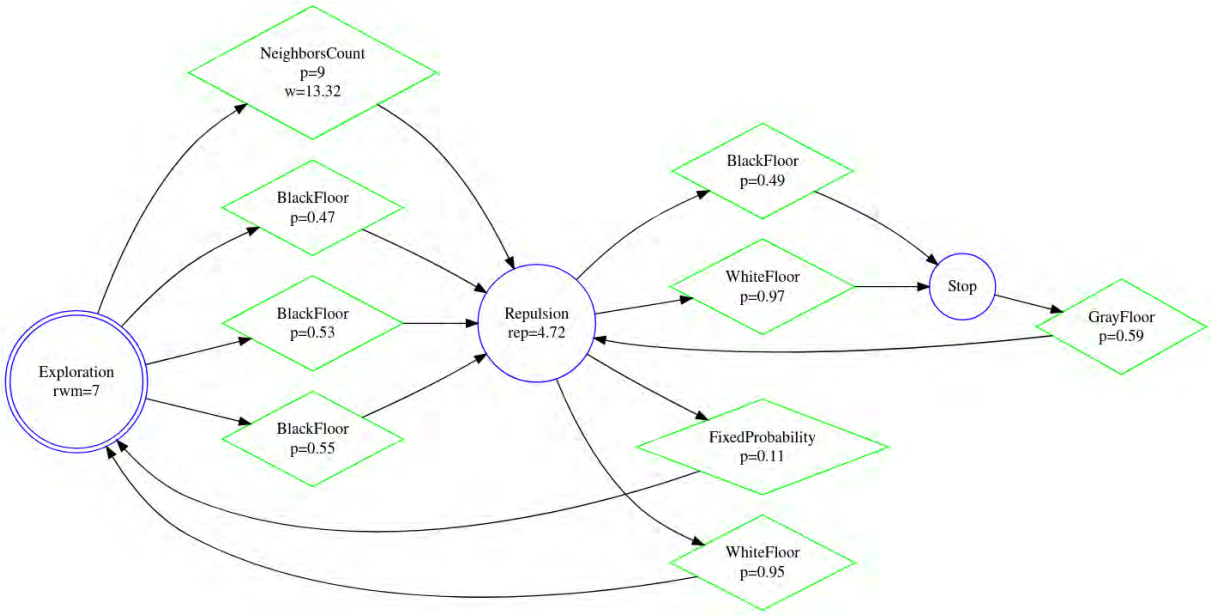



Figure A.7: FSM 7 for the aggregation mission

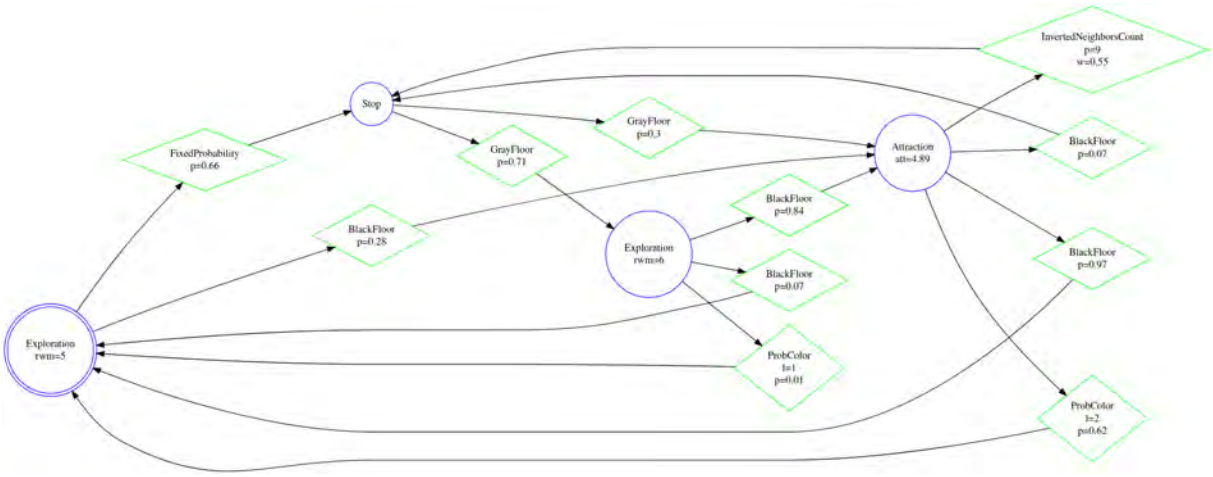



Figure A.8: FSM 8 for the aggregation mission

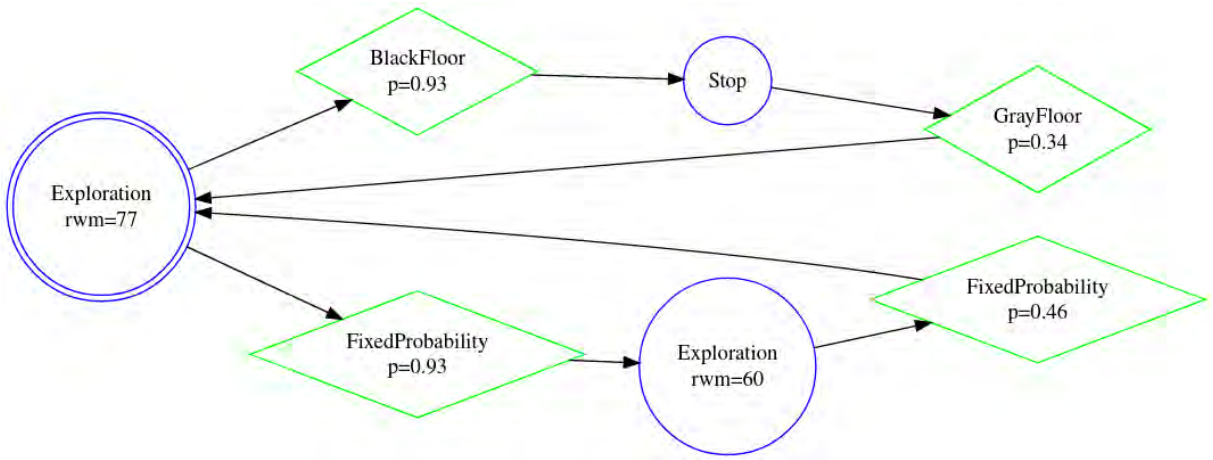



Figure A.9: FSM 9 for the aggregation mission

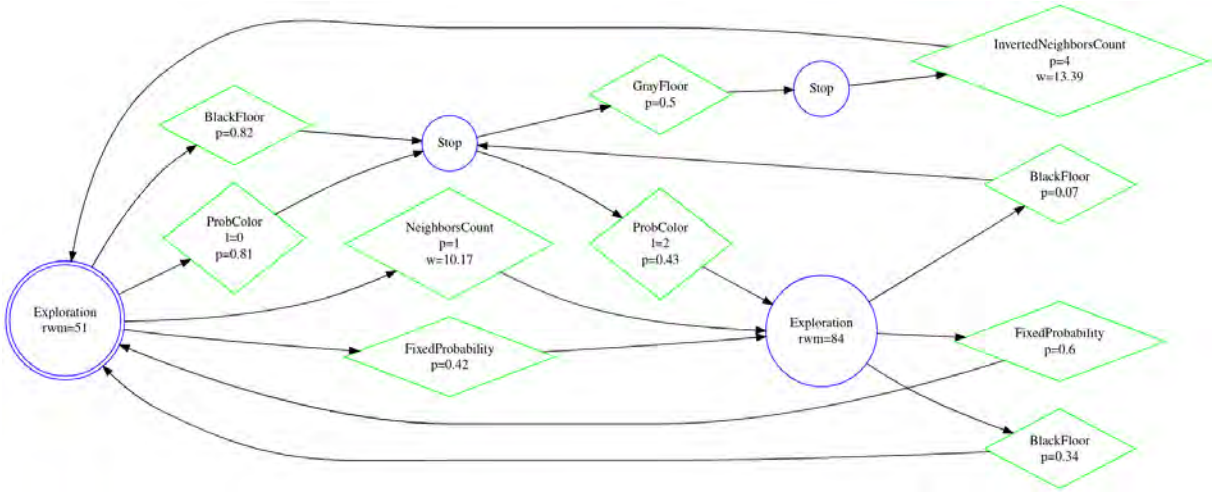



Figure A.10: FSM 10 for the aggregation mission

## A.2 Grid exploration

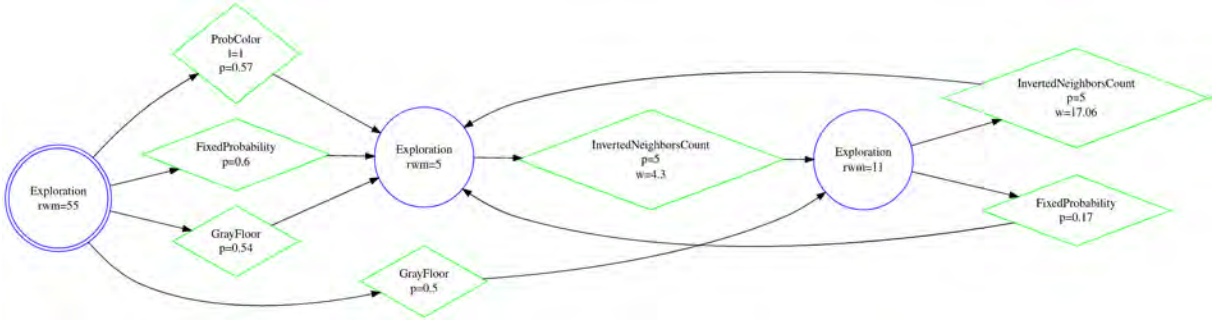



Figure A.11: FSM 1 for the grid exploration mission

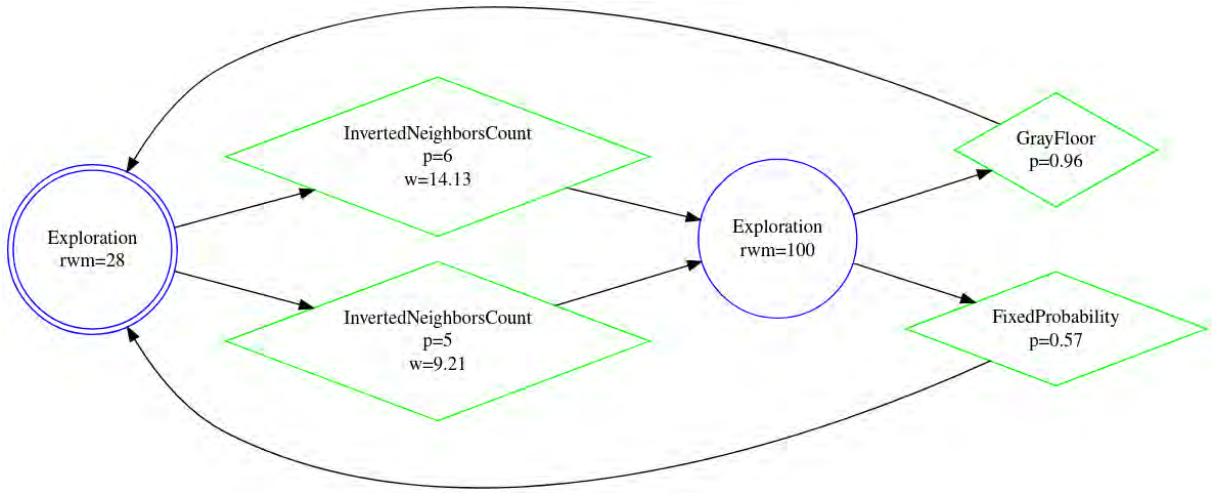



Figure A.12: FSM 2 for the grid exploration mission

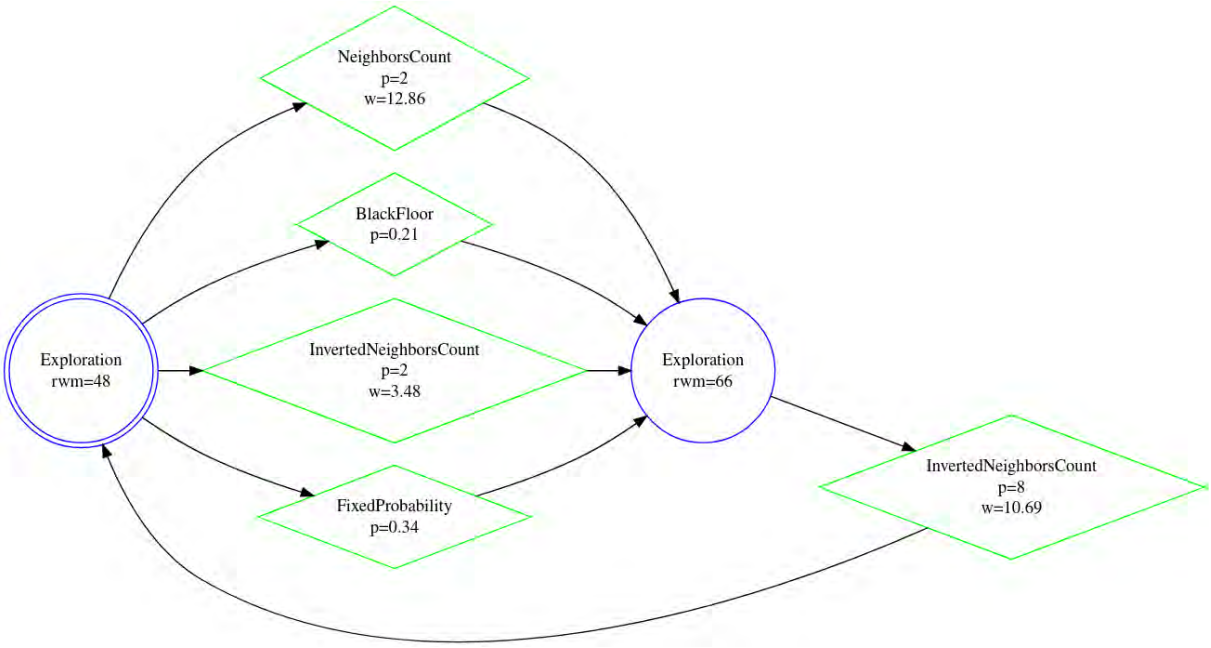



Figure A.13: FSM 3 for the grid exploration mission

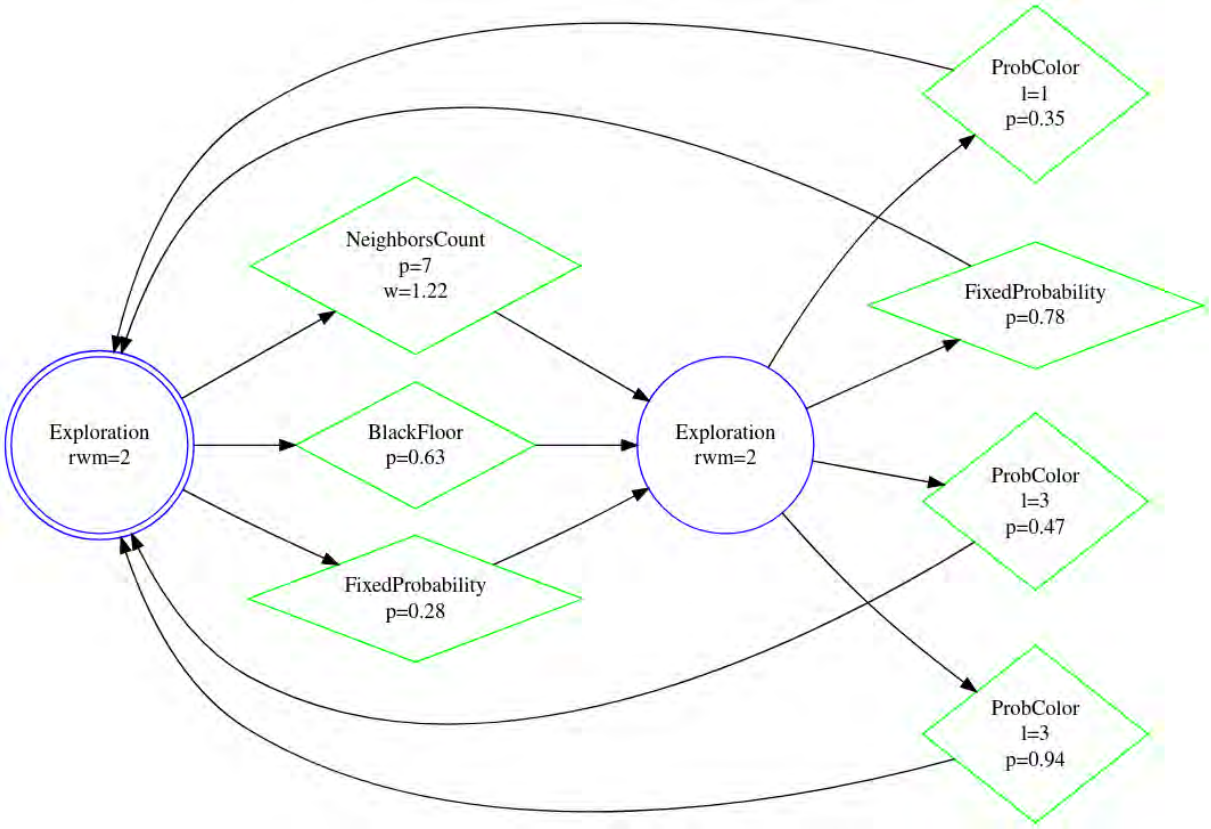



Figure A.14: FSM 4 for the grid exploration mission

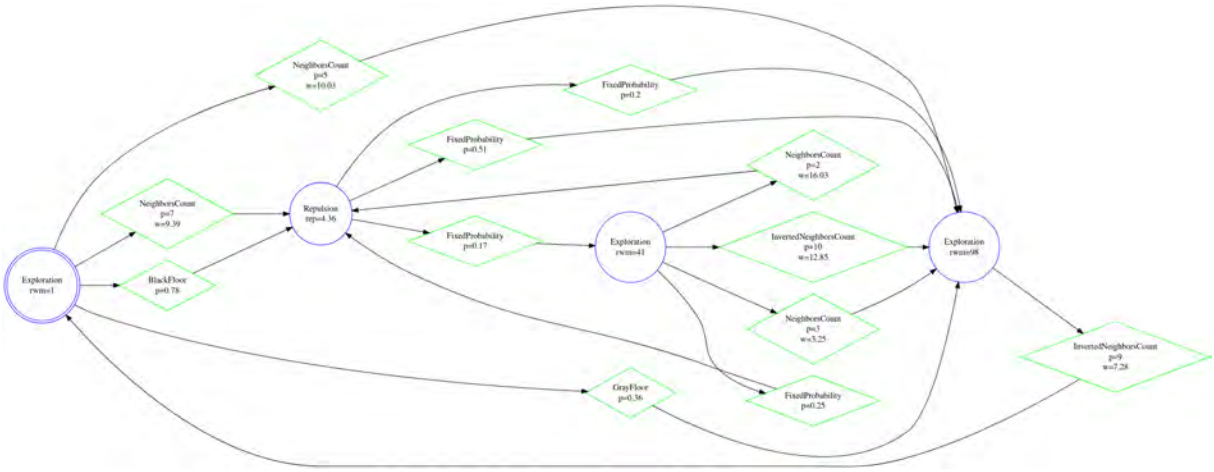


Figure A.15: FSM 5 for the grid exploration mission

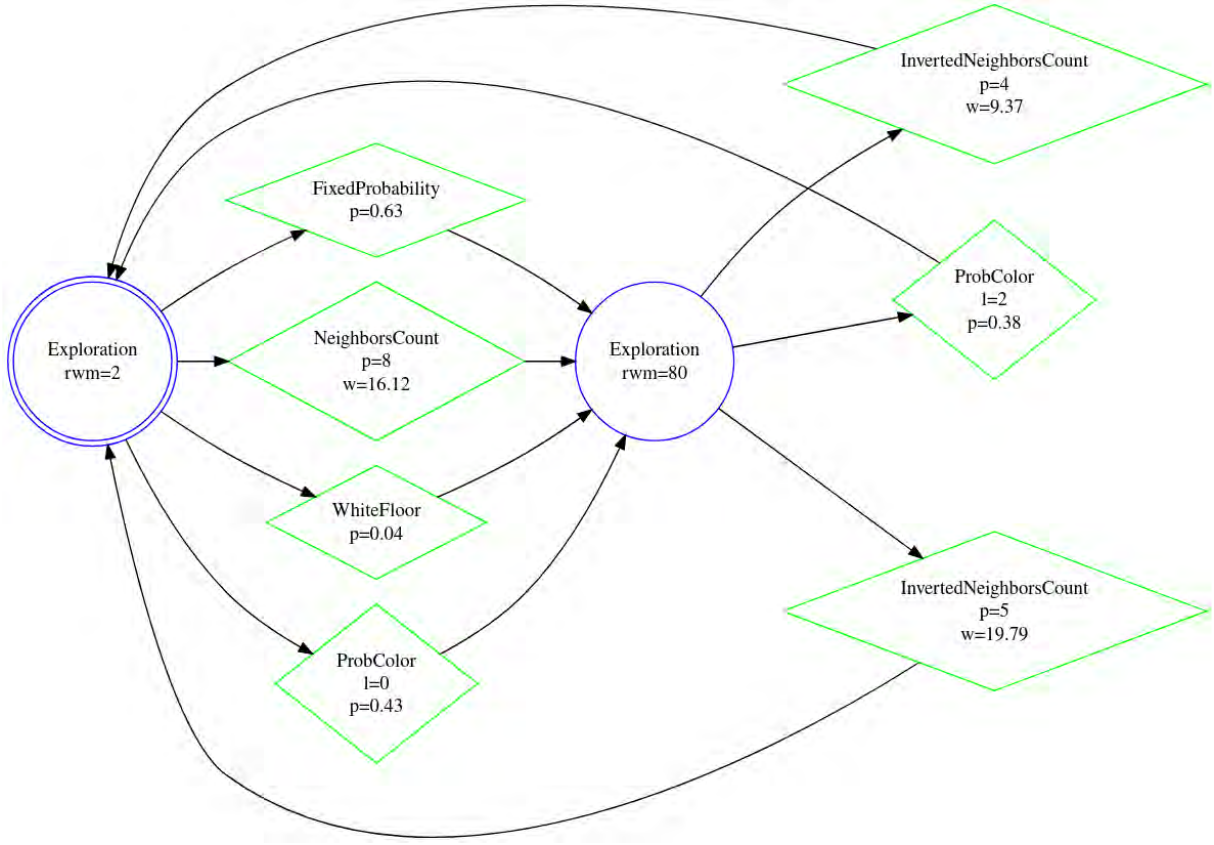



Figure A.16: FSM 6 for the grid exploration mission

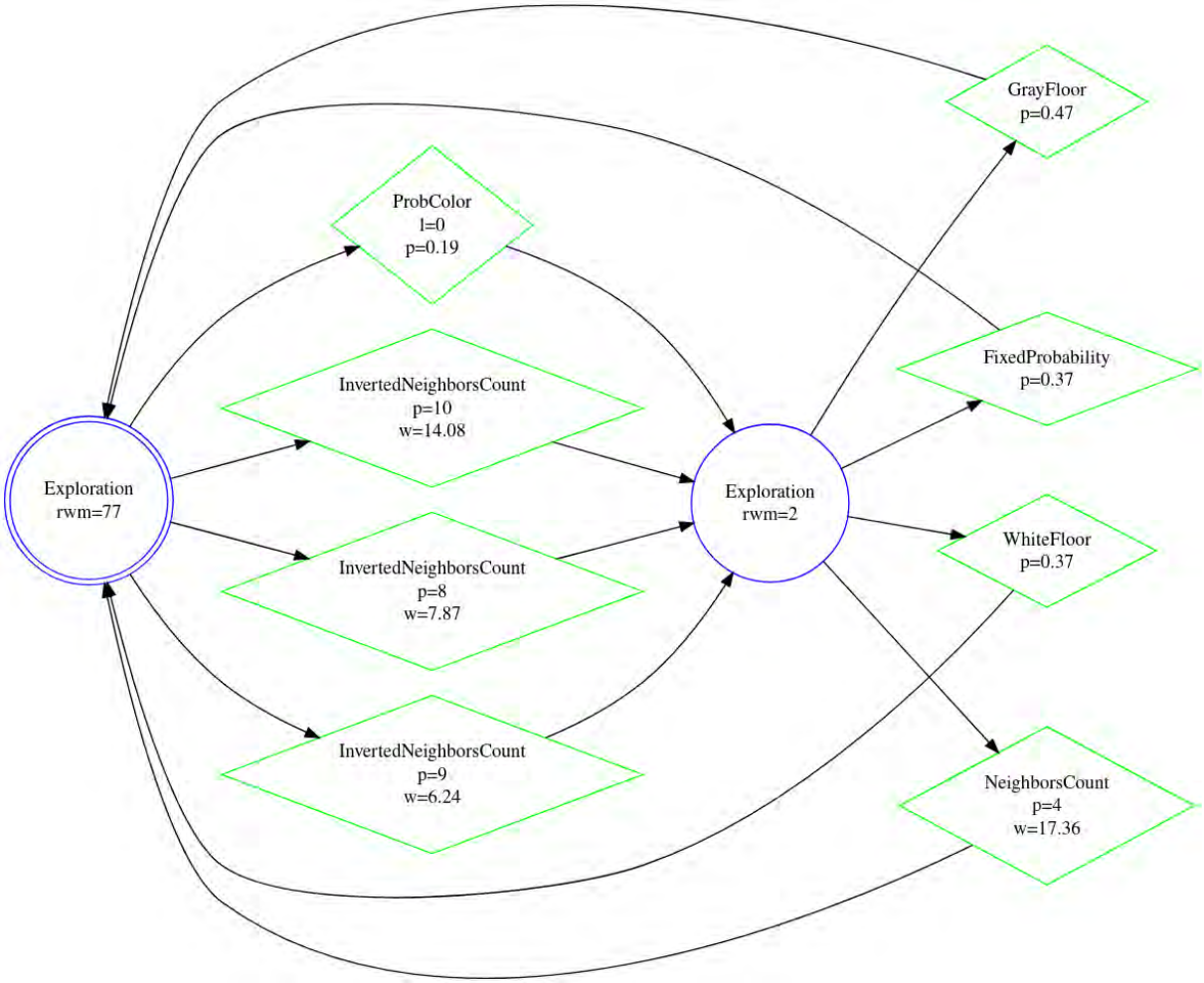



Figure A.17: FSM 7 for the grid exploration mission

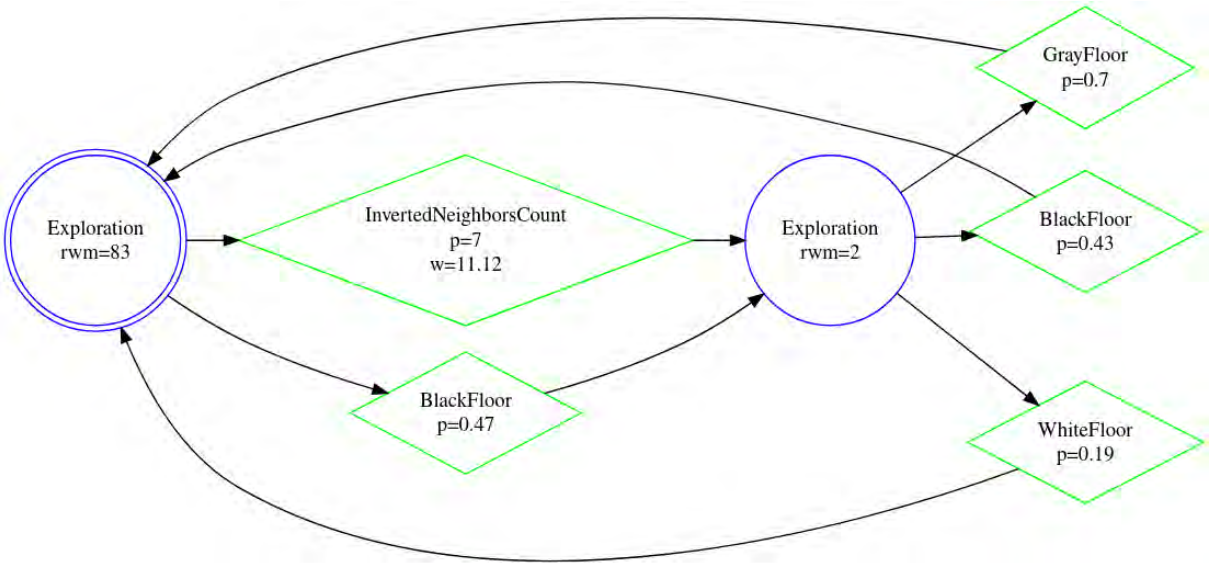



Figure A.18: FSM 8 for the grid exploration mission

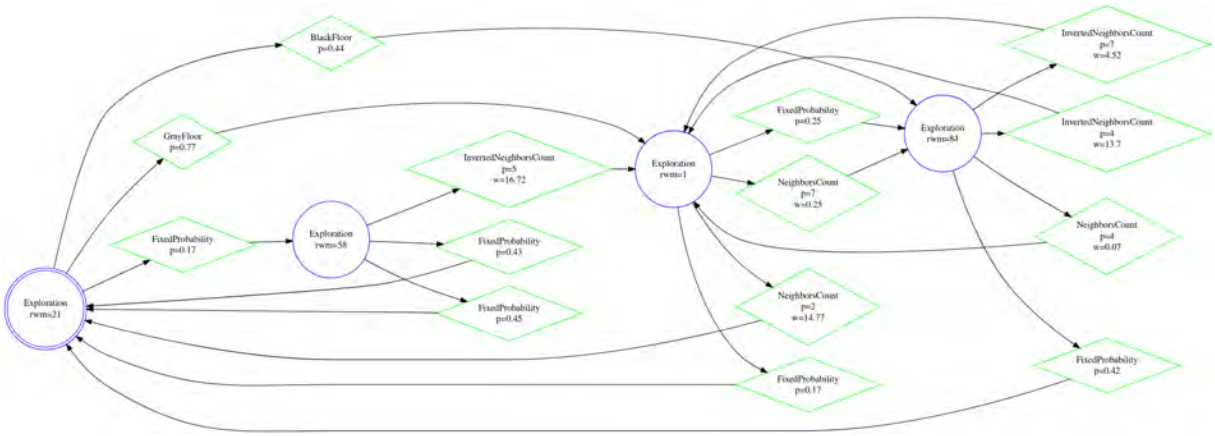



Figure A.19: FSM 9 for the grid exploration mission

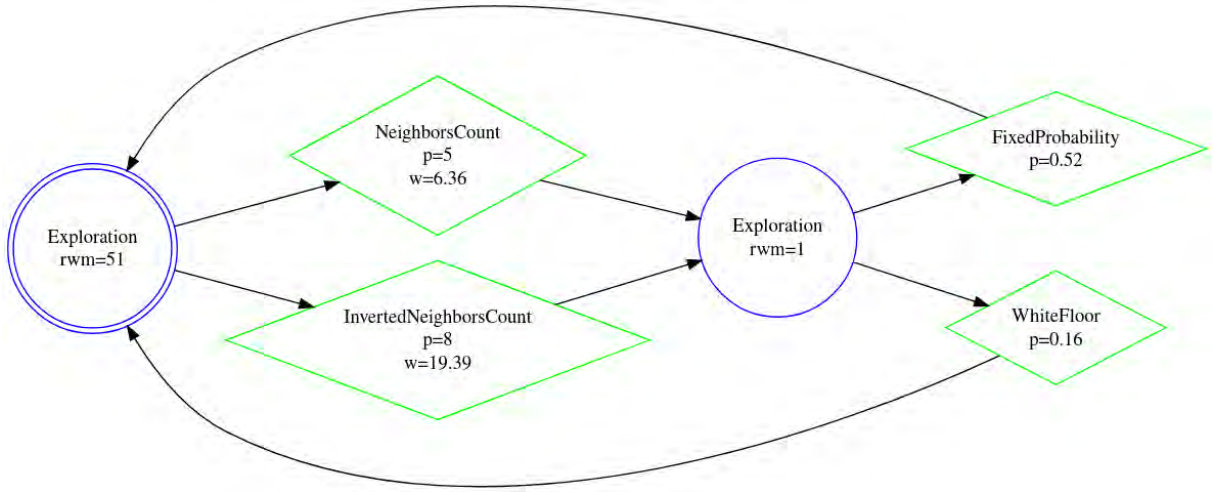



Figure A.20: FSM 10 for the grid exploration mission

## A.3 Color selection

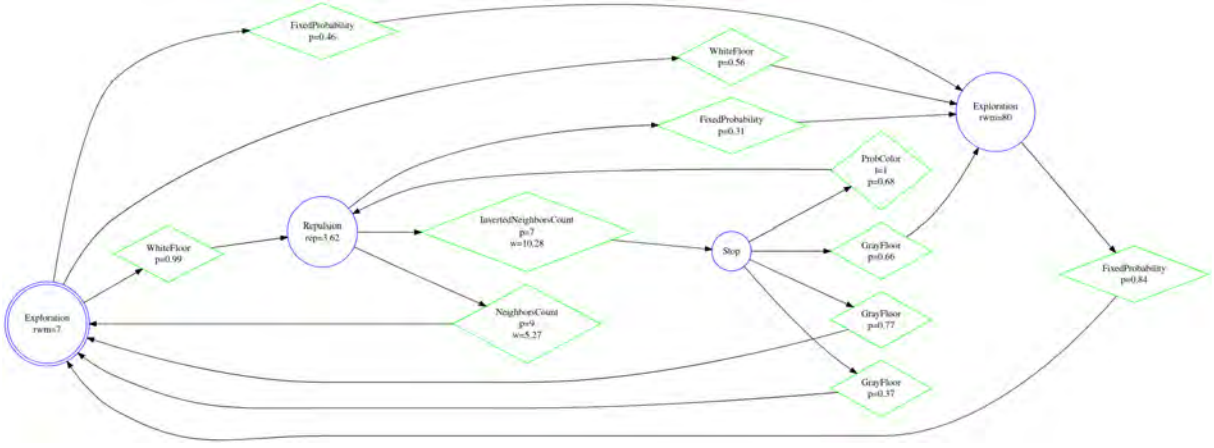



Figure A.21: FSM 1 for the color selection mission

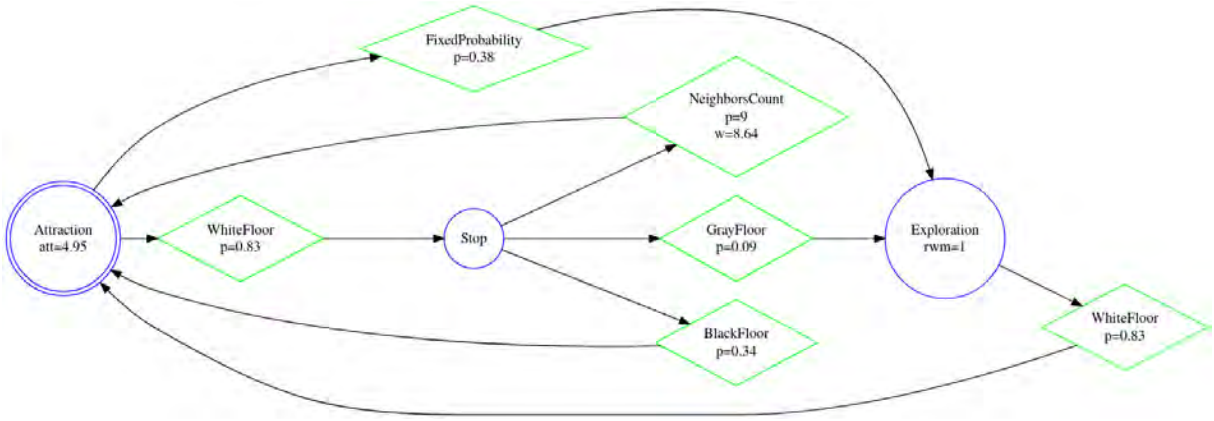



Figure A.22: FSM 2 for the color selection mission

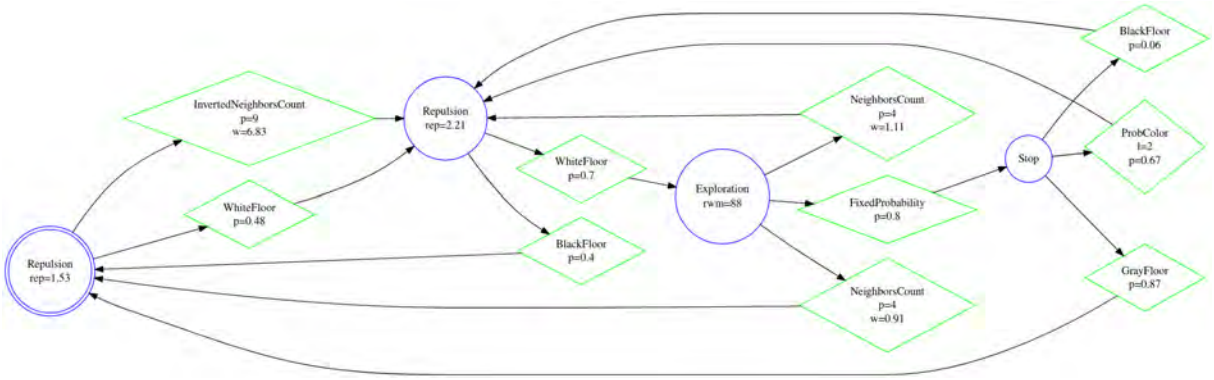



Figure A.23: FSM 3 for the color selection mission

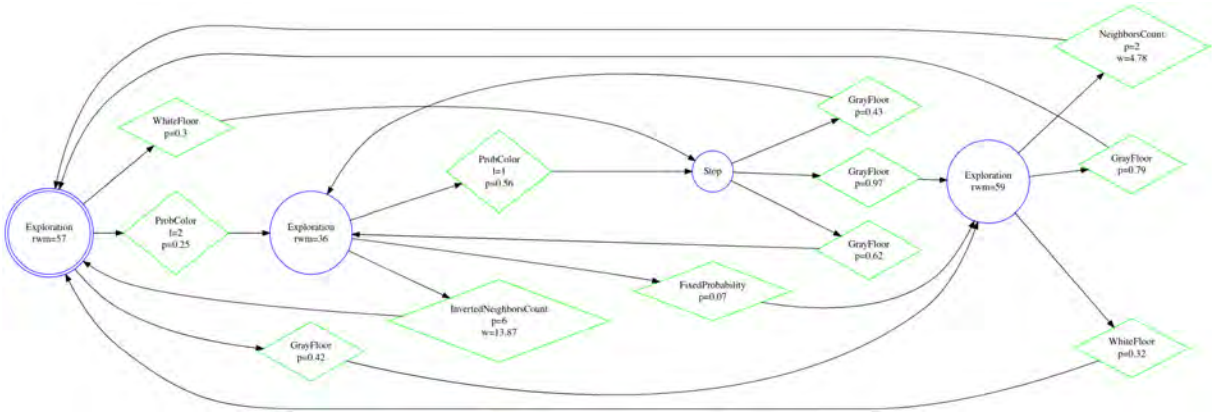



Figure A.24: FSM 4 for the color selection mission

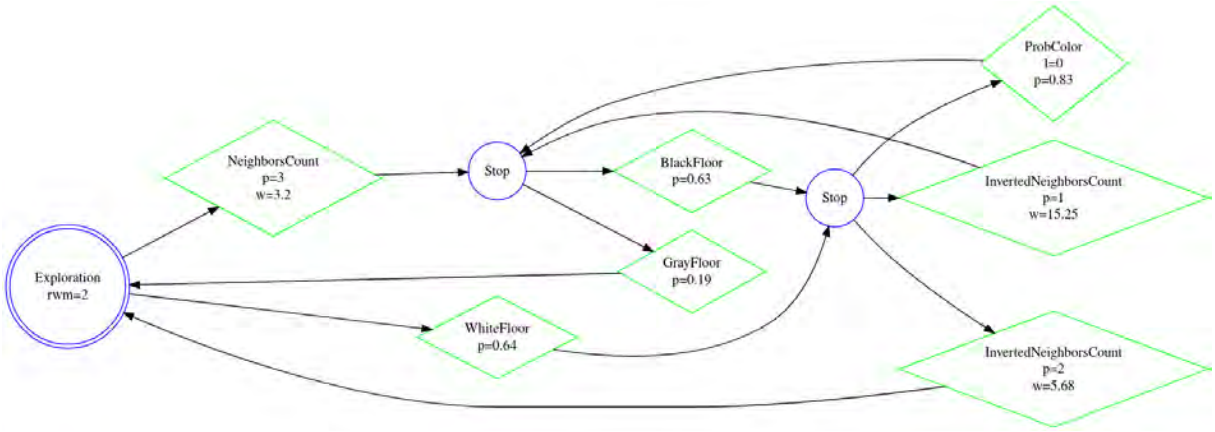



Figure A.25: FSM 5 for the color selection mission

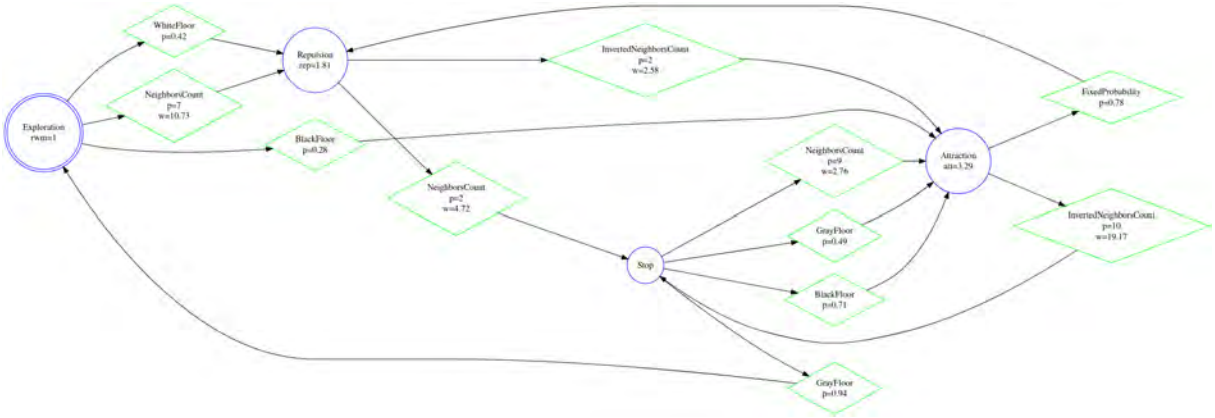


Figure A.26: FSM 6 for the color selection mission

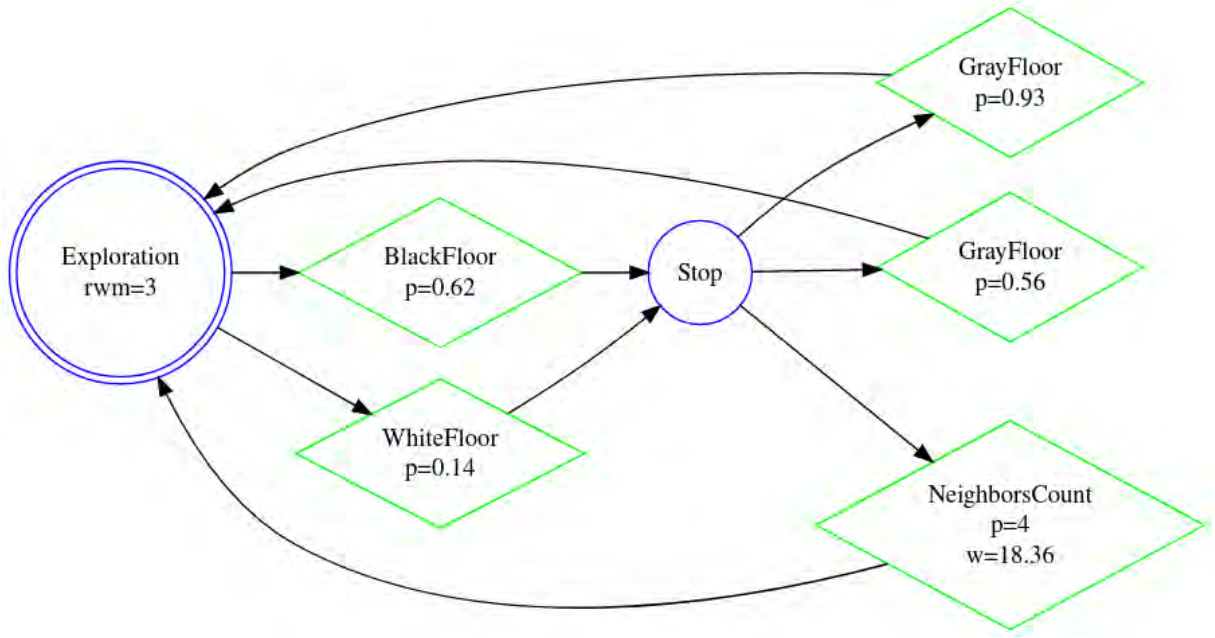



Figure A.27: FSM 7 for the color selection mission

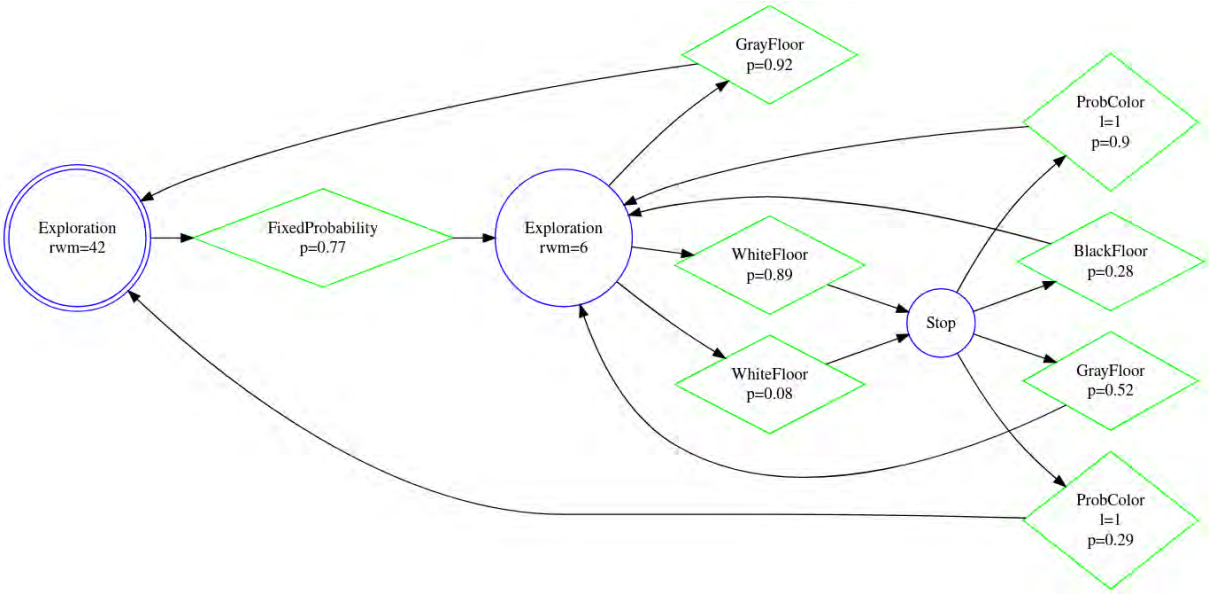



Figure A.28: FSM 8 for the color selection mission

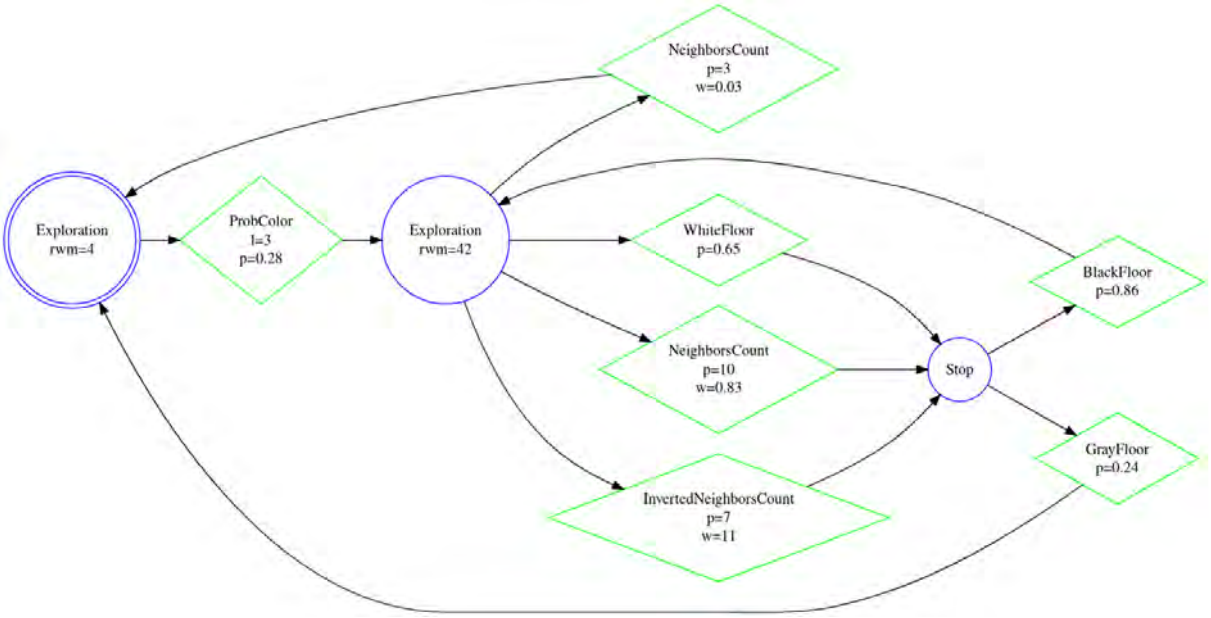



Figure A.29: FSM 9 for the color selection mission

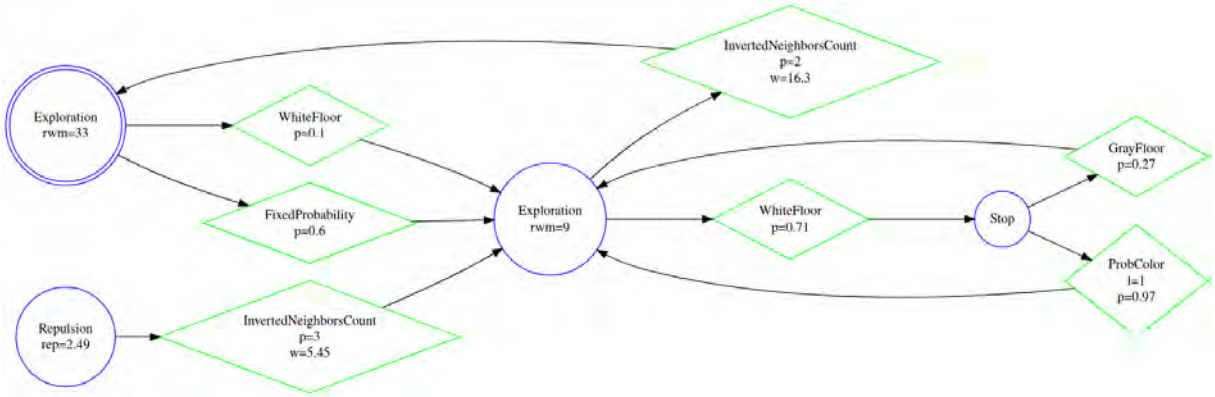



Figure A.30: FSM 10 for the color selection mission

# BIBLIOGRAPHY


[1] Muhanad H Mohammed Alkilabi, Aparajit Narayan, and Elio Tuci. "Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies". In: *Swarm intelligence* 11.3 (2017), pp. 185–209.

[2] Arduino. *Arduino Uno Rev3 — Arduino Official Store*. URL: http://store.arduino.cc/products/arduino-uno-rev3. Accessed on 01/03/2022.

[3] Jasmine Azadani et al. "SPHERiCAR". B.S. thesis. 2019.

[4] Avishek Banerjee et al. "Construction of Effective Wireless Sensor Network for Smart Communication Using Modified Ant Colony Optimization Technique". In: *Advanced Computing and Intelligent Technologies*. Ed. by Monica Bianchini et al. Singapore: Springer Singapore, 2022, pp. 269–278. ISBN: 978-981-16-2164-2.

[5] Naya Baslan et al. "Navigation and Vision System of a Mobile Robot". In: *2018 19th International Conference on Research and Education in Mechatronics (REM)*. 2018, pp. 99–104. DOI: 10.1109/REM.2018.8421777.

[6] Levent Bayındır. "A review of swarm robotics tasks". In: *Neurocomputing* 172 (2016), pp. 292–321. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.05.116. URL: https://www.sciencedirect.com/science/article/pii/S0925231215010486.

[7] Manuel Bernal and Javier Civera. "LoCoQuad: A Low-Cost Arachnoid Quadruped Robot for Research and Education". In: *arXiv preprint arXiv:2003.09025* (2020).

[8] Mauro Birattari. *Notes on the Estimation of the Expected Performance of Automatic Methods for the Design of Control Software for Robot Swarms*. fre. 2020.

[9] Mauro Birattari et al. "F-Race and Iterated F-Race: An Overview". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by Thomas Bartz-Beielstein et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 311–336. ISBN: 978-3-642-02538-9. DOI: 10.1007/978-3-642-02538-9_13. URL: https://doi.org/10.1007/978-3-642-02538-9_13.

[10] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. eng. Santa Fe Institute Studies on the Sciences of Complexity. New York: Oxford University Press, 1999, p. 320. ISBN: 9780195131581. DOI: 10.1093/oso/9780195131581.001.0001. URL: https://doi.org/10.1093/oso/9780195131581.001.0001.

[11] Manuele Brambilla et al. "Swarm robotics: a review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1 (Mar. 2013), pp. 1–41. ISSN: 1935-3820. DOI: 10.1007/s11721-012-0075-2. URL: https://doi.org/10.1007/s11721-012-0075-2.

[12] Canonical. *Get Ubuntu Server | Download | Ubuntu*. URL: https://ubuntu.com/download/server. Accessed on 03/03/2022.

[13] G. Caprari and R. Siegwart. “Mobile micro-robots ready to use: Alice”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 3295–3300. DOI: 10.1109/IROS.2005.1545568.

[14] CARMEN. *CARMEN*. http://carmen.sourceforge.net/. Accessed on 03/03/2022.

[15] Hande Celikkanat. “Optimization of felf-organized flocking of a robot swarm via evolutionary strategies”. In: *2008 23rd International Symposium on Computer and Information Sciences*. 2008, pp. 1–4. DOI: 10.1109/ISCIS.2008.4717880.

[16] Amitava Chatterjee et al. “A particle-swarm-optimized fuzzy-neural network for voice-controlled robot systems”. In: *IEEE Transactions on Industrial Electronics* 52.6 (2005), pp. 1478–1489.

[17] Christopher M Cianci et al. “Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics”. In: *International Workshop on Swarm Robotics*. Springer. 2006, pp. 103–115.

[18] K-Team Corporation. *Kilobot - K-Team Corporation*. URL: https://www.k-team.com/mobile-robotics-products/kilobot. Accessed on 02/03/2022.

[19] Demiurge. *Plugin for argos3 containing a model of the e-puck robot extended with ground sensor, range-and-bearing board, Omnivision module, and Linux board.* URL: https://github.com/demiurge-project/argos3-epuck. Accessed on 05/03/2022.

[20] Marco Dorigo. “SWARM-BOT: An experiment in swarm robotics”. In: *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* IEEE. 2005, pp. 192–200.

[21] Marco Dorigo. “Swarm-Bots and Swarmanoid: Two Experiments in Embodied Swarm Intelligence.” In: *Web intelligence*. 2009, pp. 2–3.

[22] Marco Dorigo, Mauro Birattari, and Thomas Stützle. “Ant Colony Optimization”. In: *Computational Intelligence Magazine, IEEE* 1 (Dec. 2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[23] Ernesto Fabregas et al. “Teaching control in mobile robotics with V-REP and a Khepera IV library”. In: *2016 IEEE conference on Control Applications (CCA)*. IEEE. 2016, pp. 821–826.

[24] G Farias et al. “A Khepera IV library for robotic control education using V-REP”. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 9150–9155.

[25] Raspberry Pi Foundation. *Operating system images – Raspberry Pi*. URL: https://www.raspberrypi.com/software/operating-systems/. Accessed on 03/03/2022.

[26] Raspberry Pi Foundation. *Raspberry Pi 4 Model B – Raspberry Pi*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed on 01/03/2022.

[27] Gianpiero Francesca et al. “An Experiment in Automatic Design of Robot Swarms”. In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Cham: Springer International Publishing, 2014, pp. 25–37. ISBN: 978-3-319-09952-1.

[28] Gianpiero Francesca et al. “AutoMoDe-Chocolate: automatic design of control software for robot swarms”. In: *Swarm Intelligence* 9 (2015), pp. 125–152.

[29] Gianpiero Francesca et al. “AutoMoDe: A novel approach to the automatic design of control software for robot swarms”. In: *Swarm Intelligence* 8.2 (June 2014), pp. 89–112. ISSN: 1935-3820. DOI: 10.1007/s11721-014-0092-4. URL: https://doi.org/10.1007/s11721-014-0092-4.

[30] David Garzón Ramos and Mauro Birattari. “Automatic Design of Collective Behaviors for Robots that Can Display and Perceive Colors”. In: *Applied Sciences* 10 (July 2020), p. 4654. DOI: 10.3390/app10134654.

[31] David Garzón Ramos and Mauro Birattari. “Automatic Design of Collective Behaviors for Robots that Can Display and Perceive Colors”. In: *Applied Sciences* 10.13 (2020). ISSN: 2076-3417. DOI: 10.3390/app10134654. URL: https://www.mdpi.com/2076-3417/10/13/4654.

[32] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems”. In: *In Proceedings of the 11th International Conference on Advanced Robotics*. 2003, pp. 317–323.

[33] Paulo Gonçalves et al. “The e-puck, a Robot Designed for Education in Engineering”. In: *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions* 1 (Jan. 2009).

[34] Roderich Groß and Marco Dorigo. “Towards Group Transport by Swarms of Robots”. In: *International Journal of Bio-Inspired Computation* 1 (Jan. 2009). DOI: 10.1504/IJBIC.2009.022770.

[35] Álvaro Gutiérrez et al. “Open e-puck range & bearing miniaturized board for local communication in swarm robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3111–3116.

[36] Heiko Hamann, Marc Szymanski, and Heinz Wörn. “Orientation in a Trail Network by Exploiting its Geometry for Swarm Robotics.” In: *SIS*. 2007, pp. 310–315.

[37] Ken Hasselmann, Frédéric Robert, and Mauro Birattari. “Automatic Design of Communication-Based Behaviors for Robot Swarms”. In: Oct. 2018, pp. 16–29. ISBN: 978-3-030-00532-0. DOI: 10.1007/978-3-030-00533-7_2.

[38] Hengjing He et al. “Cloud based real-time multi-robot collision avoidance for swarm robotics”. In: *International Journal of Grid and Distributed Computing* 9.6 (2016), pp. 339–358.

[39] Jared Jackson. “Microsoft robotics studio: A technical introduction”. In: *IEEE Robotics Automation Magazine* 14.4 (2007), pp. 82–87. DOI: 10.1109/M-RA.2007.905745.

[40] C. Jones and M.J. Mataric. “Adaptive division of labor in large-scale minimalist multi-robot systems”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 2. 2003, 1969–1974 vol.2. DOI: 10.1109/IROS.2003.1248936.

[41] Kiattisin Kanjanawanishkul. “Map-based Navigation of a Mobile Robot Using MRPT Modules”. In: *Applied Mechanics and Materials*. Vol. 574. Trans Tech Publ. 2014, pp. 505–510.

[42] Serge Kernbach, O. Kornienko, and Paul Levi. “Minimalistic Approach towards Communication and Perception in Microrobotic Swarms”. In: Sept. 2005, pp. 2228–2234. DOI: 10.1109/IROS.2005.1545594.

[43] Hala Khankhour, Otman Abdoun, and Jâafar Abouchabaka. “A New Design of an Ant Colony Optimization (ACO) Algorithm for Optimization of Ad Hoc Network”. In: *Networking, Intelligent Systems and Security*. Ed. by Mohamed Ben Ahmed et al. Singapore: Springer Singapore, 2022, pp. 231–241. ISBN: 978-981-16-3637-0.

[44] *Kilobot plugin for the multi-robot ARGoS3 simulator*. URL: https://github.com/ilpincy/argos3-kilobot. Accessed on 05/03/2022.

[45] Jonas Kuckling et al. "Behavior Trees as a Control Architecture in the Automatic Modular Design of Robot Swarms". In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Cham: Springer International Publishing, 2018, pp. 30–43. ISBN: 978-3-030-00533-7.

[46] Arjun S. Kumar et al. "Search and rescue operations using robotic darwinian particle swarm optimization". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017, pp. 1839–1843. DOI: 10.1109/ICACCI.2017.8126112.

[47] Antoine Ligot and Mauro Birattari. "Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms". en. In: *Swarm Intell.* 14.1 (Mar. 2020), pp. 1–24.

[48] M. Anwar Ma'sum et al. "Autonomous quadcopter swarm robots for object localization and tracking". In: *MHS2013*. 2013, pp. 1–6. DOI: 10.1109/MHS.2013.6710447.

[49] Ralf Mayet et al. "Antbots: A feasible visual emulation of pheromone trails for swarm robots". In: *International conference on swarm intelligence*. Springer. 2010, pp. 84–94.

[50] Olivier Michel. "WebotsTM: Professional Mobile Robot Simulation". In: *International Journal of Advanced Robotic Systems* 1 (Mar. 2004). DOI: 10.5772/5618.

[51] micro:bit. *Micro:bit Educational Foundation | micro:bit*. URL: https://microbit.org/. Accessed on 01/03/2022.

[52] Microsoft. *Robotics: Simulating the World with Microsoft Robotics Studio | Microsoft Docs*. https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/june/robotics-simulating-the-world-with-microsoft-robotics-studio. Accessed on 03/03/2022.

[53] Z. Minchev et al. "Fuzzy logic based intelligent motion control of robot swarm simulated by Khepera robots". In: *2004 2nd International IEEE Conference on 'Intelligent Systems'. Proceedings (IEEE Cat. No.04EX791)*. Vol. 1. 2004, 305–310 Vol.1. DOI: 10.1109/IS.2004.1344687.

[54] Francesco Mondada, E. Franzi, and A. Guignard. "The Development of Khepera". In: (Dec. 2010).

[55] Francesco Mondada et al. "The cooperation of swarm-bots: Physical interactions in collective robotics". In: *Robotics & Automation Magazine, IEEE* 12 (July 2005), pp. 21–28. DOI: 10.1109/MRA.2005.1458313.

[56] JG Monroy, JL Blanco, and J González-Jiménez. "An open source framework for simulating mobile robotics olfaction". In: *15th International Symposium on Olfaction and Electronic Nose (ISOEN)*. 2013, pp. 2–3.

[57] MRPT. *MRPT – Empowering C++ development in robotics*. https://www.mrpt.org/. Accessed on 03/03/2022.

[58] Venkat Raman Nagarajan and Pavan Singh. "Obstacle Detection and Avoidance For Mobile Robots Using Monocular Vision". In: *2021 8th International Conference on Smart Computing and Communications (ICSCC)*. IEEE. 2021, pp. 275–279.

[59] Iñaki Navarro and Fernando Mata. "An introduction to swarm robotics". In: *International Scholarly Research Notices* 2013 (2013).

[60] Adam Lee Newton, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. "The Robot in the Swarm: An Investigation into Agent Embodiment within Virtual Robotic Swarms". In: *Advances in Artificial Life*. Ed. by Wolfgang Banzhaf et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 829–838. ISBN: 978-3-540-39432-7.

[61] Hyondong Oh, Ataollah R. Shiraz, and Yaochu Jin. "Morphogen diffusion algorithms for tracking and herding using a swarm of kilobots". In: *Soft Computing* 22.6 (Mar. 2018), pp. 1833–1844. ISSN: 1433-7479. DOI: 10.1007/s00500-016-2182-2. URL: https://doi.org/10.1007/s00500-016-2182-2.

[62] E Peralta et al. "Development of a Khepera IV Library for the V-REP Simulator". In: *IFAC-PapersOnLine* 49.6 (2016), pp. 81–86.

[63] Nuno Pereira et al. "ARENA: The Augmented Reality Edge Networking Architecture". In: *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2021, pp. 479–488.

[64] Giovanni C. Pettinaro, Ivo W. Kwee, and Luca M. Gambardella. "Swarmbot3D: a New Simulating Tool for Swarm Robotics Investigations". In: *Proceedings of the 5th EuroSim Congress on Modelling and Simulations (Eurosim '04)*. ESIEE Paris Cité Descartes, Marne La Valée, France, Sept. 2004.

[65] Carlo Pinciroli. *Integration between ARGoS3 and the Khepera IV robot.* URL: https://github.com/ilpincy/argos3-kheperaiv. Accessed on 05/03/2022.

[66] Carlo Pinciroli et al. "ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems". In: *Swarm Intelligence* 6.4 (2012), pp. 271–295.

[67] Carlo Pinciroli et al. "Simulating Kilobots Within ARGoS: Models and Experimental Validation". In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Cham: Springer International Publishing, 2018, pp. 176–187. ISBN: 978-3-030-00533-7.

[68] Lenka Pitonakova et al. "Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators". In: *Towards Autonomous Robotic Systems*. Ed. by Manuel Giuliani, Tareq Assaf, and Maria Elena Giannaccini. Cham: Springer International Publishing, 2018, pp. 357–368. ISBN: 978-3-319-96728-8.

[69] Jim Pugh and Alcherio Martinoli. "Multi-robot learning with particle swarm optimization". In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 441–448.

[70] ROS. *ROS: Home*. Accessed on 03/03/2022.

[71] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm". In: *Science* 345.6198 (2014), pp. 795–799. DOI: 10.1126/science.1254295. eprint: https://www.science.org/doi/pdf/10.1126/science.1254295. URL: https://www.science.org/doi/abs/10.1126/science.1254295.

[72] Erol Şahin. "Swarm Robotics: From Sources of Inspiration to Domains of Application". In: *Swarm Robotics*. Ed. by Erol Şahin and William M. Spears. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 10–20. ISBN: 978-3-540-30552-1.

[73] Erol Şahin et al. "Swarm Robotics". In: *Swarm Intelligence: Introduction and Applications*. Ed. by Christian Blum and Daniel Merkle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 87–100. ISBN: 978-3-540-74089-6. DOI: 10.1007/978-3-540-74089-6_3. URL: https://doi.org/10.1007/978-3-540-74089-6_3.

[74] Thomas Schmickl and Karl Crailsheim. “A navigation algorithm for swarm robotics inspired by slime mold aggregation”. In: *International Workshop on Swarm Robotics*. Springer. 2006, pp. 1–13.

[75] Rajesh Singh et al. “A Review on Implementation of Robotic Assistance in Covid-19 Epidemics: A Possibility Check”. In: (2020).

[76] Jorge M. Soares, Iñaki Navarro, and Alcherio Martinoli. “The Khepera IV Mobile Robot: Performance Evaluation, Sensory Data and Software Toolbox”. In: *Robot 2015: Second Iberian Robotics Conference*. Ed. by Luís Paulo Reis et al. Cham: Springer International Publishing, 2016, pp. 767–781. ISBN: 978-3-319-27146-0.

[77] Gaëtan Spaey et al. “Evaluation of Alternative Exploration Schemes in the Automatic Modular Design of Robot Swarms”. In: *Artificial Intelligence and Machine Learning*. Ed. by Bart Bogaerts et al. Cham: Springer International Publishing, 2020, pp. 18–33. ISBN: 978-3-030-65154-1.

[78] Sphero. *Educational Coding Robot Car | Learn Coding & STEM | Sphero RVR*. https://sphero.com/products/rvr?_pos=11&_sid=1f95ddb4f&_ss=r. 2022. Accessed on 03/03/2022.

[79] Sphero. *Sphero RVR SDK to run on Raspberry Pi using Python*. URL: https://github.com/sphero-inc/sphero-sdk-raspberrypi-python. Accessed on 01/03/2022.

[80] Marc Szymanski et al. “Distributed shortest-path finding by a micro-robot swarm”. In: *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer. 2006, pp. 404–411.

[81] Terabee. *Sensing Kit | 8 Sensor Modules | Custom Configurations*. URL: https://www.terabee.com/shop/lidar-tof-range-finders/teraranger-multiflex/. Accessed on 01/03/2022.

[82] Ali Turgut et al. “Kobot: A mobile robot designed specifically for swarm robotics research”. In: (Jan. 2007).

[83] Ali E Turgut et al. “Self-organized flocking in mobile robot swarms”. In: *Swarm Intelligence* 2.2 (2008), pp. 97–120.

[84] Zoltán Tuza, János Rudan, and Gábor Szederkényi. “Developing an integrated software environment for mobile robot navigation and control”. In: *2010 International Conference on Indoor Positioning and Indoor Navigation*. 2010, pp. 1–6. DOI: 10.1109/IPIN.2010.5647506.

[85] Mohamed H. Wagdy, Hady A. Khalil, and Shady A. Maged. “Swarm Robotics Pattern Formation Algorithms”. In: *2020 8th International Conference on Control, Mechatronics and Automation (ICCMA)*. 2020, pp. 12–17. DOI: 10.1109/ICCMA51325.2020.9301540.

[86] Xiaoshu Xiang et al. “Demand coverage diversity based ant colony optimization for dynamic vehicle routing problems”. In: *Engineering Applications of Artificial Intelligence* 91 (2020), p. 103582. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2020.103582. URL: https://www.sciencedirect.com/science/article/pii/S0952197620300592.

[87] *YDLIDAR X4_YDLIDAR|Focus on lidar sensor solutions*. URL: https://www.ydlidar.com/products/view/5.html. Accessed on 01/03/2022.

[88] Guohua Ye et al. “Managing group behaviors in swarm systems by associations”. In: *2006 American Control Conference*. 2006, 8 pp.-. DOI: 10.1109/ACC.2006.1657266.

[89] Yanqi Zhang, Bo Zhang, and Xiaodong Yi. “The Design and Implementation of Swarm-Robot Communication Analysis Tool”. In: *Geo-Spatial Knowledge and Intelligence*. Ed. by Hanning Yuan et al. Singapore: Springer Singapore, 2018, pp. 631–640. ISBN: 978-981-13-0896-3.