ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Robot recognition using a 360-degree vision module for swarm robots

A new view on swarm robotics

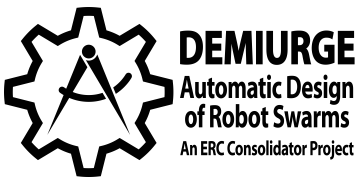
Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en Informatique à finalité spécialisée

Franck Trouillez

Directeur
Professeur Mauro Birattari

Superviseur
Miquel Kegeleirs

Service
IRIDIA



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

Acknowledgements

Throughout the writing of this master thesis, I have received a great deal of assistance and support.

First, I would like to thank Professor Mauro Birattari, Research Director of the fund for scientific research F.R.S.-FNRS affiliated with IRIDIA, which helped me to define the subject of this thesis and which trusted me for this work, by considering me as a researcher for the lab during this year.

I would also like to acknowledge my supervisor, Miquel Kegeleirs, which followed my work and helped me in improving the work I was producing. Your feedback really helped me to increase my level of knowledge in swarm robotics and in research in general. It also greatly improved my skills in terms of writing skills and critical thinking.

I would like to thank David Garzón Ramos, which also closely followed my master thesis and gave me very relevant advice. Your expertise and critical thinking made me become aware of a lot of aspects in research that I did not know existed. This also allowed me to sharpen my critical and scientific thinking.

I would also like to thank all the researchers of IRIDIA for their help during this thesis, which include in particular Guillermo Legarda Herranz and Jonas Kuckling. Indeed, your expertise helped me to understand concepts in my research. I am grateful to you for your comments and valuable help on this thesis.

Finally, I want to express my very profound gratitude to my parents and to Maëlle for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Franck Trouillez

Résumé

Robot recognition using a 360-degree vision module for swarm robots

La robotique en essaim est actuellement un domaine de recherche prometteur. En effet, elle vise à réaliser des tâches complexes avec un groupe de robots simples, en utilisant quelques capteurs. Cependant, il existe une limite quant aux informations pouvant être récupérées par un essaim de robots, en raison du peu de capteurs utilisés par chaque robot. Une façon d'augmenter les informations pour chaque robot, et par conséquent pour l'ensemble de l'essaim, est d'ajouter un module de vision.

Les modules de vision sont actuellement assez chers et ce coût est multiplié pour un essaim de robots. Ce mémoire vise à concevoir un module de vision à 360 degrés à un prix raisonnable pour un essaim de robots *Sphero RVR*, et à utiliser ce module de vision pour identifier les autres robots de l'essaim. Le module de vision est composé d'un microprocesseur et de différentes caméras.

Trois méthodes différentes sont considérées pour permettre la reconnaissance des robots. Premièrement, une segmentation par couleur utilise les LEDs des robots *Sphero RVR* pour estimer leur position. Ensuite, des marqueurs *ArUco* placés autour du robot permettent de localizer les autres robots. Enfin, un réseau neuronal entraîné sur un jeu de données créé pour la reconnaissance des robots *Sphero RVR* dans le contexte de la robotique en essaim est utilisé pour identifier le reste du groupe. Ces trois méthodes sont évaluées à l'aide d'un jeu de données expérimental. Des expériences sont également menées pour évaluer les performances de ces algorithmes dans une application en temps réel.

Les résultats ont montré que les opérations en temps réel impliquant la vision par ordinateur nécessitent beaucoup de ressources. La bibliothèque *ArUco* est difficile à utiliser lorsque les robots se déplacent trop rapidement, en raison du flou de mouvement. La méthode utilisant un réseau neuronal est capable de détecter les robots en mouvement, mais prend plus de temps, ce qui la rend actuellement difficile à utiliser dans une application en temps réel.

Mots-clés : module de vision, robotique en essaim, reconnaissance de robots

Mémoire présenté pour en vue de l'obtention du diplôme d'Ingénieur civil en Informatique à finalité spécialisée
Franck Trouillez, 2021-2022

Abstract

Robot recognition using a 360-degree vision module for swarm robots

Swarm robotics is currently a promising research field. Indeed, it aims at completing complex tasks with a group of simple robots, using a few sensors. However, there exists a limit on which information can be retrieved by a swarm of robots, because of the few sensors that each robot uses. A way to increase the information for each robot, and consequently for the whole swarm, is to add a vision module.

Vision modules are currently quite expensive and this cost is multiplied for a swarm of robots. This master thesis aims at designing a 360-degree vision module at a reasonable price for a swarm of *Sphero RVR* robots, and to use this vision module to identify the other robots in the swarm. The vision module is composed of a microprocessor and different cameras.

Three different methods are considered to allow the robot recognition. First, a colour segmentation uses the LEDs of the *Sphero RVR* robots to estimate their position. Secondly, *ArUco* markers placed around the robot allow to localize the other robots. Lastly, a neural network trained on a dataset created for *Sphero RVR* robot recognition in the context of swarm robotics is used to identify the rest of the swarm. These three methods are evaluated through an experimental dataset. Experiments are also conducted to assess the performance of these methods in a real-time application.

The results showed that real-time operations involving computer vision requires a lot of resources. The library *ArUco* is hard to use when the robots are moving too fast, because of motion blur. The method using a neural network is able to detect the moving robots, but takes more time, which currently makes it hard to use in a real-time application.

Keywords: vision module, swarm robotics, robot recognition

Contents

Chapter 1

Introduction

Swarm robotics is currently a very promising research field that makes use of a group of simple robots to complete tasks. Robots used in these swarms can now implement more sensors, in order to gather more information. However, cameras are expensive and the use of such sensors in this field is currently limited.

Vision modules are considered in order to add a new stream of information to each individual of the swarm, and consequently, improving the information of the whole swarm. A vision module can be considered as a sensor that allows machine vision and image processing. It is composed of cameras and a microprocessor. This is currently a rare case in swarm robotics. Indeed, such vision modules are quite expensive, and this cost is of course multiplied in a swarm of robots.

This master thesis aims to design a 360-degree vision module for a swarm of *Sphero RVR* robots for a reasonable price, and to implement a *Sphero RVR* robot recognition algorithm. Indeed, this is a challenge in swarm robotics since vision modules are expensive. The *Sphero RVR* robot recognition algorithm is implemented using the *Robot Operating System* (ROS)[41]. Three different methods are considered for this application, and compared through experimental datasets, and with experiments requiring real-time operations in the context of swarm robotics.

This chapter is structured as follows: Section 1.1 introduces the objectives of the thesis; Section 1.2 identifies the main contributions of the thesis; Section 1.3 explains the structure of this thesis.

1.1 Objectives of the thesis

This master thesis aims to both provide a 360-degree vision module for a swarm of *Sphero RVR* robots for a reasonable price, and implement *Sphero RVR* robot recognition algorithms.

First, a 360-degree vision module for a swarm of *Sphero RVR* robots with low budget is designed and provided. A 360-degree vision module means that the vision module needs to have a 360-degree view. Multiple cameras are used for designing this vision module. It also requires a microprocessor. Indeed, it must both compute image processing and to communicate with the controller of the robot. A controller refers to the behaviour of the

robot, which means that it translates the code and inputs from the sensors to actions for the robot in order to operate and complete its task. Designing this vision module is a challenge since most 360-degree vision systems, which do not include a microprocessor, are very expensive because of the cameras.

Secondly, the implementation of *Sphero RVR* robot recognition algorithms is considered. Three different methods are evaluated and compared to complete the task of recognizing and localizing the other robots of the swarm. The first method applies colour segmentation thanks to the LEDs of the *Sphero RVR* robots. The second one uses *ArUco*[12, 33] markers placed on every robot of the swarm. *ArUco* is a library that allows localizing in a 3D space a marker appearing on an image. The last one makes use of a neural network to recognize the robots and to estimate their position. This neural network is trained on a dataset created in this master thesis for *Sphero RVR* robot recognition in the context of swarm robotics. These three methods are compared on experimental datasets. Experiments are also conducted to assess the performance of the *Sphero RVR* robot recognition algorithms in the context of swarm robotics. This implies real-time operations. These experiments aim at evaluating the use of this vision module for robot recognition in the context of swarm robotics applications.

1.2 Main contributions of the thesis

This thesis aims at contributing to swarm robotics research.

First, on a hardware point of view, I conducted research to design a vision module for the *Sphero RVR* robot in the context of swarm robotics. This vision module aims to be reasonably priced so that it can be used for a swarm of robots. Moreover, this vision module has a 360-degree view, which allows getting images in every direction for the robot.

Secondly, I created a dataset creation protocol for robot recognition, which allows to create a dataset for a potential supervised learning. In this master thesis, it is used in order to create a dataset of images to recognize the *Sphero RVR* robots in the context of swarm robotics. However, this protocol can be applied to other robots and to other applications in robotics.

Lastly, I compared and evaluated three different methods for *Sphero RVR* robot recognition through experimental datasets and experiments in the context of swarm robotics. The first one uses colour segmentation, which is a very used algorithm in computer vision. It allows to estimate the position of the robots containing the reference colour. The second one makes use of the *ArUco*[12, 33] library, which works with markers that can be localized in a 3D space based on an image containing this *ArUco* marker. This consequently gives the opportunity to estimate the precise position of the robot wearing these markers. The last method uses a neural network trained on the dataset generated and mentioned before in order to detect and localize the *Sphero RVR* robots without any additional marker.

1.3 Structure of the thesis

The thesis is structured as follows:

- The current section, Section 1, introduces the thesis, by exposing the objectives and the contributions of this master thesis
- Section 2 states the current state-of-the-art and the related works in relation to this thesis.
- Section 3 describes the *Sphero RVR* robot.
- Section 4 addresses the 360-degree vision module designed in this thesis.
- Section 5 describes the *Sphero RVR* robot recognition algorithms.
- Section 6 compares and evaluates the different methods for the *Sphero RVR* robot recognition on experimental datasets.
- Section 7 presents real-time experiments to assess the application of the different methods used for the *Sphero RVR* robot recognition.
- Section 8 exposes the future improvements that can be considered thanks to this work.
- Section 9 concludes the thesis.

Chapter 2

Related work

Swarm intelligence is attracting more and more interest. Indeed, it allows a group of simple agents to complete complex tasks. This field of artificial intelligence can be applied with physical agents thanks to swarm robotics. Indeed, swarm robotics gives the opportunity to a group of robots to work together to achieve complex tasks.

However, swarm robots are very simple and use only a few simple sensors. Cameras and vision modules are still new in swarm robotics, because of the budget it requires. That is why the state-of-the-art is currently limited in terms of computer vision in swarm robotics.

This chapter is structured as follows: Section 2.1 describes swarm intelligence and its main principles; Section 2.2 defines swarm robotics and the corresponding main properties; Section 2.3 exposes the different platforms used in swarm robotics; Section 2.4 describes ROS and its application in swarm robotics; Section 2.5 exposes the current 360-degree cameras that currently exist on the market; Section 2.6 presents the current state of computer vision in robotics; Section 2.7 describes the current state-of-the-art of computer vision in swarm robotics.

2.1 Swarm intelligence

Swarm intelligence is part of artificial intelligence. It is the discipline that uses systems composed of many agents that coordinate together using decentralized control and self-organization[8]. It focuses on the behaviour resulting from local interactions between the agents of the system or the environment. Multiple algorithms exist and are designed to use swarm intelligence to solve complex tasks or missions. For example, the Ant Colony Optimization (ACO)[9] is able to solve complex optimization problems, such as the Quadratic Assignment Problem (QAP) which is known as NP-hard[42].

A swarm intelligence system typically has some characteristics[8].

- The system is composed of many agents
- The agents are homogeneous, which means that they are all identical or that they belong to a few typologies
- The interactions among the agents are based on simple rules which exploit local information that they can directly exchange, or that they can share thanks to the

environment

- The behaviour of the whole system is the result from the local interactions of agents, which means that the overall behaviour self-organizes

Moreover, the swarm system, which includes the environment and the agents, should be able to behave in a coordinated way without an external coordinator.

Lastly, a swarm intelligence system has multiple particular properties[8].

- Scalability of the system:
  The swarm system are expected to scale with the size of the swarm, which means with the number of agents in the swarm, without requiring to change the rules for the local interactions of the agents. When the number of agents increase, performances should not degrade. Moreover, some systems can increase their efficiency only by increasing the size of the swarm.

- Flexibility:
  This means that multiple agents can act at the same time in different places. Indeed, since every agent is influenced by its local neighbours and the environment, the whole system does not need to be synchronized. In fact, since it is a decentralized system, every agent can act and can influence the behaviour of the swarm at each moment.

- Fault tolerant system:
  Swarm systems are fault tolerant. Indeed, because of the size of the swarm, and because of the decentralized and self-organized system, a fault of an individual is diluted by the rest of the swarm.

2.2 Swarm robotics

Swarm robotics is the application of swarm intelligence with robots as agents. This research field aims at designing the robots and studying their corresponding behaviour[37]. This is nevertheless an emerging research field, since the potential for the industry applications are multiple. However, it is currently hard to rely on such systems with real robots since the results between simulated systems are not accurate enough for real applications. Moreover, it is currently hard to predict the general behaviour of the swarm only knowing the individual ones. Indeed, by definition of a swarm, it self-organizes and the system is decentralized, which means that no one is able to control the behaviour of the whole swarm. However, swarm robotics is a relatively new research field and is already proposing solutions from theoretical concepts to real prototypes.

The properties of swarm robotics systems are considered as a consensus from multiple authors[4, 37, 45].

- Scalability:
  The swarm robotics system should be able to scale with the size of the swarm, which means that the swarm should be able to operate with different sizes of group sizes.

- Fault tolerance:
  The system needs to be robust, which means that it should be able to continue operating even if one of the agents is faulty, malfunctioning or lost.

- Local interactions:
  Robots can only have local interactions, and they can not have access to a centralized controller. The interactions can be done through sensors and communication protocols.

- Global behaviour:
  The behaviour of the swarm is self-organized by the robots in the swarm.

- Capabilities:
  Swarm robotics systems should be able to achieve different tasks thanks to cooperation among the robots.

2.3 Platforms in swarm robotics

In swarm robotics, there exists multiple platforms. The following analysis is mainly based on the work of Nadia and Luneque[28]. The platforms can be divided into physical and simulated robots.

2.3.1 Physical robots

First of all, physical robots are of course one of the main types of platforms used in this field. Indeed, the robots need to be cheap, not very computationally powerful and are most of the time quite small, in order to design experiments with a large group of robots.

One of the most known robots in swarm robotics is the *e-puck*[26]. The robot is shown on Figure 2.1. One of the main advantages of the *e-puck* is that it is very simple to use. Moreover, it is already implementing a CPU and sensors. It also allows infrared communication. Moreover, there exists some extensions for the *e-puck*, allowing to have more features on the robot, such as the *omnidirectional vision turret*. Also, it is affordable on a budget aspect, which makes it one of the best candidates for research in swarm robotics.

Figure 2.1: An *e-puck* robot

There also exists the *Kilobot*[35], which is a very small and low-cost robot. It is aimed to be used for experiments with a large group of agents. It moves thanks to vibrations, and is not moving using wheels. This explains the low cost of this robot. It can be seen on Figure 2.2.

Figure 2.2: A *Kilobot* robot

Of course, there exists a lot of robots for swarm robotics, such as the *S-bot*[10], which can aggregate and connect with other similar robots, or the *marXbot*[3].

2.3.2 Simulators

Since conducting experiments with physical robots can be expensive in terms of time or budget, simulators are very popular in swarm robotics[37].

Multiple platforms for simulations of swarm robots currently exist. These can be 2-dimensional or 3-dimensional, and they are able to represent robots.

For example, *Gazebo*[14] is a very well-known free 3-dimensional simulator for a population of mobile robots.

There also exists *ARGoS*[30], which is a multi-physics robot simulator. This can be used to simulate large-scale swarms of robots efficiently. It can be upgraded through multiple plugins to fit different needs.

2.4 ROS

For implementation of controllers for robots, and more specifically swarm robots, controllers with *ARGoS* are developped. However, there exist some frameworks now in order to ease the implementation for robots. The Robot Operating System (ROS) is a framework that helps to build robot applications.

ROS[41] allows to have an architecture organized in nodes and topics. Topics are common places in the computer, where nodes, that are the current programs executed on the robot, can either publish or subscribe. This means that any node can send data through a given topic, by publishing messages, or receive data from it, by subscribing to the topic. This allows data exchange between the nodes.

Multiple works involving ROS exist in robotics. For example, Karimi et al.[20] and Araújo et al.[2] make use of ROS as their framework for a mobile robot platform for research and

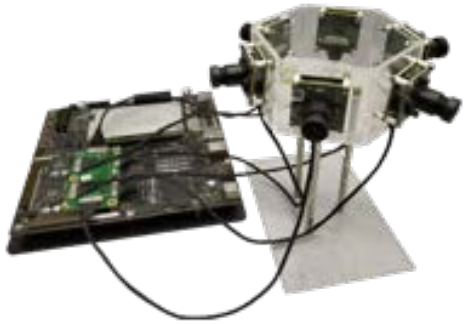


Figure 2.3: *HexCamera*

Figure 2.4: *Ladybug5+*

education.

There also exist some works using ROS in swarm robotics. Indeed, Kegeleirs, Garzón Ramos, and Birattari[21] use ROS as the framework to implement the controller for a swarm of *e-pucks* mapping its environment. That is also the case for Ackerman et al.[1] with their work that aims at producing prototypes for robot swarms that would collect resources on the surface of Mars. This is done in the context of a swarm robotics programming challenge where the swarm of robots needs to search for, pick up and drop off resources in a collection zone.

2.5 360-degree cameras

360-degree cameras are more and more popular on the market.

For example, the *HexCamera*[1] is a module containing 6 cameras with a DFOV of 118°. It can be organized such as the 6 cameras are looking in a different direction, to get a 360° view, as shown by Figure 2.3. This device is currently on sale for $1499 on this website.

Teledyne FLIR also proposes some vision modules for a more professional use. For example, the *Ladybug5+*[2], shown on Figure 2.4, is an omnidirectional camera composed of 6 cameras placed with 5 cameras on the same plane, looking in different directions, and a last camera looking up. This device can reach a 8k resolution while keeping a frame rate of 30FPS. The price is not indicated on the website itself because it requires to request a quote, but it should cost around $20,000[3].

[1]HexCamera by e-con Systems. URL: `https://www.e-consystems.com/multiple-csi-cameras-for-nvidia-jetson-tx2.asp`

[2]Ladybug5+ by Teledyne FLIR. URL: `https://www.flir.com/products/ladybug5plus/`

[3]see `https://360rumors.com/flir-ladybug5-8k-360-camera-has-six-23-inch-sony-sensors/#:~:text=Ladybug5%2B%20is%20now%20shipping.,got%20a%20quote%20of%20%2419%2C995.`

A cheaper option is the *GoPro MAX*[4], currently on sale for $399 on the official website. It proposes a 360° vision module, using 2 *fisheye* cameras covering a whole view by projecting the images on a sphere. This kind of cameras is very popular nowadays because it allows to have a 360-degree view camera at a reasonable price for personal use. However, this implies distortion and it requires very high quality cameras to still get a high resolution on every part of the image, and even with that, there exists some blurry parts on the image.

2.6 Computer vision in robotics with 360-degree vision module

Currently, applications of computer vision using 360-degree vision module in robotics are numerous.
Kim, Jung, and Park have made a robot able to map an environment with a 360-degree view[22]. It uses 4 *fisheye* cameras and then blend the images in order to project them on a cube around the robot. A robot using 2 *fisheye* cameras is described by Sadowski[36]. It also allows to have a 360-degree view of the scene.

There also exists some vision modules using other sensors than just cameras.

For example, Silva, Roche, and Kondoz have made a robot using a LiDAR and a *fisheye* camera to get redundancy and hence, to have more robust data[38].

Ding, Liu, and Li make use of RGB-D cameras to get a very wide panorama of images with depth information in it[7]. This kind of cameras is using IR light to estimate the depth.

Sumigray et al. have created a way to create a 6DoF environment visualization for a teleoperated robot[43]. It uses 6 *fisheye* cameras and 2 RGB-D cameras to create this VR environment. This configuration produces very good results.

2.7 Computer vision in swarm robotics with 360-degree vision module

The state-of-the-art in computer vision using 360-degree vision module in swarm robotics is limited. However, there still exists some works about a vision module for computer vision in swarm robotics.

Indeed, there exists some papers about vision modules implemented on swarm robots. However, for most of them, the vision module basically relies on 1 camera looking in a given direction[5, 15, 51, 23]. Papers about a 360-degree vision module also exist. For example, Garzón Ramos and Birattari have designed a swarm of robots able to detect colours and to behave according to the colour perceived and emitted from other robots[13]. It uses the omnidirectional vision turret of the *e-puck* robot, which is basically a *fisheye*

[4]GoPro MAX. URL: `https://gopro.com/en/us/shop/cameras/max/CHDHZ-202-master.html`

camera. One of the main problem of their approach is that the image from the camera is completely distorted. This only allows to recognize colours, instead of more complex features.

Concerning vision modules containing more than one camera, there exists a paper about an 360-degree vision module for aerial robot swarms[16]. Recently, a swarm of robots including this module has been designed[18]. It contains 4 cameras to cover a 360-degree view, and an additional one to cover the area above.

Chapter 3

The *Sphero RVR* robot

The *Sphero RVR* robot[1] is an educational and programmable robot sold by *Sphero*. It is a relatively new robot, introduced in early 2019. The robot includes multiple sensors and actuators. The *Sphero RVR* robot can also implement extra modules and sensors.

Currently, there is no known application of swarm robotics using the *Sphero RVR* robot as a platform.

This chapter is structured as follows: Section 3.1 describes the hardware architecture of the *Sphero RVR* robot; Section 3.2 details the software architecture of the *Sphero RVR* robot; Section 3.3 analyses the robot as a platform in swarm robotics;

3.1 Hardware architecture

The *Sphero RVR* is an educational and programmable robot. It can implement code and instructions using the *Sphero Edu* application. The *Sphero RVR* robot can be seen on Figure 3.1.

The *Sphero RVR* is a robot already implementing a set of sensors and actuators. It includes[32] :

- Sensors:
  - Ambient light sensor
  - RGB sensor with normalizing LED and focus lens
  - Full 9-axis IMU - accelerometer, gyro, magnetometer
  - Infrared sensor
  - High-resolution 20-pole magnetic encoders
- Actuators:
  - 10 individually addressable RGB LEDs
  - 2 high-power compact motor using high-precision involute gears with vibration and noise dampening

[1]Sphero RVR. URL: `https://sphero.com/products/rvr`

Figure 3.1: A *Sphero RVR* robot

It can also be noted that the wheels are covered with an hypergrip all-terrain treads, allowing a better grip on the floor.

The *Sphero RVR* robot also includes a 4-Pin UART port (RX, TX, GND, 5V, with 3.3V signals). This means that in addition with the actuators and sensors that the *Sphero RVR* already includes, external boards can be attached, including *micro:bit*[2], *Arduino*[3] and *Raspberry Pi*[4] boards.

3.2 Software architecture

The controller of the *Sphero RVR* robot can be accessed through the *Sphero Edu* application. It allows to either program the robot, or remotely control it thanks to an external device using a *Bluetooth* connection.

The *Sphero RVR* robot also implements the *Sphero Public SDK*[40]. It allows, thanks to an API available in *Python* and *Javascript*, to get access to the sensors and actuators from a software level. Using an external board, it is hence possible to control the robot, by interacting with the API responsible for the sensors and actuators of the *Sphero RVR* robot.

[2] *micro:bit*. URL: `https://microbit.org/`
[3] *Arduino*. URL: `https://www.arduino.cc/`
[4] *Raspberry Pi*. URL: `https://www.raspberrypi.org/`

3.3 System analysis

The *Sphero RVR* robot is a relatively new robot. It already includes very common sensors and actuators. Moreover, its design and motors allow the robot to accurately move in different environments. Lastly, the 4-Pin UART port gives the possibility to use an external board. Thanks to the *Sphero Public SDK*, it gives the possibility to adapt the controller of the robot to the wanted application. However, even if the *Sphero RVR* robot presents a lot of advantages, it has never been used in the context of swarm robotics.
The robot is relatively new. Since this robot has never been used before in swarm robotics, there is less support and documentation about this robot, and its efficiency in this field of research has not been proven yet compared to other platforms such as the *e-puck* robot or the *Kilobot*.

Also, the *Sphero RVR* robot has a length of 18cm, a width of 22cm and a height of 11cm. Compared to other robots used in swarm robotics, it is relatively voluminous. Indeed, for example, the *e-puck* robot has a diameter of 7cm with a height of 5.5cm. For comparison, the surface taken by the *Sphero RVR* robot is more than 10 times more than the area of the *e-puck* robot. This could be a limitation in the context of swarm robotics. Indeed, usual experiments require of course a swarm of robots. This means that multiple robots need to be involved. These robots are required to act and behave in a given environment. However, the space required by an experiment with a swarm of 10 *e-puck* robots is much lower compared to the area required by the same experiment with a swarm of 10 *Sphero RVR* robots. This is of course a strong limitation for researchers that want to work in swarm robotics.

However, even with these downsides, the *Sphero RVR* robot also have very good points. Indeed, the motors are very powerful and accurate. There is also the presence of high-resolution 20-pole magnetic encoders, which allow precise measurements concerning the movement of the robot. Moreover, the *Sphero RVR* already contains multiple sensors, listed before. These type of sensors allows usual experiments in swarm robotics.

That is why the *Sphero RVR* robot is the platform considered in this master thesis.

Chapter 4

360-degree vision module

A vision module is composed of a microprocessor and cameras. In order to allow the robot to get images thanks to the cameras, and to extract features such as the robot recognition, one needs to choose the correct components and the correct design for the vision module. Moreover, in order to have a 360-degree vision module, multiple cameras need to be selected.

The setup of the microprocessor is described in Appendix C, and the communication between the 360-degree vision module and the microprocessor considered as the controller of the robot is detailed in Appendix D.

This chapter is structured as follows: Section 4.1 describes the microprocessor chosen for this vision module; Section 4.2 details the different cameras considered for the conception of the vision module; Section 4.3 exposes the design of the vision module for the *Sphero RVR* robot; Section 4.4 describes the vision module prototype built in this thesis.

4.1 Microprocessor

As explained in the dedicated section, the *Sphero RVR* robot can work with external boards, such as the *micro:bit*, *Arduino* and *Raspberry Pi* boards. Since these are the recommended boards that can be used with the *Sphero RVR*[1], the discussion is focused on these 3 types of boards.

micro:bit boards aim at giving easy programming and coding for educational purposes. This kind of boards are quite simple and gives less freedom for complex purposes. For instance, it can not install a full operating system, which is annoying in the context of this thesis, where ROS, a very useful framework for swarm robotics, could be used.

Arduino boards uses GPIO pins to control sensors and actuators. This already have more computational power than *micro:bit* boards. However, it still can not install an operating system.

Raspberry Pi boards are very attractive since it allows to have a much higher computational power, while allowing to fully install an operating system. This means that ROS

[1]Sphero RVR FAQ. URL: `https://sdk.sphero.com/docs/faqs/rvr_faq`

can be used on *Raspberry Pi* boards, which is a huge advantage. However, in terms of budget, *Raspberry Pi* boards are more expensive than *Arduino* and *micro:bit* ones. Lastly, since the vision module aims at extracting features using stream of images, a high computational power is required.

Regarding the current available options, a *Raspberry Pi* board is chosen in the context of this master thesis. Indeed, it allows to use ROS, but also to have the highest computational power between the three options. The *Raspberry Pi Model B*[2] is selected for this vision module.

4.2 Cameras

Before reporting the cameras, it is important to say that these should fit the *Raspberry Pi 4 Model B* previously selected. Indeed, the cameras will be directly connected to the microprocessor. However, there are some hardware restrictions. In the datasheet of the *Raspberry Pi*, ports are limited.

- 1 Camera Serial Interface (CSI) port
- 4 USB ports

Indeed, the unique CSI port, which is a very reliable and fast way of transmitting images data, implies that there would be at most 1 camera using this port. The 4 USB ports are less vital than the uniqueness of the CSI port but it also gives the information that at most 4 devices can be connected through this port. However, it is still possible to use a USB hub in order to get more devices, but it will decrease the bandwidth and hence, the rate at which data is transmitted.

Hence, the cameras reported are separated between cameras having a CSI port, and those which supports USB connection.

4.2.1 CSI cameras

The CSI cameras are reported in Table 4.1.

[2]Raspberry Pi 4 Model B specifications. URL: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/`

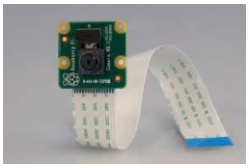
Name	Raspberry Pi Camera Module (G)[3]	OV5647 Camera Board w/ CS Mount for Raspberry Pi (B0032)[4]	Raspberry Pi Camera Module v2[5]
Picture			
Frame rates		QSXGA (2952x1944)@15fps 1080p@30fps 960p@45fps 720p@60fps VGA (640x480)@90fps QVGA (320x240)@120fps	1080@30fps 720@60fps
Max resolution	2592x1944	2592x1944	3280x2464
Focal length	Adjustable		3.04mm
Field of view	160°D (120°H)		62.2°H x 48.8°V
Dimensions	25mm x 24mm x 20mm (mesured)	36mm x 36mm x 38mm (measured)	23.86mm x 25mm x 9mm
Availability	Currently available (RobotShop)	Currently available (RobotShop)	Currently available (Mouser)
Price	27€00	27€84	28€33

Table 4.1: CSI cameras specifications

Multiple points need to be taken into account when comparing the cameras in the context of a vision module for swarm robotics.

In this case, the price is of course a very important parameter, since a difference for one camera is multiplied for the whole robot and even more for the whole swarm. In this analysis, the differences in price are very low, which means that it will not be a critical factor for the choice of the camera.

The second parameter that need to be taken into account is the field of view. Indeed, the wider the view of the camera, the more it can see. However, this also means, for similar resolution, a lower quality of image for a given field of view. In this table, it can be seen that the *Raspberry Pi Camera Module (G)* has a field of view of 160°. In fact, it is considered as a *fisheye* camera, which simply means it aims a shooting images with a very large field of view.

[3]Raspberry Pi Camera Module (G) specifications. URL: `https://www.robotshop.com/eu/en/raspberry-pi-camera-module-g-fisheye-lens.html`

[4]OV5647 Camera Board w/ CS Mount for Raspberry Pi (B0032) specifications. URL: `https://www.arducam.com/downloads/modules/RaspberryPi_camera/OV5647DS.pdf`

[5]Raspberry Pi Camera Module v2 specifications. URL: `https://www.robotshop.com/media/files/pdf/70812916-datasheet.pdf`

The frame rates and the resolutions at which the camera can capture images is also a very important parameter. Indeed, the best is to have high resolutions at high frame rates. However, it can be seen that a trade-off between the two is inevitable. The table shows that the cameras can work at various frame rates and resolutions, and that the *OV5647 Camera Board w/ CS Mount for Raspberry Pi (B0032)* has a large panel of configurations.

The focal length is also an important factor. Indeed, in order to have sharp images, the focal length must correspond with the distance at which the relevant elements will appear. However, it is hard to know in advance this distance. That is why an adjustable focal length can be very interesting, such as what is proposed by the *Raspberry Pi Camera Module (G)*.

The maximum resolution can also be an interesting parameter. Indeed, even if the frame rate is very low to reach such resolution, it can be interesting in case of capturing images with a very high resolution occasionally. All the cameras can reach a very high resolution, and the *Raspberry Pi Camera Module v2* can even reach 8MP.

The dimensions can also be considered, since the cameras need to be placed on the *Sphero RVR* robot. However, while it does not take an abnormal space, this is not a critical parameter.

Lastly, the availability of the camera is of course important. If the camera is not available, it is not possible to use it in the vision module. The three cameras are available in online shops.

4.2.2 USB cameras

The USB cameras are reported in Table 4.2.

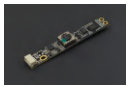
Name	DFRobot FIT0701[6]	M5Stack U082-X[7]	DFRobot SEN0286[8]	DFRobot DFR0620[9]	Waveshare IMX335 5MP USB Plug-and-Play Camera[10]
Picture					
Frame rates					2592x1944@20fps 1080p@30fps 720p@30fps
Max resolution	640x480	2048x1536	2592x1944	1600x1200	2592x1944
Focal length		Adjustable	Auto-focus		3.91mm
Field of view		66.5°D	72°D		106°D 92.6°H 48.6°V
Dimensions	30mm x 25mm x 21.4mm	48mm x 24mm x 15mm	65mm x 9mm x 5mm	41mm x 21mm x 6.5mm	38mm x 38mm x 36.98mm
Availability	Currently available (Mouser)	Currently available (Mouser)	Soon available (31/12/2021 on Mouser)	Currently available (Mouser)	Soon available (23/02/2022 on RobotShop)
Price	7€44	15€71	21€88	27€87	43€21

Table 4.2: USB cameras specifications

The same analysis as for CSI cameras is done for the USB ones.

The prices are much more uneven. Indeed, the range of price goes from less than 10€ to more than 40€. For this master thesis, 40€ is quite expensive for the vision module, which is already a bad point for the *Waveshare IMX335 5MP USB Plug-and-Play Camera*. Keeping the budget as low as possible while having interesting capabilities is of course the goal when selecting components.

Concerning the field of view, there are no *fisheye* cameras, even if the *Waveshare IMX335 5MP USB Plug-and-Play Camera* has a wider field of view than the rest of the cameras. It is however not a critical factor.

The frame rates and resolutions are not mentioned in most of the datasheet of the cameras. This will hence not be taken into account and be determining for the selection of cameras.

Concerning the focal length, the *M5Stack U082-X* has an adjustable focal length, and

[6]DFRobot FIT0701 specifications. URL: `https://www.dfrobot.com/product-2089.html`

[7]M5Stack U082-X specifications. URL: `https://docs.m5stack.com/en/unit/timercam_x`

[8]DFRobot SEN0286 specifications. URL: `https://www.dfrobot.com/product-1956.html`

[9]DFRobot DFR0620 specifications. URL: `https://www.dfrobot.com/product-1931.html`

[10]Waveshare IMX335 5MP USB Plug-and-Play Camera specifications. URL: `https://www.waveshare.com/imx335-5mp-usb-camera-a.htm`

the *DFRobot SEN0286* can auto-focus. Auto-focusing means that the focal length is adjustable, and can automatically adjust depending on the image captured by the camera.

The maximum resolution is also a parameter assessing the quality of images that the camera can capture. The *DFRobot FIT0701* can only take images with a resolution of 640x480 at most, which is relatively low. The rest of the cameras have relatively high resolution.

In terms of size and dimensions, it can be observed that the *DFRobot SEN0286* is very small. Indeed, it is quite long, with a length of 65mm, but the width and height make it very portable and relatively not cumbersome.

All the cameras are currently available, except for the *DFRobot SEN0286* and the *Waveshare IMX335 5MP USB Plug-and-Play Camera*. The *DFRobot SEN0286* can still be available quite soon, while the *Waveshare IMX335 5MP USB Plug-and-Play Camera* requires 2 more months.

As a last remark, the *M5Stack U082-X* is using a software layer on top of the module, meaning that it implies a dependency to this software if this camera is used.

4.3 Design

In order to design the vision module, a 3D modelling software is used. In this thesis, *Blender*[6] will generate the 3D models. Indeed, it is an open and free 3D modelling software. Moreover, it gives the opportunity to create cameras inside of it, and to see through them. The *Sphero RVR* robot has been modelled and the cameras are rendered as boxes with a *Blender* camera in it.

For the design itself, there is a main constraint for the module. Indeed, the final robot will also have a *LiDAR* sensor, rotating at the top of the car. Hence, it is not possible to place a module with all the cameras very close to the centre of the robot, as a compact vision module. The idea is to split the cameras over the *Sphero RVR* robot, in order to have a working vision module, compatible with the use of a *LiDAR* sensor. However, this implies a bigger offset from the center, which automatically leads to blind spots at closer range.

Also, compared to what Sadowski[36] did, which was using a vision module with 2 *fisheye* cameras back-to-back to cover the 360-degree view, the quality of the images need to be high enough in order to extract interesting features such as the robot recognition. Consequently, the design proposed by Sadowski is not selected. Indeed, even if the *fisheye* cameras allow to have a much wider field of view, it decreases the quality of the image because of the distortion and the very wide angle of the camera for the same resolution as other cameras.

Also, the region of interest for this robot will be on the front of it. Indeed, most missions in swarm robotics involving vision consider that the region of interest is in front of the robot[15]. Consequently, this part needs a particular attention and need to have very high

quality images.

In the context of this master thesis, the price of the selected cameras needs to be as low as possible, while allowing extracting interesting features such as the robot recognition in a 360-degree view. Three selections of cameras are considered in this thesis.

1. 1 *Raspberry Pi Camera Module (G)* and 3 *DFRobot SEN0286*
   Estimated cost = 92€64
2. 1 *Raspberry Pi Camera Module (G)*, 1 *Waveshare IMX335 5MP USB Plug-and-Play Camera* and 2 *DFRobot SEN0286*
   Estimated cost = 113€97
3. 1 *Raspberry Pi Camera Module (G)* and 2 *Waveshare IMX335 5MP USB Plug-and-Play Camera*
   Estimated cost = 113€42

As it can be seen, the CSI camera *Raspberry Pi Camera Module (G)* is always chosen. Indeed, its large field of view allows getting information for a very wide angle. However, this will also imply more distortion, so the quality of the information will be slightly decreased.

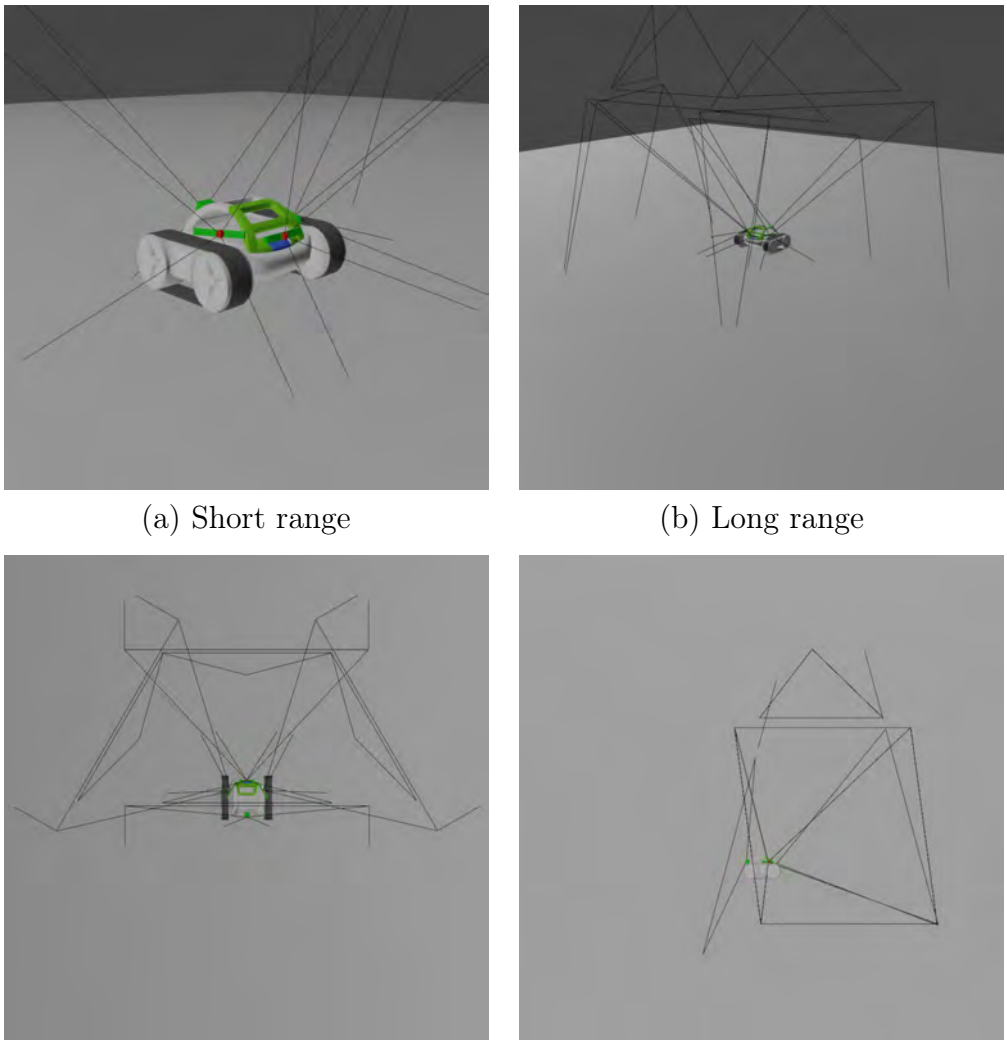
All the considered designs are described in Appendix A. Only the most interesting ones are reported here.

4.3.1 Selection of cameras 1

This selection contains 1 *Raspberry Pi Camera Module (G)* and 3 *DFRobot SEN0286*, and is estimated to cost 92€64.

4.3.1.1 Asymmetric placement of cameras oriented upwards

The design can be seen on Figure 4.1.



(a) Short range (b) Long range

(c) From above (d) From the right

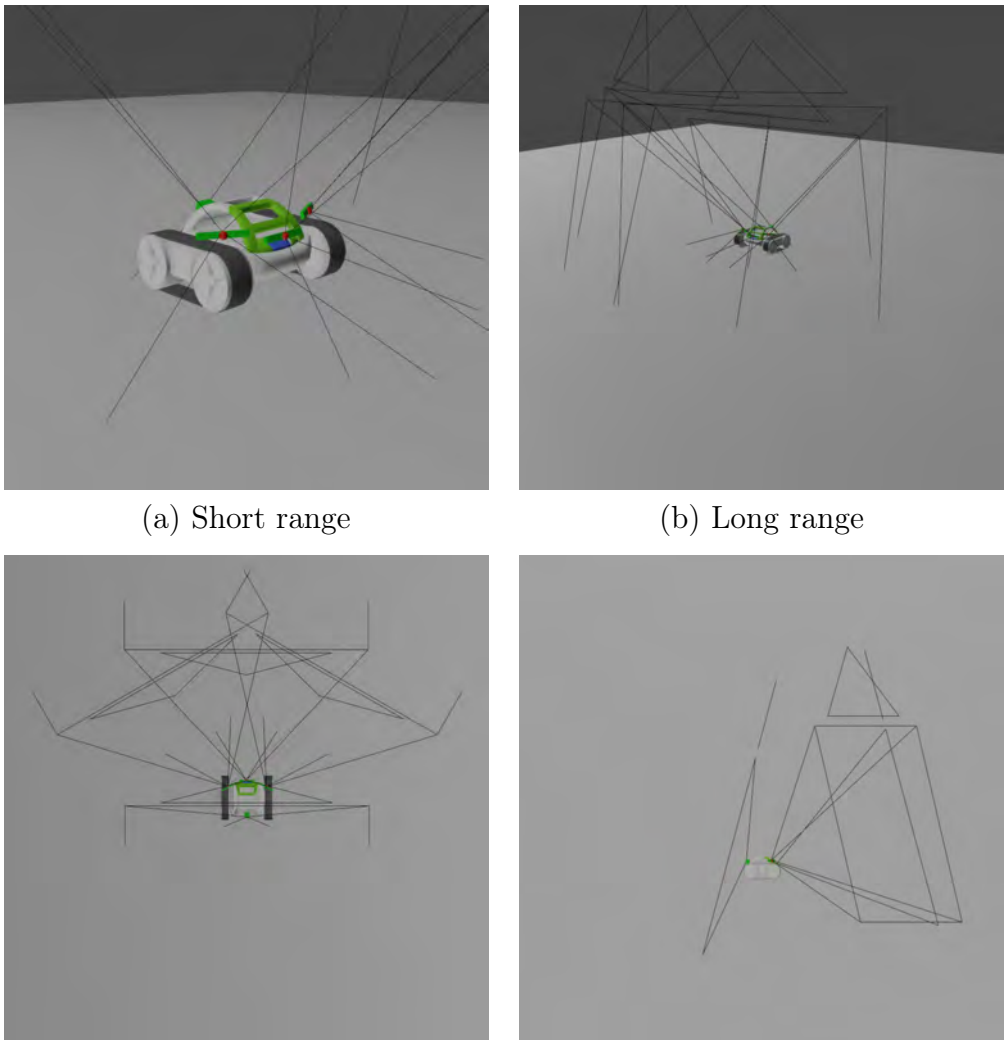
Figure 4.1: Selection of cameras 1 - Asymmetric placement of cameras oriented upwards

Compared to a basic asymmetric design, where the cameras are placed oriented to the front of the *Sphero RVR* robot, the cameras are a little bit oriented to the top. As it could have been seen on Figure A.2-d, the cameras have a part of the image always seeing too close. This design allows seeing more upwards than before, and also gives the opportunity to get features that are not at the ground level.

In the context of this thesis, the robot recognition is the focus in terms of features. However, future interesting features can be extracted thanks to the cameras oriented upwards.

4.3.1.2 Asymmetric placement of cameras with bigger offset from the centre oriented upwards

The design can be seen on Figure 4.2.



(a) Short range (b) Long range

(c) From above (d) From the right

Figure 4.2: Selection of cameras 1 - Asymmetric placement of cameras with bigger offset from the centre oriented upwards

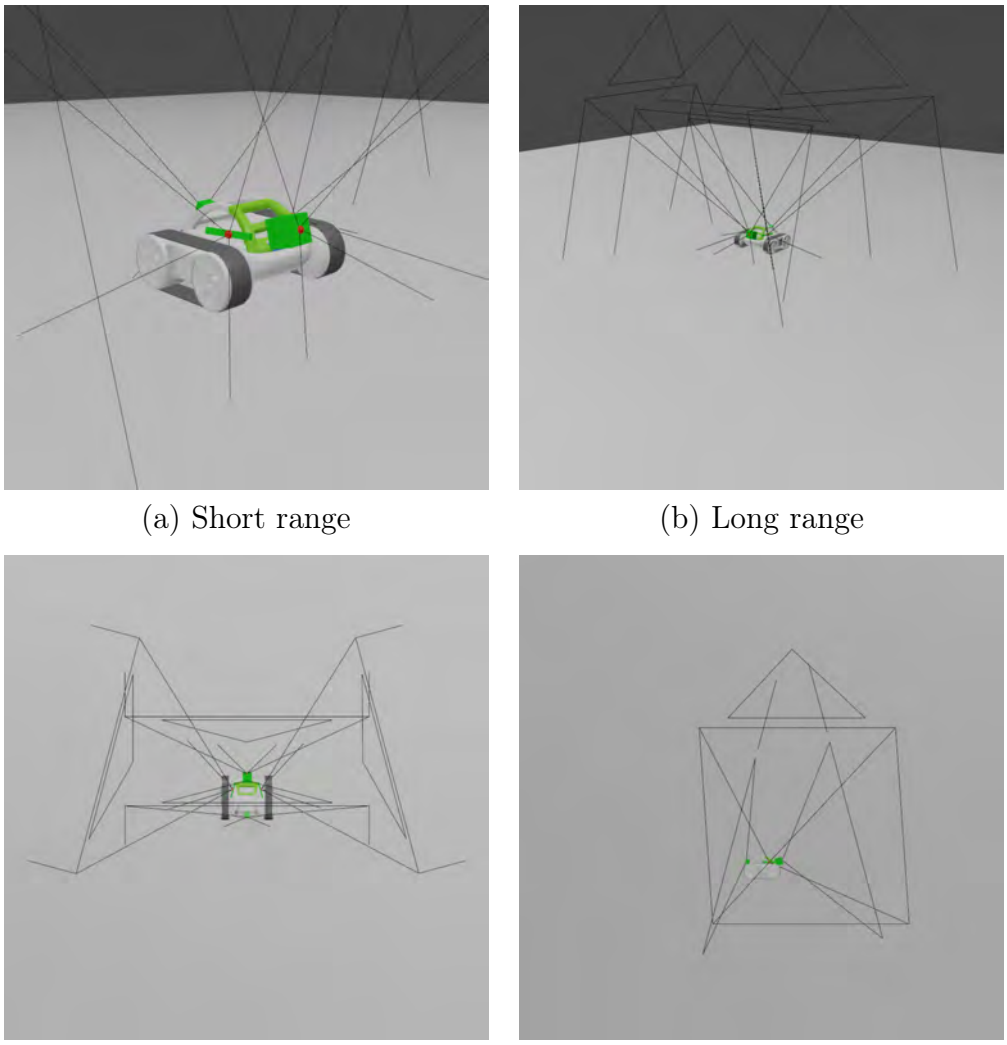
This design tries to take into account the problem stated by the design described in subsubsection 4.3.1.1. However, compared to the design detailed by a similar configuration where the cameras are not oriented upwards, the offset required from the centre is lower to avoid looking directly at the ground.

4.3.2 Selection of cameras 2

This selection contains 1 *Raspberry Pi Camera Module (G)*, 1 *Waveshare IMX335 5MP USB Plug-and-Play Camera* and 2 *DFRobot SEN0286*, and is estimated to cost 113€97.

4.3.2.1 Asymmetric placement of cameras oriented upwards

The design can be seen in Figure 4.3.



(a) Short range (b) Long range

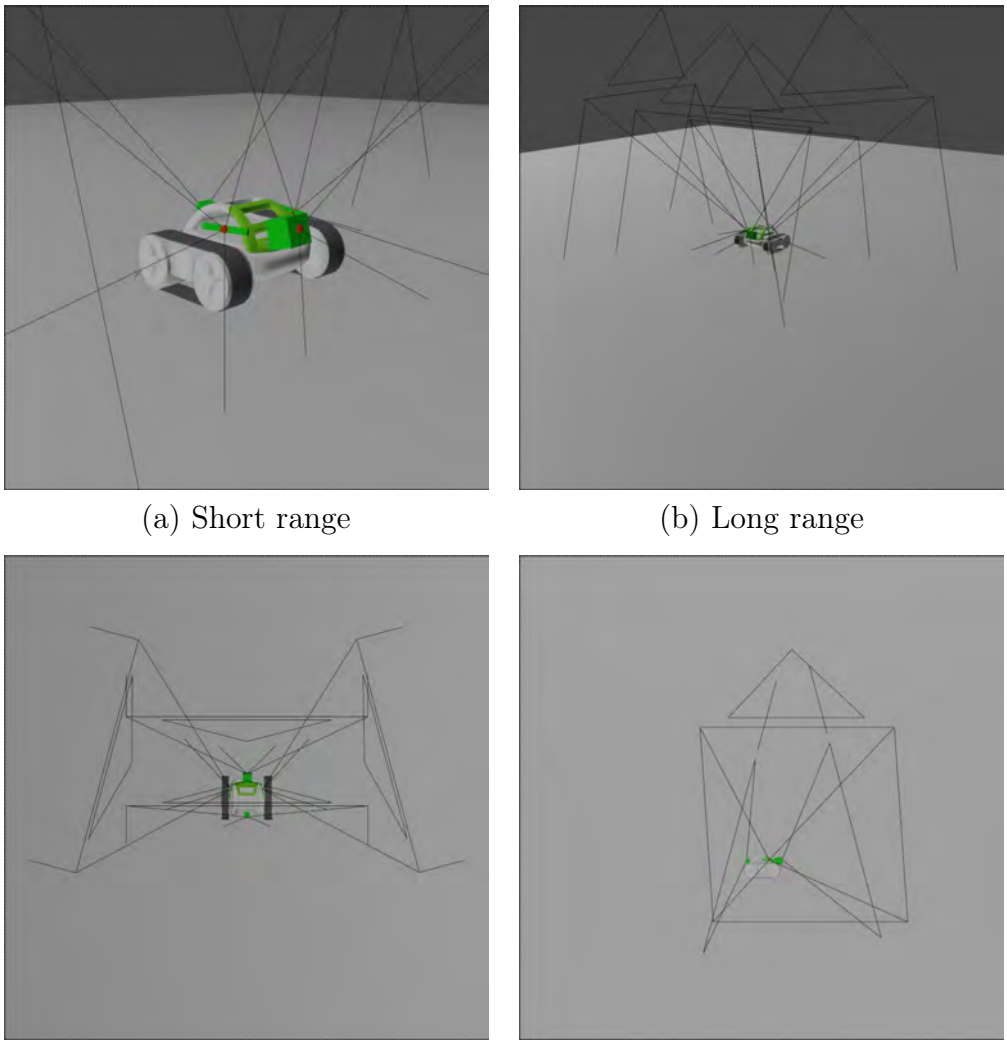
(c) From above (d) From the right

Figure 4.3: Selection of cameras 2 - Asymmetric placement of cameras oriented upwards

This is an asymmetric design, where the cameras are more oriented to the front of the *Sphero RVR* robot, but the cameras are also oriented more upwards. The idea is the same as the design described by subsubsection 4.3.1.1. This way, the cameras can have more useful information.

4.3.2.2 Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 1

The design can be seen in Figure 4.4.



(a) Short range (b) Long range

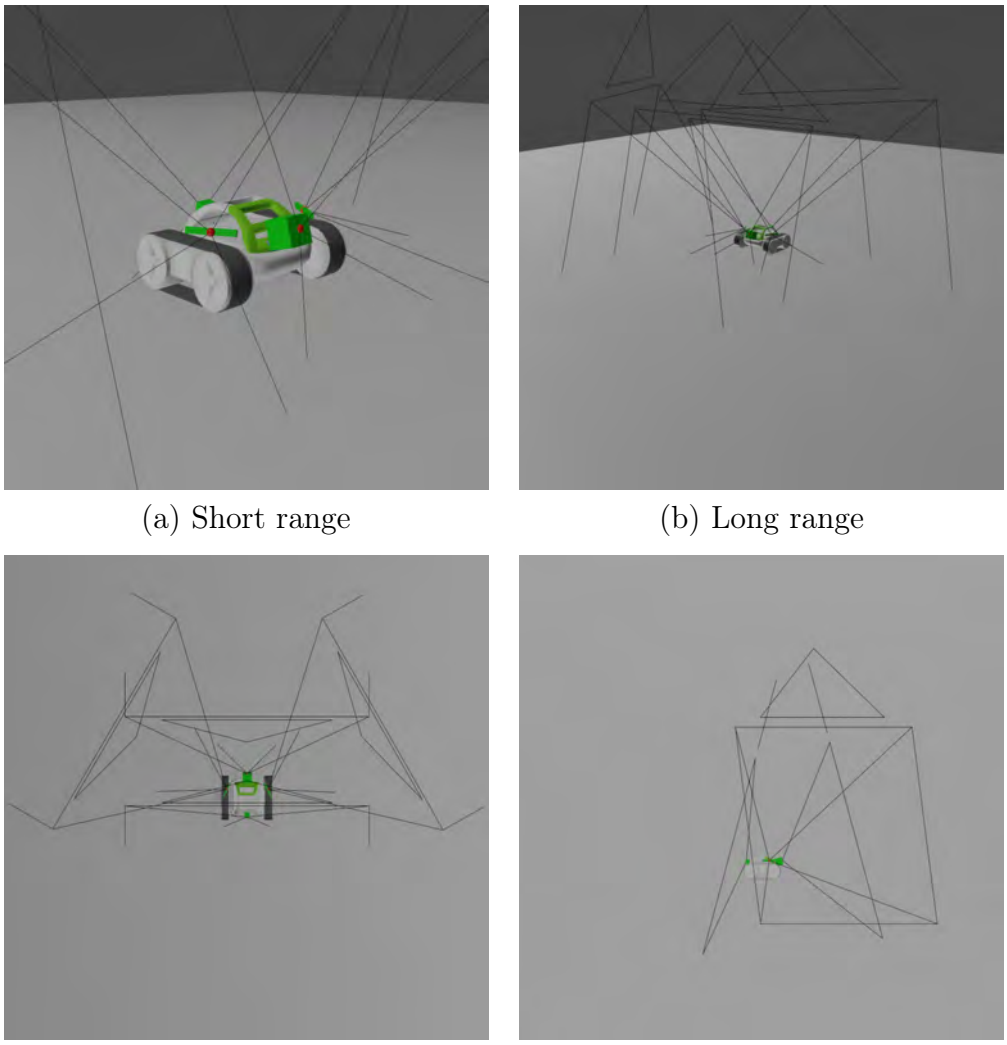
(c) From above (d) From the right

Figure 4.4: Selection of cameras 2 - Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 1

This design follows the same idea as subsubsection 4.3.1.2. This takes into account a bigger offset and the cameras oriented upwards.

4.3.2.3 Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 2

The design can be seen in Figure 4.5.



(a) Short range (b) Long range

(c) From above (d) From the right

Figure 4.5: Configuration 2 : Asymmetric placement with bigger offset oriented upwards - version 2

This design is nearly the same as the previous one described in subsubsection 4.3.2.2, but it includes the idea reducing the dead angles in the front of the car. This way, the vision module is able to have a better vision in the front, while having a wider view than before.

4.3.3 Selection of cameras 3

This design has 1 *Raspberry Pi Camera Module (G)* and 2 *Waveshare IMX335 5MP USB Plug-and-Play Camera*, and costs 113€42.

2 different designs are considered, that can be found in Appendix A. However, this selection of cameras does not lead to any satisfying design.

4.3.4 Design discussion

The designs using the third selection of cameras are risky. Indeed, in the considered designs that can be found in Appendix A, the very front of the car is not covered by a

Figure 4.6: Vision module prototype

camera having as an optical axis the direction of the car.

The second selection of cameras leads to very interesting configurations. However, the camera *Waveshare IMX335 5MP USB Plug-and-Play Camera* is relatively big compared to the other ones. Moreover, this only allows to have a better vision on the back left and right of the car, which are very unlikely to have very interesting features for the robot. In addition, this second selection is the most expensive one.

Hence, the designs using the first selection of cameras are considered. Out of them, the choice has been done on the design described by subsubsection 4.3.1.2 because it allows to have an image full of useful pixels. Indeed, there are much less useless pixels seeing the wheels or the ground directly, while giving the chance to the vision module to get information from higher places.

4.4 Prototype

With the design selected, a vision module prototype was built. At first, this prototype aimed at having the cameras placed exactly at the same spot as described by the design selected in the section 4.3. However, because of technical issues, it was not possible to reach the exact same configuration. In order to still assess the effectiveness of the vision module, a simplified prototype was built. The vision module alone is presented on Figure 4.6, and is shown on the *Sphero RVR* robot on Figure 4.7.

The cameras selected are the same as the ones described in the chosen design, which is detailed in subsubsection 4.3.1.2. Moreover, it follows the same idea of having the cameras focused on the front part of the *Sphero RVR* robot, while having the *fisheye* camera looking at the back.
This prototype will be modified in the future in order to integrate other modules for the *Sphero RVR* robot. Indeed, this prototype does not currently take into account the controller microprocessor for instance. However, it allows conducting experiments to assess the efficiency of the vision module.

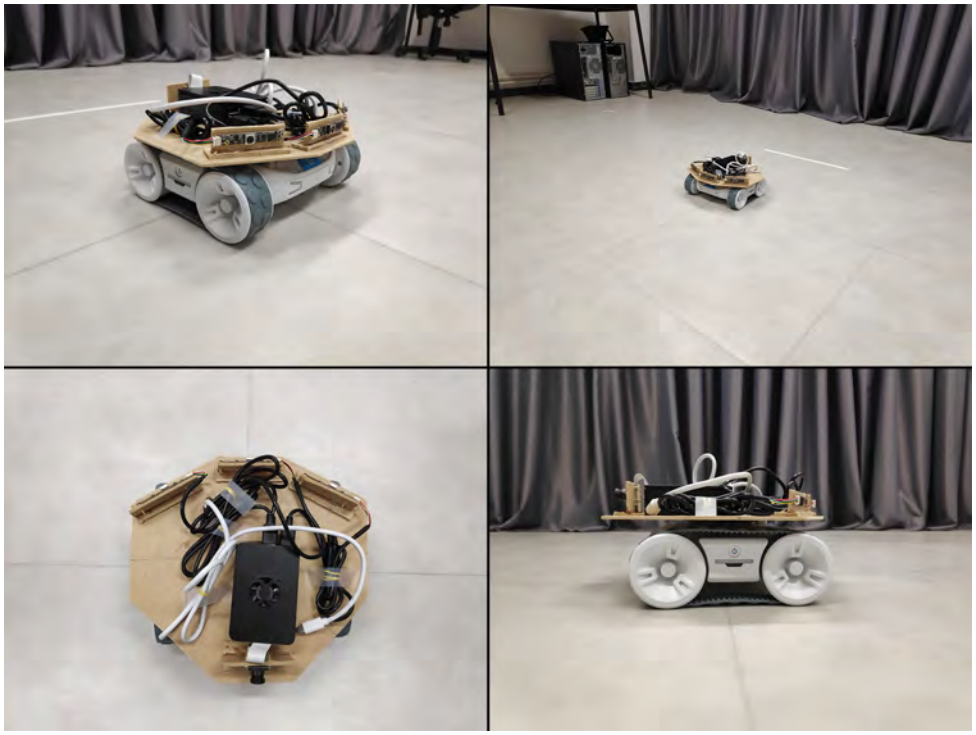


Figure 4.7: Vision module prototype on the *Sphero RVR* robot

Chapter 5

Robot recognition

Robot recognition is a very promising feature in the context of swarm robotics. Indeed, it allows the robots to know where are their peers, and to better behave in the swarm through cameras. Three methods are considered in order to achieve robot recognition using the 360-degree vision module. The implementation is done through ROS. The built package is available on *github*[1].

In this thesis, three different methods are considered for the robot recognition. First, a method based on colour segmentation using the LEDs of the *Sphero RVR* robot allows to estimate the robot position. Secondly, the *ArUco*[12, 33] library makes use of specific markers in order to estimate the position of these markers in space. Lastly, a method using supervised learning with a neural network is implemented in order to recognize the robots.

Once every camera has applied the robot recognition algorithm, the results can be compiled in order to deduce the robot recognition in the robot coordinate system. Indeed, each camera gives relative positions of the robots, but the objective is to have a global robot recognition for the whole robot, which includes all the cameras.

This chapter is structured as follows: Section 5.1 describes the method based on a colour segmentation using the LEDs of the *Sphero RVR* robot; Section 5.2 details the method using the *ArUco* library and markers; Section 5.3 exposes the method using a neural network to detect and localize the robots; Section 5.4 compares the three methods described in the previous sections from a theoretical point of view; Section 5.5 describes how the relative localization from the cameras using the methods detailed before are compiled into the robot coordinate system.

5.1 Colour segmentation

The colour segmentation is a simple algorithm in image processing. Indeed, it allows to directly segment area containing the reference colour in an image.

5.1.1 Methodology

The LEDs placed all around the robot are used in order to apply colour segmentation with the *Sphero RVR* robots. These LEDs are shown on Figure 5.1.

[1] ros_rvr_recognition package. URL: `https://github.com/ftrouill/ros_rvr_recognition`

For colour segmentation to work, a reference colour is determined. Indeed, the colour segmentation works by removing everything on an image that is too different from a given colour. With this methodology, knowing that only the robots contain this reference colour, it is possible to only segment the LEDs on the images. In this master thesis, the green colour is chosen. All the LEDs are displaying this reference colour.

Once the reference colour is chosen, the colour segmentation can be applied on the image. It segments the LEDs, but it also keeps some noise in the detection. In order to get rid of the noise, a threshold on the area of the segmented part of the image is applied. Indeed, the small areas are considered as noise, and should not be considered as LEDs. In fact, LEDs most likely have a larger surface area than noise. A segmentation of LEDs can be seen on Figure 5.2

Once the segmentation is done and noise filtered, only the LEDs are kept in the mask. However, a new problem appears. Indeed, a single *Sphero RVR* robot can contain more than one LED, which means that every LED doesn't necessarily correspond to a *Sphero RVR* robot, as it can be seen on Figure 5.1. Consequently, an algorithm on top of the colour segmentation is designed in order to gather the LEDs belonging to the same *Sphero RVR* robot.

The algorithm is described in detail in Appendix B. The main idea of this algorithm is to first identify the nature of the LED segmented. Indeed, the LEDs of the *Sphero RVR* have different shapes, depending on the LED. For example, the LEDs on the front face of the robot are much different than the ones on the sides and on the back face. For clarity, the LEDs on the front face are called *eyes*, and the other ones are named *lines*. Once the nature of the LEDs are determined, one can cluster them knowing the placement of the LEDs on the real *Sphero RVR* robot.

Once the segmented LEDs are clustered, the position of the robot can be estimated. The horizontal displacement is estimated knowing the centre of the cluster of LEDs. Then, the distance between the robot and the camera is estimated thanks to the area of the segmented LEDs, which allows to estimate the real position of the robot. This implies a calibration in order to estimate the relation between the area of the segmented LEDs and the real distance between the camera and the robot.

5.1.2 Limitations

The very first constraint is that the *Sphero RVR* robots need to display the reference colour using the LEDs, and can never change. This means that the LEDs can not be used for anything else.

Moreover, this detector also assumes that only the *Sphero RVR* robot contains the reference colour for the segmentation. Indeed, if there is another element containing this colour, the detector can misinterpret the segmented area and could consider it as a peer, while it is in fact just an element in the environment.

This detector only works under the assumption that the colour is well segmented. Hence,

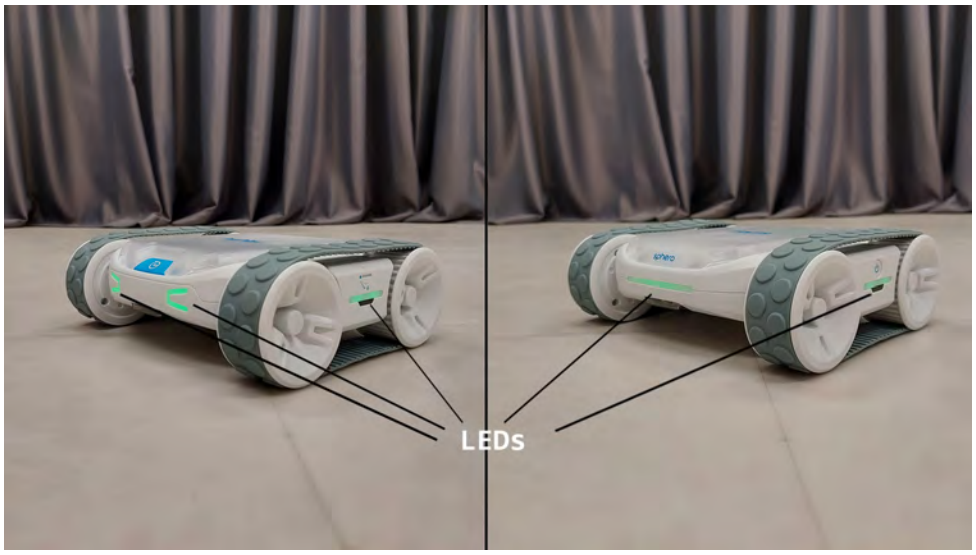


Figure 5.1: LEDs on a *Sphero RVR* robot

Figure 5.2: Colour segmentation using the LEDs of the *Sphero RVR* robot

it is sensible to change of light and colours that can occur in the environment.

Since the robot contains multiple LEDs, it can also happen that the LEDs clustering algorithm fails to correctly group the segmented LEDs. This can of course lead to errors in robot recognition.

The size of the robot is also hard to compute. Indeed, since the LEDs are already relatively small compared to the size of the robot, any noise can lead to a very different estimate for the size of the robot.

5.2 *ArUco* library

ArUco[12, 33] library is developed to allow Augmented Reality applications based on *OpenCV*. This work includes in particular localization in 3D space of specific markers. Indeed, it is able to estimate both the position and the orientation of such markers. An *ArUco* marker can be seen on Figure 5.3.

Figure 5.3: An *ArUco* marker

5.2.1 Methodology

In order to use the *ArUco* library, the *ArUco* markers need to be placed on the *Sphero RVR* robot. Since the markers need to be seen whatever the orientation of the robot, 1 marker is placed on every face of the *Sphero RVR* robot. The placement of the *ArUco* markers is described on Figure 5.4.

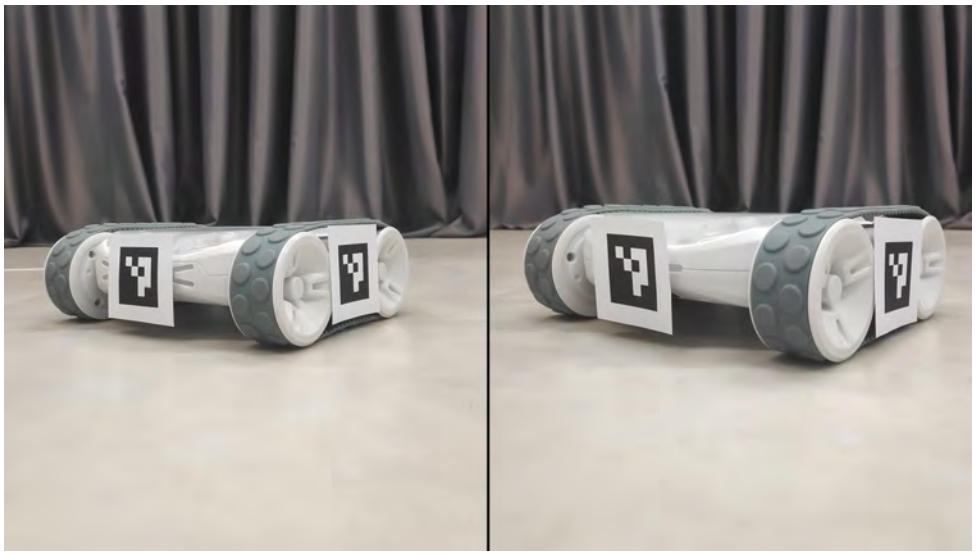


Figure 5.4: Placement of the *ArUco* markers on the *Sphero RVR* robot

Once the placement of the markers is chosen, the idea to estimate the robot position is to retrieve the estimated position and orientation of the visible markers and to add an offset to the position of the marker according to its orientation to get the centre of the *Sphero RVR* robot.
Also, it could happen that multiple markers belonging to the same robot are detected. However, it is not an issue, compared to what was described with the colour segmentation using the LEDs. Indeed, each *ArUco* marker is uniquely identified. This means that markers with the same ID can be placed on the robot, meaning that it is possible to easily differentiate markers that belong to different robots.

5.2.2 Limitations

The *ArUco* library is a very reliable way of estimating position and orientation of markers. However, 2 strong assumptions are required in order to be able to work with *ArUco*

markers.

Indeed, the main requirement of this method is that markers need to be placed on the *Sphero RVR* robot. This means that the robots can not be recognized without this additional help.

Moreover, it also relies on the fact that the markers are fully visible. Indeed, if the marker is not visible enough, or not at all, it is of course not possible for the *ArUco* library to estimate the position of the marker. Consequently, the position of the robot can not be inferred.

5.3 Supervised learning with a neural network

Neural networks are a very useful tool in computer vision. Indeed, they allow extracting features that can not be identified by more conventional methods. To achieve object recognition, supervised learning is required. This means that the neural network works by training on a given dataset, and then generalizes its knowledge to other cases that are not seen in the training dataset.

5.3.1 Methodology

Since no dataset for *Sphero RVR* robot recognition exists, a dataset creation protocol is provided in Appendix E. It describes how to generate a complete dataset for robot recognition in the context of swarm robotics. This protocol has been used in order to create a dataset for the *Sphero RVR* robot recognition.

In order to generate a neural network able to train on a dataset, one could create a neural network from scratch. However, there already exists neural network designed for object detection. Indeed, instead of creating a new neural network, it is possible to start from an existing neural network already trained on datasets. This is what is called transfer learning, because the neural network starts its training with already some knowledge, which is very powerful and advantageous for the training phase[48]. Indeed, it allows to not start from nothing and to have a working architecture for the supervised learning.

Multiple frameworks exist in order to design and implement neural networks. In the context of this thesis, *TensorFlow*[2] is used. Indeed, it is a very widely used framework for machine learning. Moreover, it exists a version designed for mobiles and edge devices, called *TensorFlow Lite*. Since a *Raspberry Pi* is used in this thesis, it is very interesting to know that this framework is easily portable.

There exists numerous neural networks designed for object detection[29]. For example, *YOLOv3* is considered as a state-of-the-art real-time object detection system[31]. However, a new type of object detection system called *EfficientDet* has been created by Google lately[44]. Compared to *YOLOv3*, *EfficientDet* is as accurate, but is more efficient in terms of operations required. That is why this architecture is chosen in the context of this master thesis. Multiple models exist using this architecture. They balance the accuracy of

[2]Tensorflow. URL: `https://www.tensorflow.org/`

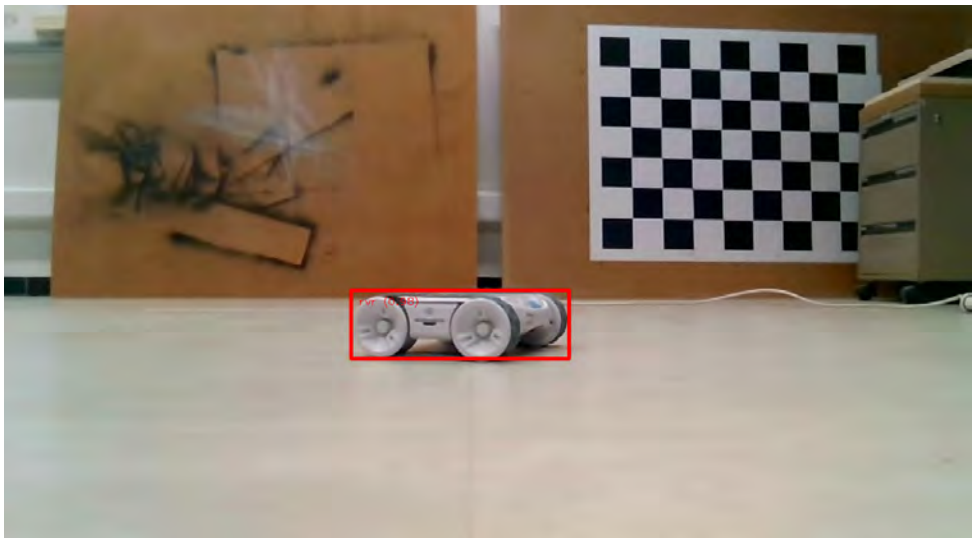


Figure 5.5: Detected robot using a neural network

the model with its latency. In the context of this thesis, since the latency is a critical factor in real-time applications, the *EfficientDet-D0* model is chosen. It corresponds to the fastest model.

Using a trained neural network for object detection, it is now possible to estimate the position of the robot. Indeed, the detected boxes on the image gives information about the real position of the object. The area of the box gives an information about the depth, knowing that the robot has a constant size, and the horizontal position of the box on the image allows to estimate the relative horizontal displacement of the robot. A detected robot is shown on Figure 5.5.

5.3.2 Limitations

The main limitation with using neural networks with supervised learning is that it is data-dependant. Indeed, the neural network is trained on a given dataset. The neural network will only be reliable in the environments described in the dataset. Hence, one should give the scope of environments in which this method should work. For this thesis, it has been reduced to the *Sphero RVR* arena at IRIDIA[17], which corresponds to the environment described in the dataset produced thanks to the protocol in Appendix E.

A second limitation using neural networks is the latency. Indeed, a neural network asks for more computational power than other traditional methods. It can then be an issue if the speed required for the detection is higher than what the model can actually do.

5.4 Comparison of the methods

The three methods described just before can be compared from a theoretical point of view. a feature table can be reported in order to allow a comparison between the methods. Regarding the features described in Table 5.1, one should know what are typical experiments in swarm robotics. One very common behaviour for swarm of robots is the

	Colour	*ArUco*	Neural network
Distance estimation	×	✓	∼
Direction estimation	∼	✓	✓
Single robot detection	∼	✓	✓
Multiple robots detection	∼	✓	∼
Very close robots detection	∼	×	∼
No assumption on robots	×	×	✓
No assumption on environment	×	✓	∼
Fast processing	✓	∼	×

Table 5.1: Feature table between the 3 methods (✓= correct, ∼ = approximate, × = incorrect)

aggregation[37]. This means that experiments involves multiple robots that can be relatively close from each other.

Regarding the feature table reported on Table 5.1, one could say that the best choice is the method using the *ArUco* library, assuming that the appearance of the robot can be modified and that the robots are not too close from each other. Indeed, in terms of position estimation, it is theoretically the best candidate. Moreover, since experiments in swarm robotics involves multiple robots, it is easier to identify each robot using the *ArUco* markers. However, if the markers are not visible enough, the robot that wear these markers will not be recognised. This could be the case when the robots are very close from each other. It can happen when the robots are aggregating or if the robots are too close from the camera, since it cannot see all the tags. Indeed, it could happen that the *ArUco* marker is not fully visible, and only a part of it is captured by the cameras.

On the other hand, the method using a neural network could also be a good fit in the context of swarm robotics. The only bad point, which is very important in the context of real-time experiments, is the speed of the algorithm. Indeed, neural networks requires more time to process than the two other methods. If the time spent is too high, this method will not be suitable enough for robot recognition in real-time in the context of swarm robotics. However, besides this disadvantage, this method could allow position estimation in every case, without any additional modification of the robots. Moreover, it could also identify robots that are very close.

5.5 Robot recognition compilation from cameras

Once the cameras have estimated the relative position of the robots, the results need to be merged together in order to have a robot recognition for the whole *Sphero RVR* robot. Indeed, the methods mentioned in the section 5.1, section 5.2 and section 5.3 are applied for each camera. Hence, it gives an estimation of the robot positions relative to each camera. Consequently, these results need to be compiled in order to have a global robot recognition for the whole robot.

Knowing the position of the cameras, it is easy to retrieve the position of the recognised robots in the system of coordinate of the robot. Indeed, knowing the relative position

and orientation of the cameras to the position of the robot, one could apply geometric transformations in order to have everything in the same coordinate system. The relative position of the robots is given in a 2D plane corresponding to the ground.

However, it can happen that 2 different cameras detect the same robot. To avoid cases where the robot recognition counts twice the same robot, a last verification on the detected robots is applied. It computes the distance between pairs of detected robots, and merge them together if they are close enough.

The robot recognition is given every time one of the cameras has estimated the position of the robots. This means that the robot recognition for the whole robot can contain older estimations for the cameras that did not refresh their estimation yet.

Chapter 6

Methods comparison on experimental datasets

In order to evaluate the 3 methods described in chapter 5, an idea is to compare their performance on an experimental dataset of images. This comparison aims at assessing the accuracy of robots position estimation for every method in the context of swarm robotics. It identifies from an experimental point of view the strengths and weaknesses of each method.

This chapter is structured as follows: Section 6.1 describes the dataset used for this methods comparison; Section 6.2 details the metrics used for the comparison; Section 6.3 exposes and analyses the results.

6.1 Experimental datasets

The idea to compare the three methods is to evaluate them on a given experimental dataset.

This dataset contains images from the IRIDIA research laboratory[17] Sphero RVR arena. It describes a square with a side length of 3 metres.

In order to create this dataset, multiple cases are considered. Indeed, some base cases are taken into account,, but also cases where robots are aggregated are considered, since it is a very common behaviour in swarm robotics. Lastly, cases where a robot is very close or very far from the camera are considered. These considerations have lead to multiple configurations.

1. Base cases:

   These cases corresponds to situations where the robots are close enough to be detected, without being too near from the camera. Also, these robots are isolated.

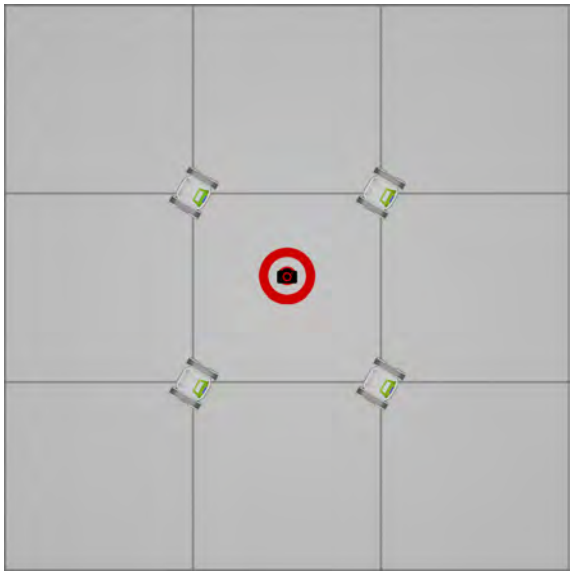


Figure 6.1: Base case - configuration 1

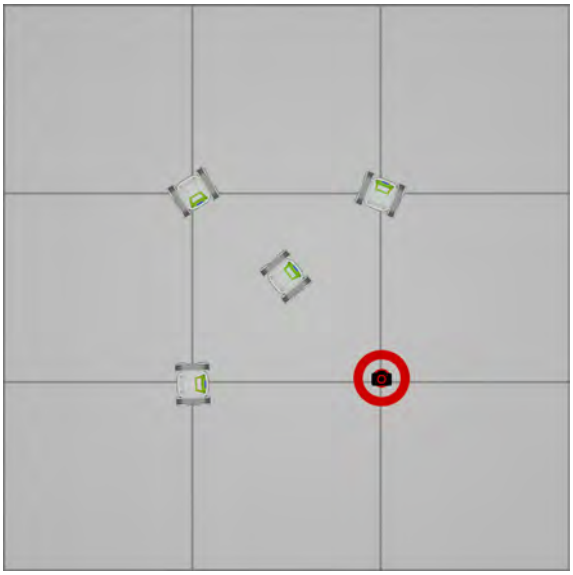


Figure 6.2: Base case - configuration 2

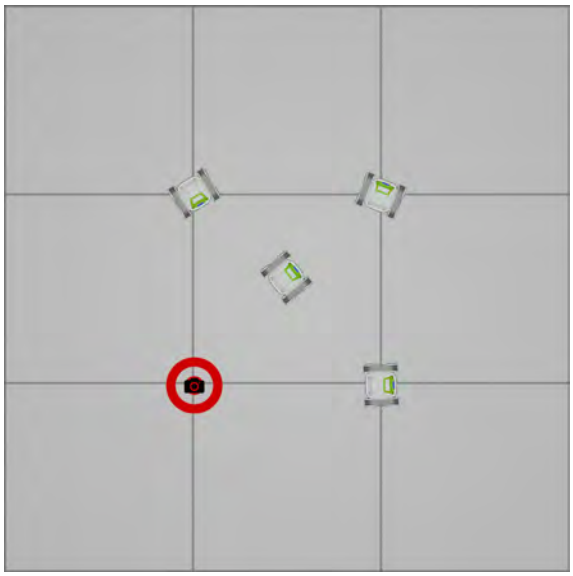


Figure 6.3: Base case - configuration 3

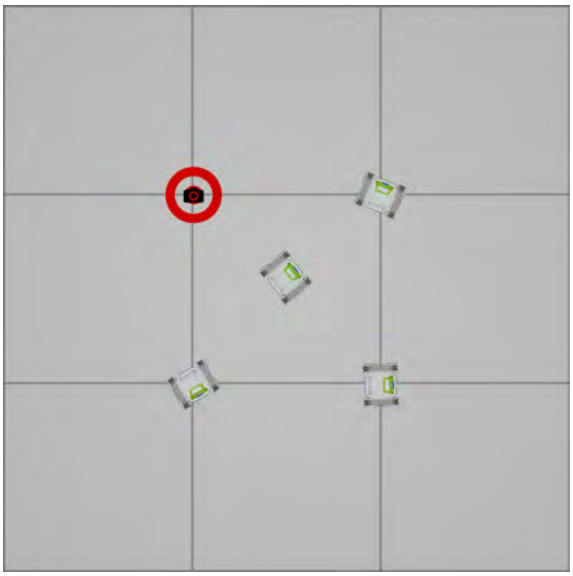


Figure 6.4: Base case - configuration 4

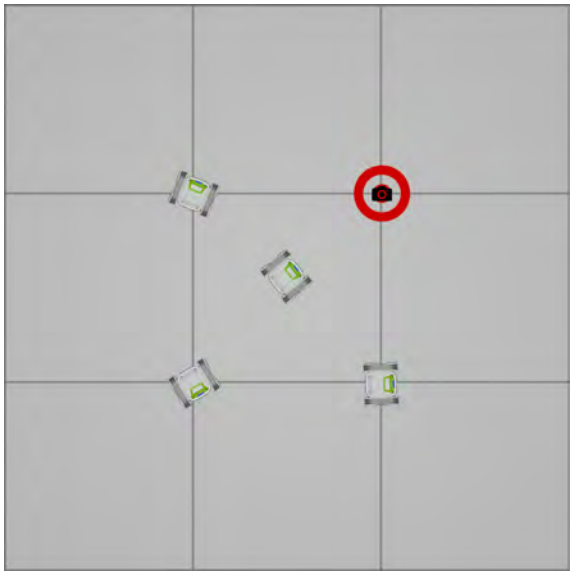


Figure 6.5: Base case - configuration 5

2. Aggregation cases:

   These cases takes into situations where multiple robots are very close from each other.

Figure 6.6: Aggregation cases - configuration 1

Figure 6.7: Aggregation cases - configuration 2

Figure 6.8: Aggregation cases - configuration 3

Figure 6.9: Aggregation cases - configuration 4

3. Close range cases:

   Such cases considers a very close robot from the camera.

Figure 6.10: Close range cases - configuration 1

Figure 6.11: Close range cases - configuration 2

4. Long range cases:

   This describes cases where one robot is relatively far from the camera.

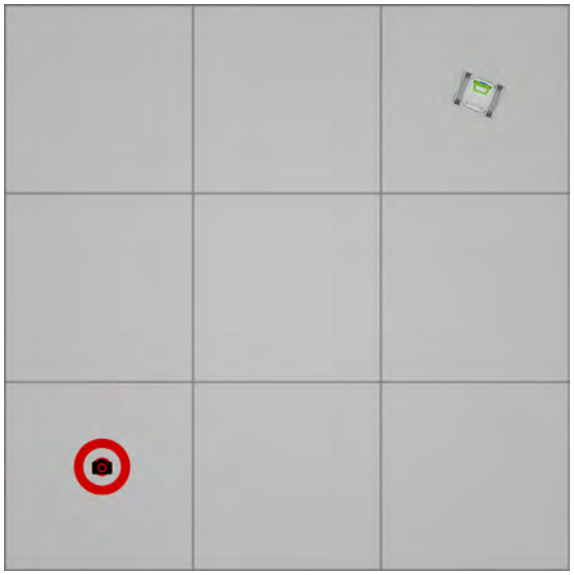


Figure 6.12: Long range cases - configuration 1

Figure 6.13: Long range cases - configuration 2

Using these configurations, images are taken in precise orientation, in order to know exactly the relative position of the robots. This way, it is possible to compute the position error estimation for every method. The resolution of the images taken is 1280x720.

The initial idea was to use the same dataset to compare the three methods. However, it is not possible since every method works under different assumptions. Indeed, the experimental dataset for the method using colour segmentation includes the same reference colour for all the *Sphero RVR* robots. This means that one can not assess the efficiency of the method using a neural network with the same dataset as the one for the colour segmentation. This conclusion is the same concerning the method using *ArUco* markers. Lastly, a common experimental dataset between the method with colour segmentation and the one with *ArUco* markers is also biased by the fact that the *ArUco* markers would need to be placed differently in order to not hide the LEDs. That is why the best solution in this situation is to generate three different datasets in the same conditions, which includes the same placement of robots and same orientation for the camera that shoot the images. An example image from each dataset for the same configuration is shown on Figure 6.14.

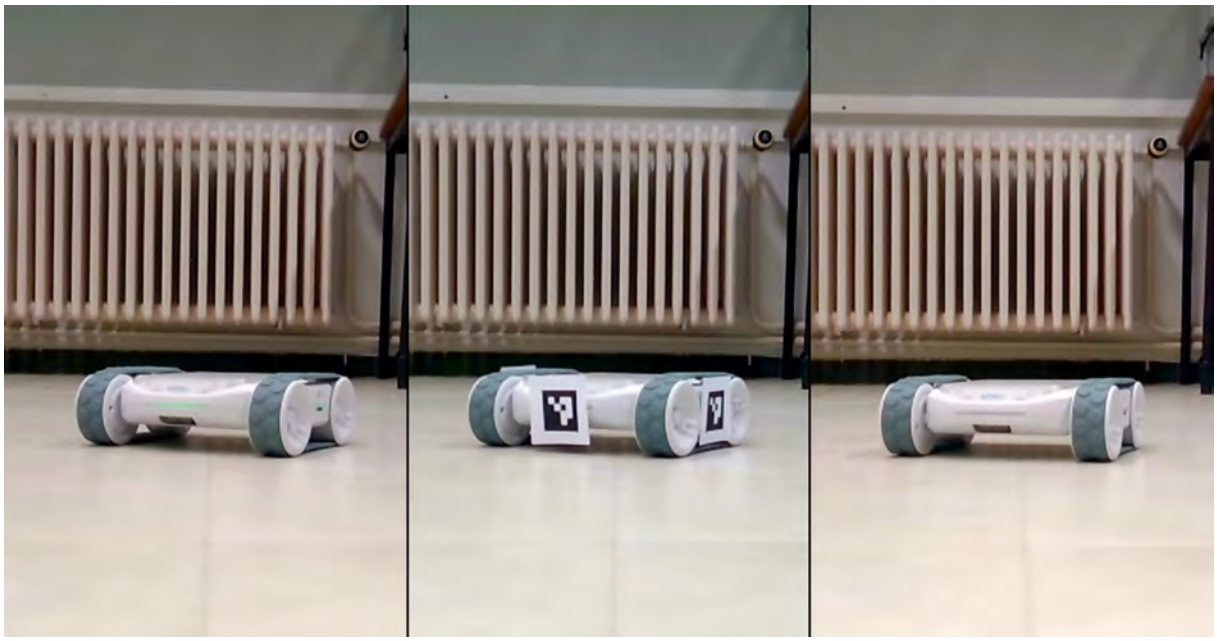


Figure 6.14: An image from the experimental dataset for the method using colour segmentation (left), *ArUco* markers (middle) and a neural network (right)

6.2 Metrics

The different methods are compared on the accuracy of their robots position estimations. Multiple simple metrics can be used to assess this accuracy.

First of all, the number of robots detected on the image is an important metric. Indeed, it is already a strong and very simple metric in order to compare different algorithms. Moreover, this already indicates if an algorithm has an error. In order to measure the performance on this indicator, the root-mean-squared error (RMSE) is used over the experimental datasets, in order to penalize more the big differences with the real number of robots on the image.

A second metric is the error on the position of the detected *Sphero RVR* robots on the image. The error is simply the distance between the real position of the *Sphero RVR* robot, and the estimated position given by the algorithm. The RMSE is also used in order to quantify the relative error. It is indeed a good indicator for comparison, as done in many works involving position estimation[24, 47].

Another metric is the estimated distance between the detected robots and the camera. Indeed, by computing the RMSE between the real distance and the estimated distance, it is possible to assess how good a method is for distance estimation of *Sphero RVR* robots. As said before, the RMSE for such estimates is a good indicator.

Also, the error on the direction, expressed in radians, can be computed. The RMSE is also used in this case, and allows seeing how well an algorithm can estimate the direction of the detected robots.

Lastly, a last metric that can be considered is the time spent by the algorithm to compute the result. This is indeed very interesting when real-time operations are required. This is expressed as a frequency given by the ROS topic for the position of the *Sphero RVR* robots.

6.3 Results and analysis

The results of the evaluation of each method on the entire experimental datasets are reported in Table 6.1.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	2.429	1.742	0.602
$RMSE_{position}$ **(cm)**	23.478	10.405	21.271
$RMSE_{distance}$ **(cm)**	21.582	6.890	18.050
$RMSE_{direction}$ **(rad)**	0.2386	0.0432	0.2239
Frequency (Hz)	15.018	4.823	1.281

Table 6.1: Comparison between the 3 methods on the entire experimental datasets

A general comment about the performance of the three methods is that the distance estimation between the robots and the camera is the main problem in position estimation. Indeed, the error on the distance estimate is relatively close to the error on position estimate, which shows that the major part of the error on position estimation comes from the distance estimation.

As it can be seen, the method using colour segmentation seems to be the worst, since it does not have a better accuracy in position estimation than the two others, nor in the number of robots estimate. Its only advantage is its high frequency.

The method using *ArUco* seems to be the most accurate one, by having the lowest error on position estimation. It also has a higher frequency compared to the method using a neural network, which is really slow. However, Table 6.1 also indicates a higher error on the number of robots estimated for the method using *ArUco* markers than the one using a neural network.

One could also verify the performance of each method for each type of cases considered in the experimental datasets, which are base, aggregation, close range and long range cases.

6.3.1 Base cases

The results on base cases are reported on Table 6.2.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	1.323	0.5	0.5
$RMSE_{position}$ **(cm)**	34.592	8.924	10.000
$RMSE_{distance}$ **(cm)**	34.561	7.958	9.967
$RMSE_{direction}$ **(rad)**	0.0181	0.0469	0.0084

Table 6.2: Comparison between the 3 methods on the base cases

On base cases, it can be seen that the method using *ArUco* markers gives the best estimation in terms of position. The method with the neural network is also relatively close in terms of accuracy to the one using the *ArUco* library. Also, the colour segmentation method gives worse results than ones of the two other methods.

It could be surprising to see that even in these base cases, there is an error on the number of robots detected. Indeed, it is sometimes impossible for the algorithm to estimate the position of a robot, because it is hidden by other robots. Such case is described on Figure 6.15.

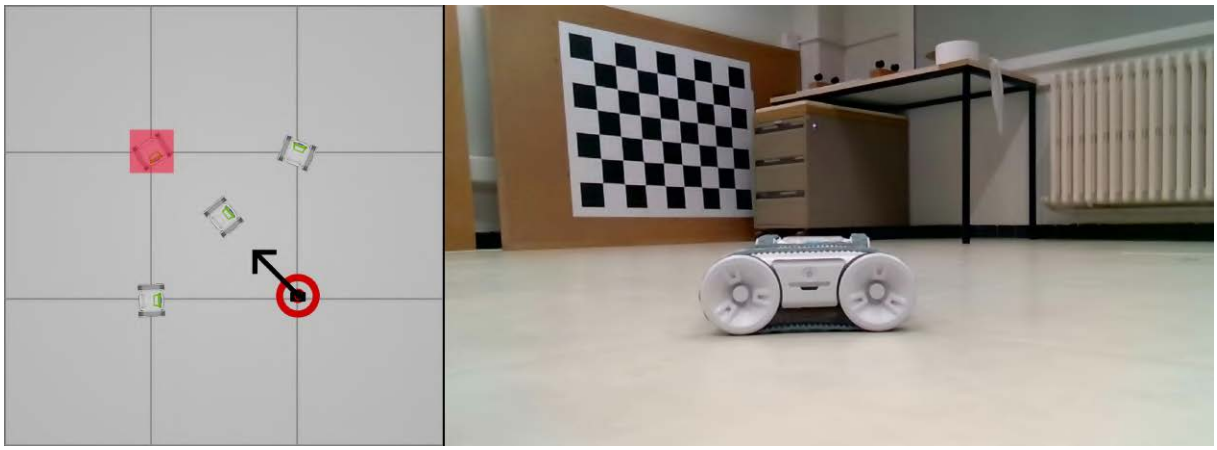


Figure 6.15: Robot highlighted in red hidden by the other robots in a base case

That is why the metrics are computed once again by removing the occluded robots from the expected robots recognised in the experimental datasets. The results are reported in Table 6.3.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	1.000	0	0
$RMSE_{position}$ **(cm)**	34.592	8.924	10.000
$RMSE_{distance}$ **(cm)**	34.561	7.958	9.967
$RMSE_{direction}$ **(rad)**	0.0181	0.0469	0.0084

Table 6.3: Comparison between the 3 methods on the base cases without occlusion cases

If the occluded robots are not taken into account, the error on the number of robots is 0 for the *ArUco* library method and the neural network one.

6.3.2 Aggregation cases

The results on aggregation cases are reported on Table 6.4.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	3.571	2.337	0.672
$RMSE_{position}$ **(cm)**	30.931	13.189	27.145
$RMSE_{distance}$ **(cm)**	27.859	10.833	24.201
$RMSE_{direction}$ **(rad)**	0.2923	0.0392	0.2273

Table 6.4: Comparison between the 3 methods on the aggregation cases

When the robots are aggregated, one can see that the results are worse than in the base cases for each method. Moreover, it can be seen that the error on the number of robots for the colour segmentation and *ArUco* library methods is much higher than the one using a neural network. However, on the detected robots, the position estimation is still better for the *ArUco* library method.

Aggregation cases also contains occlusions situations. One of these cases is described on Figure 6.16.

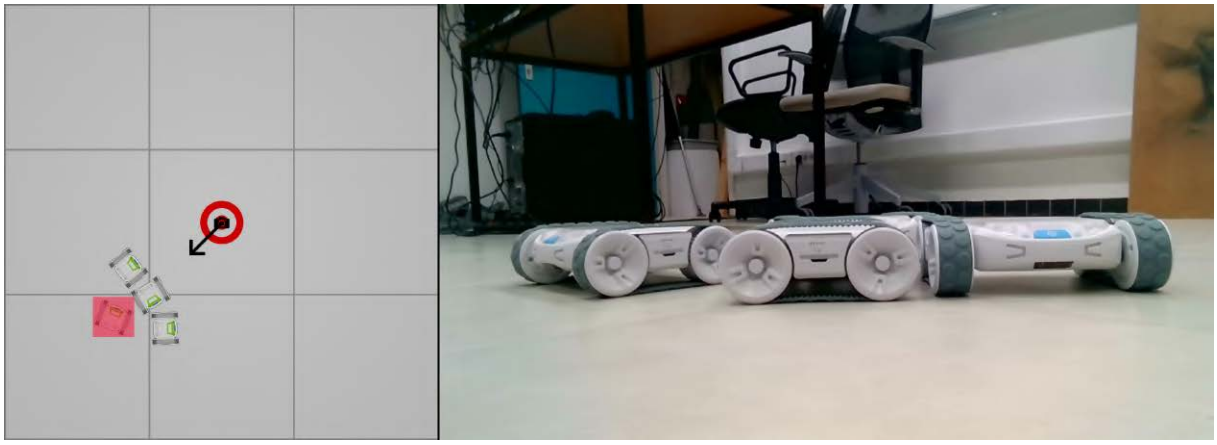


Figure 6.16: Robot highlighted in red hidden by the other robots in an aggregation case

The metrics are computed once again without the occlusions cases, and the results are reported in Table 6.5.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	2.761	1.840	0.475
$RMSE_{position}$ **(cm)**	30.931	13.189	27.145
$RMSE_{distance}$ **(cm)**	27.859	10.833	24.201
$RMSE_{direction}$ **(rad)**	0.2923	0.0392	0.2273

Table 6.5: Comparison between the 3 methods on the aggregation cases without occlusion cases

The error on the number of robots detected decreases for every method. However, even with this consideration, it appears that the error on the number of robots detected by the *ArUco* library is relatively high. Indeed, some cases are really difficult and nearly impossible for this method to work. For example, when the markers are not fully visible, the detection can not succeed. It is for example the case when the robots are too close. Such case is described on Figure 6.17. This explains why this error is higher than expected for the *ArUco* method.

Figure 6.17: *ArUco* markers not fully visible on a close robot

6.3.3 Close range cases

The results on close range cases are reported on Table 6.6.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	0.632	2.646	0.378
$RMSE_{position}$ **(cm)**	21.391	×	22.553
$RMSE_{distance}$ **(cm)**	18.410	×	17.075
$RMSE_{direction}$ **(rad)**	0.3193	×	0.4321

Table 6.6: Comparison between the 3 methods on the close range cases

As it can be seen, the methods using the *ArUco* library struggles to recognise the robots. Indeed, for this method to work, the corresponding markers need to be visible enough or completely on the image. The method using *ArUco* markers requires that the whole marker is visible on the image. This case can be seen on Figure 6.18. However, it can be seen that the colour segmentation method has a lower error on the number of robots detected. Indeed, in a very close range, if at least part of the LEDs is visible on the image, the algorithm is able to detect the robot. The method using a neural network is still able to recognize the robots because it can detect parts of the robots and generalize that it corresponds to a full robot.

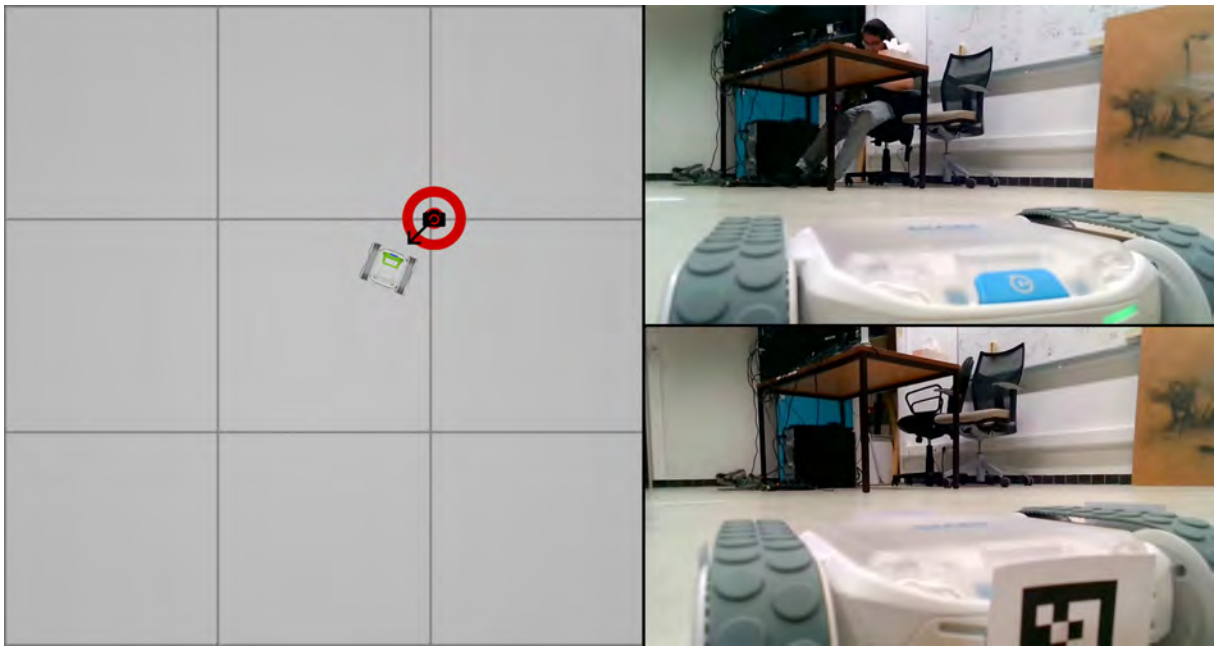


Figure 6.18: Close range case with LEDs (top) and *ArUco* markers (bottom)

6.3.4 Long range cases

The results on long range cases are reported on Table 6.7.

	Colour segmentation	***ArUco* library**	**Neural network**
$RMSE_{number}$	2.449	2.449	0.707
$RMSE_{position}$ **(cm)**	×	×	25.157
$RMSE_{distance}$ **(cm)**	×	×	13.062
$RMSE_{direction}$ **(rad)**	×	×	0.0773

Table 6.7: Comparison between the 3 methods on the long range cases

When the robots are very far, it is very difficult for the methods using colour segmentation and *ArUco* library to recognize the robots. Indeed, when the robots are very far, the colour segmentation is too difficult, since the visible LEDs are too small to not be considered as noise. The same goes for the *ArUco* library method, which struggles to identify the *ArUco* marker when it is very far. That is why these two methods are not able to recognize the robots at a long range, while the one using a neural network gives consistent results.

6.3.5 Conclusion

To conclude, the comparison on this experimental datasets confirms the theoretical expectations concerning the capabilities of the algorithms, described by Table 5.1. Indeed, it shows that the *ArUco* markers method is the most reliable one in general. However, it struggles to recognize the robots when the robots are too close or too far, while the method using a neural network is most of the time able to estimate the position of these robots, even if the average accuracy on the position is lower than the accuracy of the method using the *ArUco* library. The method using colour segmentation is clearly not a good candidate for this task. Indeed, such algorithm is able to efficiently detect parts of

the image containing a reference colour, corresponding to the LEDs in this case, but it struggles to estimate the robot position based on the different LEDs segmented. As a last remark, the method using *ArUco* markers is the most accurate and fastest one, but the method using a neural network gives satisfying results without any additional marker.

Chapter 7

Real-time robot recognition experiments

In order to assess the use of the vision module in a real-time application for robot recognition, experiments are conducted with the vision module prototype and robots moving around. The experiments are of course not taking into account all the potential cases that can be encountered in real experiments. However, it aims at evaluating the performance of the robot recognition methods using the vision module in simple cases, that can occur in swarm robotics missions.

This chapter is structured as follows: Section 7.1 describes the experimental environment for these experiments; Section 7.2 presents the experiments; Section 7.3 describes the metrics used to evaluate the results; Section 7.4 exposes and analyses the results.

7.1 Environment

The experiments are conducted in the IRIDIA research laboratory[17] Sphero RVR arena. It is an arena described by a square with a side length of 3 metres. The arena is delimited by tape on the floor. It can be seen on Figure 7.1.

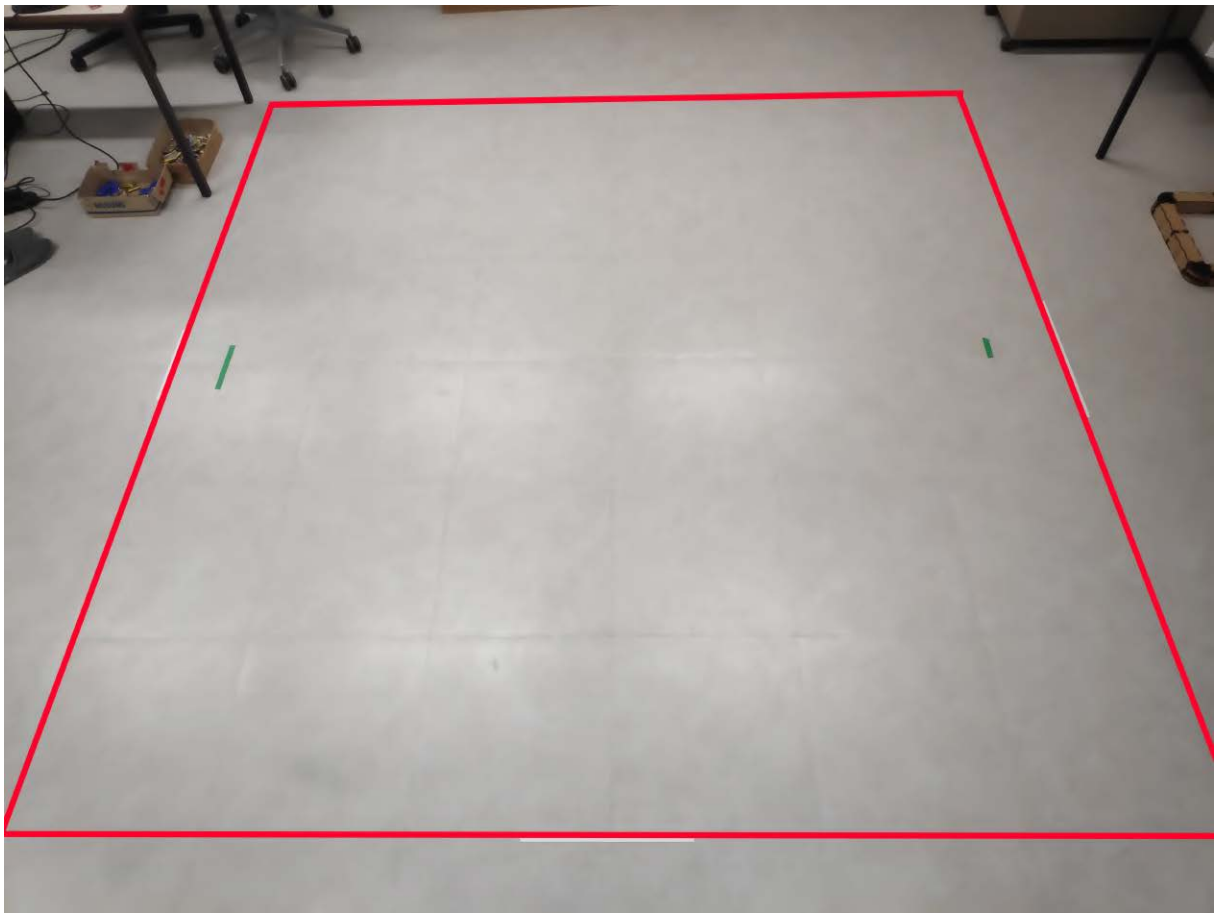


Figure 7.1: The IRIDIA *Sphero RVR* arena

The vision module is placed on a robot, and the robot recognition methods are implemented using *ROS*, the same way as what is described in chapter 5.

It can also be mentioned that *ROS* contains tools to record data on topics, such as *rosbag*[1]. This is what is used in order to record the results of the experiments.

7.2 Experimental setups

For these experiments, the vision module is powered through the battery of the *Sphero RVR* robot.

The goal of these experiments is to evaluate the robot recognition using the vision module for swarm robotics purposes. That is why these experiments are using multiple robots. The basic idea is to place the robot with the vision module in the centre of the arena, and to make the other robots move around it.

The setup used for the experiments is described on Figure 7.2.

[1]rosbag | ROS. URL: `http://wiki.ros.org/rosbag`

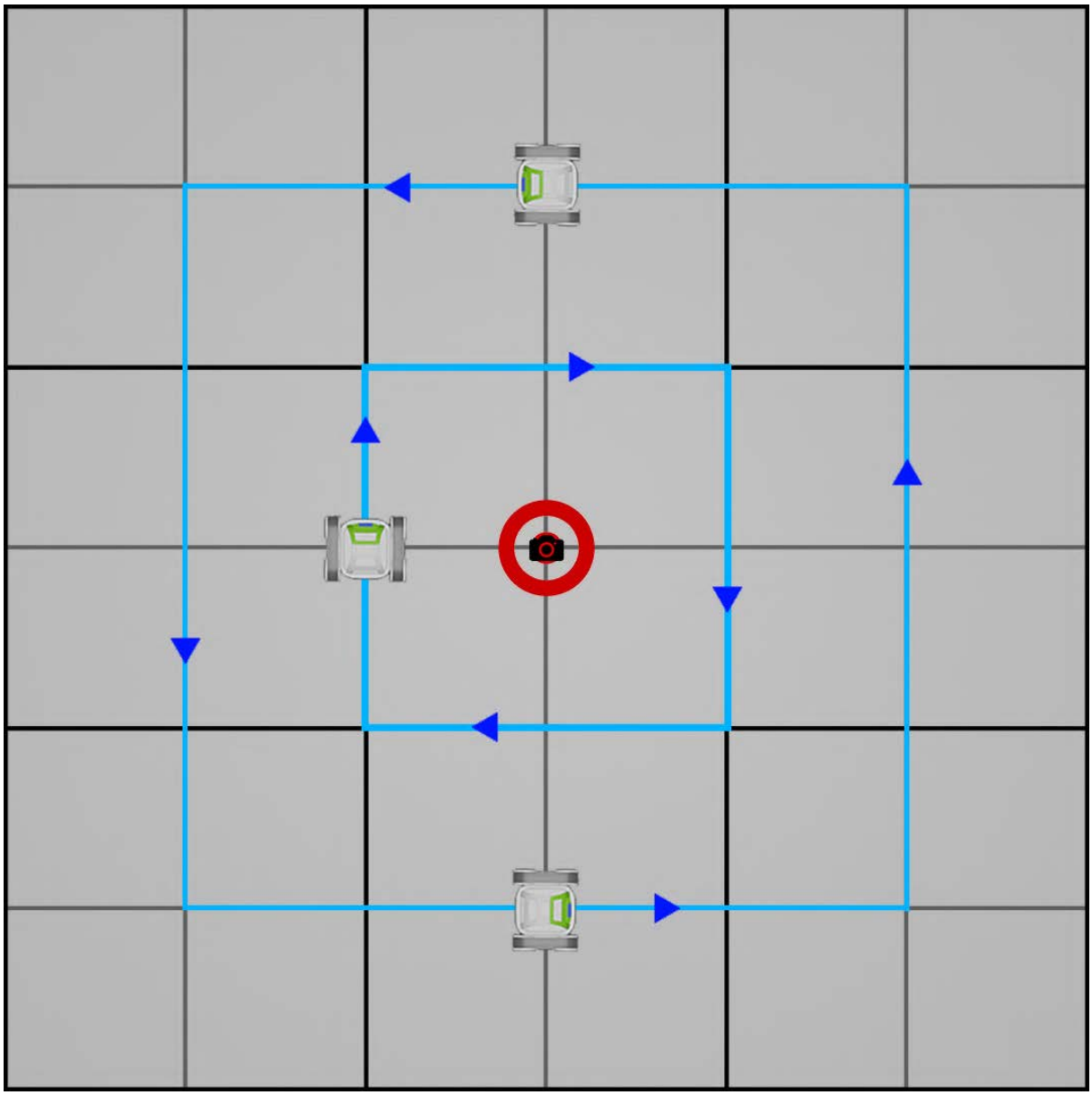


Figure 7.2: Experimental setup for the experiments in the *Sphero RVR* arena

The robot with the vision module stands in the centre of the arena. Three other robots move around it. One describes a square with a side length of 1 meter, which corresponds to the closest robot to the vision module. The two other trace another square with a side length of 2 meters, and go in the opposite direction to the first.

In order to get the real position of the robots, a camera placed above the arena is also recording the whole experiment. This allows to capture the real robot positions during the experiments. The estimations are then compared to the real positions to compute the error.

For these experiments, only the *ArUco* library and neural network methods are considered. Indeed, chapter 6 proves that the colour segmentation method gives very poor results in terms of position estimation.

With this experimental setup, the only parameters for these experiments are the speed of the robots moving around, and the time of the experiment. The time of the experiment is determined by the time spent by the two furthest robots to complete one square. The speed of the robots is an important parameter. Indeed, in chapter 6, the methods are evaluated on a dataset with stationary robots. Consequently, the idea here is to move the robot faster and faster to see if the robot recognition still work with moving robots. The range of speed goes from 0.1m/s, which corresponds to the minimal speed for the robot to move, to 0.3m/s, by step of 0.1m/s. This means that the experiment is reproduced 3 times for each method, leading to a total of 6 experiments.

7.3 Metrics

The metrics used are similar to what is described in section 6.2. This includes the error on the number of robots detected and on the estimations on the position, the distance and the direction of the recognized robots. It also includes the frequency of the two methods considered for these experiments.

7.4 Results and analysis

The results of the experiments are presented in these sections. First, the robot recognition using the *ArUco* markers is evaluated. The neural network method is also analysed. Since these two methods have different frequencies, the evaluation of each method is done when it gives a response for robot recognition. Therefore, since the neural network method has been shown to be slower, it will be evaluated on fewer samples because the experiment has the same duration for both methods.

7.4.1 Method using *ArUco* markers

The results of the experiments are reported in Table 7.1.

Speed of *Sphero RVR* robots (m/s)	0.1	0.2	0.3
$RMSE_{number}$	1.255	2.308	2.384
$RMSE_{position}$ **(cm)**	52.538	87.524	93.787
$RMSE_{distance}$ **(cm)**	17.760	21.964	16.872
$RMSE_{direction}$ **(rad)**	0.7823	0.8708	0.8867
Frequency (Hz)	1.699	1.849	1.688

Table 7.1: Results for the method using *ArUco* markers

First of all, it can be seen that the frequency is lower than what is described on the evaluation on the experimental dataset, which is detailed in section 6.3. This is explained by the fact that there are multiple detections at the same time, since there are mutliple cameras.

These results are consistent with what is discussed in chapter 6. Indeed, one can see that the error on the number of detected *Sphero RVR* robots is higher than during the evaluation through the experimental dataset. In fact, the real number of robots is constant

and set to 3. Sometimes, the vision module does not allow to capture all of them with the current configuration of the cameras. Indeed, since the vision module is focused on the front part of the *Sphero RVR*, there are blind spots on the sides of the robot. This explains why this number is higher.

Also, one can see that the faster the robots move, the higher the errors for every metric. Indeed, the faster the robots move, the hardest it is for the *ArUco* library to recognize the markers. In fact, when an object move in front of a camera, a motion blur may appear. Such case is shown on Figure 7.3. It explains why the detection sometimes fails and consequently, why it is much harder for the robot recognition to work. This effect already appears when the robots are moving slowly, but increases with the speed.

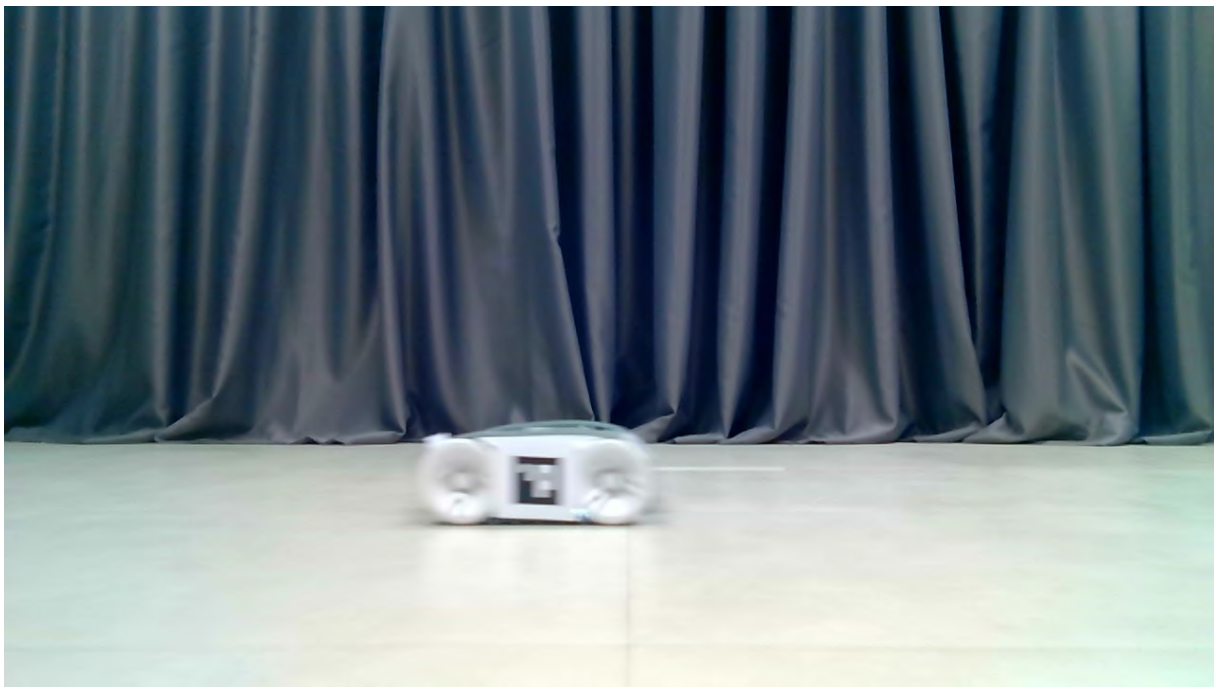


Figure 7.3: Motion blur on a *Sphero RVR* robot with *ArUco* markers

7.4.2 Method using a neural network

The results of the experiments are reported in Table 7.2.

Speed of *Sphero RVR* robots (m/s)	0.1	0.2	0.3
$RMSE_{number}$	0.727	0.567	0.554
$RMSE_{position}$ **(cm)**	55.289	66.176	51.770
$RMSE_{distance}$ **(cm)**	16.648	16.974	17.812
$RMSE_{direction}$ **(rad)**	0.9368	0.7917	0.7099
Frequency (Hz)	0.816	0.803	0.854

Table 7.2: Results for the method using a neural network

As a very first remark, it can be seen that the frequency is very low. Indeed, it corresponds to a robot recognition less than once every second. This is of course a very bad point for this method. This can be explained by the fact that neural networks requires

more computational power than the *ArUco* detection.

Secondly, one can see that the errors are higher compared to what was expected from the results with the experimental dataset. The major reason for that is the very low frequency. Indeed, as explained before for the experiments with the *ArUco* library method, the robots are moving at a given speed. However, since the robot recognition is always done on previous images, the robots have the time to move.

The error should increase with the speed of the robots, as explained for the *ArUco* method. However, it can be seen in Table 7.2 that the error does not follow this trend. In fact, it can be seen that the neural network is able to recognize robots even if there is some motion blur on the image, as shown in Figure 7.4. However, even if the detection are relatively satisfying, the very low framerate can be a problem if some decisions are required in a real-time application, since the detection is always done on past images that can be considered as outdated.

Figure 7.4: *Sphero RVR* robot with motion blur detected by the neural network

7.4.3 Conclusion

To conclude, one can see that the error for both methods is much higher than what is found during the evaluation on the experimental dataset. Indeed, one of the main reasons for this larger errors is the very low frequency.

First of all, the time spent by the method to apply the detection implies a lag for the robot recognition. Indeed, the detection is always based on past images. Because of the low frequency, the robots can move between the moment the image starts to be processed, and the moment it gives an estimation of the robots positions.

Secondly, since the global robot recognition compiles results from the 4 cameras, working asynchronously, and that an estimation for the whole robot is given every time a recognition is done for one of the cameras, the robot recognition can contain outdated information. Indeed, when one of the cameras gives an estimation, the positions estimate for the three other cameras remains the same as before when producing an estimation for the whole robot. This means that the global robot recognition contains past recognition. This of course increases the error since a given recognition for a camera is in average kept for 4 global robot recognition, since there are 4 cameras. However, keeping the last recognition for every camera for the global robot recognition always give the most updated robot recognition for the robot.

Consequently, the results in real-time experiments are worse than what is found during the evaluation through experimental datasets. The main reason for this decrease of performance is that the robots are moving. In addition to the problem described just before, motion blur that appears on the images can lead to wrong or no detection of the robots for the *ArUco* markers method, even for really slow robots. However, for the neural network method, it has been shown that it is still able to recognize robots when they are moving faster. Nevertheless, its low frequency makes it currently hard to use in a real-time application since the information got from the cameras can be outdated.

Chapter 8

Future developments

This thesis aimed at providing a vision module, and to implement robot recognition algorithms. However, multiple improvements and developments can be considered thanks to the work achieved in this thesis.

8.1 Neural network method improvement

It has been shown that the neural network method for the robot recognition was the slowest method. The very low frequency of this method makes it very hard to use in a real-time application. In order to improve that, one could use an additional GPU (Graphical Processing Unit) or TPU (Tensor Processing Unit) for the *Raspberry Pi*. Such unit could lead to a great acceleration in terms of processing time. *Coral* is providing multiple products in order to include a TPU in the configuration[1]. This could greatly increase the performance of algorithms using neural networks with the framework *TensorFlow*[2].

However, this also increases the budget allocated per vision module. This requires a work of financial and budget analysis, balancing the benefits added by such additional processing unit and the corresponding cost. Nevertheless, this will greatly improve the efficiency of neural network algorithms used with the vision module, which can include more than the *Sphero RVR* robot recognition. Indeed, deep learning allows a lot of applications in computer vision, and it could be a real gain on the long-term to implement new interesting features using neural networks for swarm robotics.

8.2 Depth estimation enhancement for robot recognition

The results have shown that the methods have a relatively low error concerning the direction of the detected robots. However, the distance estimation, closely related to depth estimation, could be enhanced. This would allow better robots position estimation.

The first idea to improve the depth estimation would be to simply use RGB-D cameras instead of simple RGB cameras. These cameras allow to estimate the depth thanks to a

[1]Coral - Products. URL: `https://coral.ai/products/`

[2]Edge TPU performance benchmark. URL: `https://coral.ai/docs/edgetpu/benchmarks/`

dedicated sensor. This could lead to a much higher accuracy in depth estimates, but it would also increase the budget for the vision module. That is why a financial analysis would be required.

The second idea would be to use stereo cameras, or to add more cameras covering the same angle. Indeed, this way, stereo matching can be done in order to estimate the depth[27]. This of course increases the budget for the vision module, since it requires either more cameras, or stereo cameras, that are more expensive than simple RGB cameras.

Another idea would be to use the information of a module dedicated for depth estimation, such as a LiDAR. Indeed, it allows to estimate the depth all around the robot at a given height. Using both the robot recognition algorithms described in this thesis and a depth sensor could enhance the position estimates.

Lastly, an idea would be to use depth estimation algorithms, thanks to deep learning. Indeed, single image depth estimation (SIDE) is currently a field of interest in computer vision[25]. It makes uses of neural networks in order to estimate the depth map of an image. This approach could enhance the depth estimation for the robot recognition.

Also, an enhanced depth estimation could also be useful for other applications and features than robot recognition.

8.3 Features using the vision module

Now that the vision module is created and implemented on the *Sphero RVR* robot, one could implement a lot of new features.

For example, the environment could now generate more interactions in the swarm robots system. Indeed, without the vision module, the interactions with the environment are limited for the swarm robots. Using the vision module, it would be much easier to include features that are part of the environment, such as symbols or markers placed on the walls. This vision module greatly increases the potential stream of information available for the robot, and by extension for the whole swarm of robots.

Human-swarm interactions could also be considered. Indeed, there exists already multiple works using such approach. For example, Kakish, Vedartham, and Berman[19] can interact with a swarm of robots using hand gesture recognition.

Chapter 9

Conclusion

In this master thesis, I have presented a 360-degree vision module and methods for *Sphero RVR* robot recognition in swarm robotics. This allows the robot to recognize and localize its peers. This 360-degree vision module also extends the capabilities of the *Sphero RVR*. Indeed, this vision module can be used to extract much more features in the future for the robot, which greatly increase the stream of information for it, and by extension, for a whole swarm of robots.

In order to design the vision module, I reviewed what is currently done in the literature in swarm robotics, but also in robotics in a more general way. I also reviewed 360-degree cameras on the market. Once I made a selection of suitable cameras for this vision module, I conceived multiple designs in order to find the best in the context of swarm robotics for robot recognition. I also chose the design of the vision module considering the potential new features that can be extracted with this module, including human-swarm interactions.

The robot recognition is done using 3 different methods, which are based on colour segmentation, on the *ArUco* library, and on a neural network. This robot recognition is implemented using *ROS*. The three methods are designed and compared from a theoretical point of view.

I have then compared the methods designed for the robot recognition using an experimental dataset. This allowed to assess the performance of each method when the robots are not moving. The results of this comparison have shown that the method using the *ArUco* tags and the one based on a neural network were relatively good compared to the one with colour segmentation, in term of position estimation. Moreover, from all the metrics used to compare the algorithms, it appeared that the distance estimation was one of the main issue and was hardly computed for every method. Also, the method using a neural network gives satisfying results in terms of position estimation without the addition of a marker on the robot, but appears to take more time to compute.

I have then conducted real-time experiments to assess the efficiency of the robot recognition algorithms in an online application. The error on the estimations are higher than what is found on the evaluation through the experimental datasets. Indeed, the time spent by the methods to compute the robot positions when they are moving implies higher errors. Moreover the method using *ArUco* markers is supposed to be the most accurate one. However, motion blur can appear when the robots are moving faster, which

greatly decreases the performance of this method. The neural network method, which is even slower, can nevertheless detect robots on images containing motion blur. However, because of its very low frequency, it is currently not suitable for a real-time application.

This vision module and the current implementation allow to easily implement new potential features. Swarm robotics is a very interesting research field and having such autonomous systems could really enhance real situations. Adding vision to such robots greatly increases the possible information and features they can receive and extract. The possibilities are nearly countless, and interactions with the environment are much easier thanks to computer vision and to the vision module. I strongly believe that computer vision can highly enhance the current information a robot swarm receives and, by extension, can lead to swarms of robots able to complete more complex tasks.

Appendix A

Designs for the vision module

All the designs considered for the vision module are reported in this appendix.

The three selections of cameras are mentioned here.

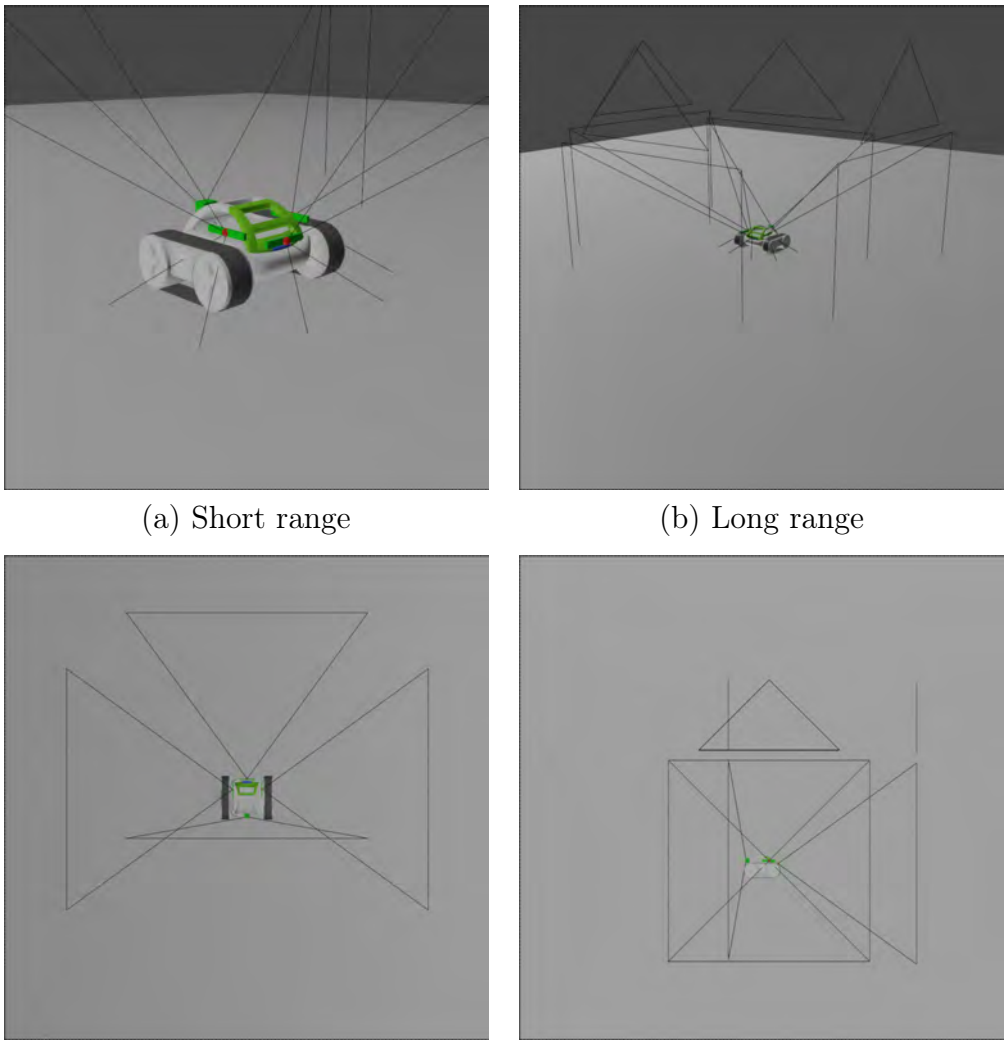
1. 1 *Raspberry Pi Camera Module (G)* and 3 *DFRobot SEN0286*
   Estimated cost = 92€64
2. 1 *Raspberry Pi Camera Module (G)*, 1 *Waveshare IMX335 5MP USB Plug-and-Play Camera* and 2 *DFRobot SEN0286*
   Estimated cost = 113€97
3. 1 *Raspberry Pi Camera Module (G)* and 2 *Waveshare IMX335 5MP USB Plug-and-Play Camera*
   Estimated cost = 113€42

1.1 Selection of cameras 1

This selection contains 1 *Raspberry Pi Camera Module (G)* and 3 *DFRobot SEN0286*, and is estimated to cost 92€64.

1.1.1 Symmetric placement of cameras

The design can be seen in Figure A.1.



(a) Short range (b) Long range

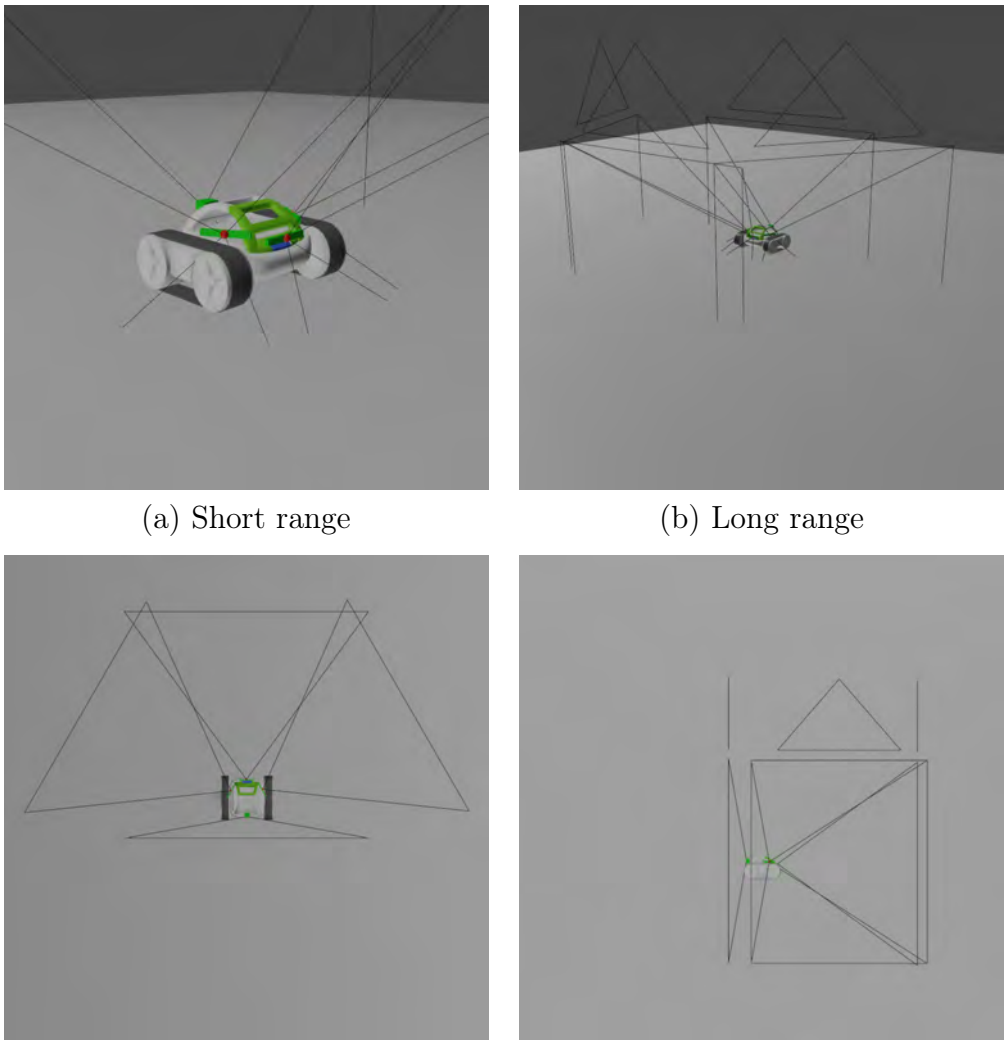
(c) From above (d) From the right

Figure A.1: Selection of cameras 1 - Symmetric placement of cameras

This is a symmetric design, where the left and right cameras are in the perpendicular direction of the *Sphero RVR* robot. The *Raspberry Pi Camera Module (G)* is placed in the back, to allow the rest of the module to focus on the front part. However, this placement implies dead angles on the front right and front left of the robot. This could be a problem since, even if the vision module should have a 360-degree view, the region of interest is most of the time in front of the vehicle.

1.1.2 Asymmetric placement of cameras

The design can be seen on Figure A.2.



(a) Short range (b) Long range

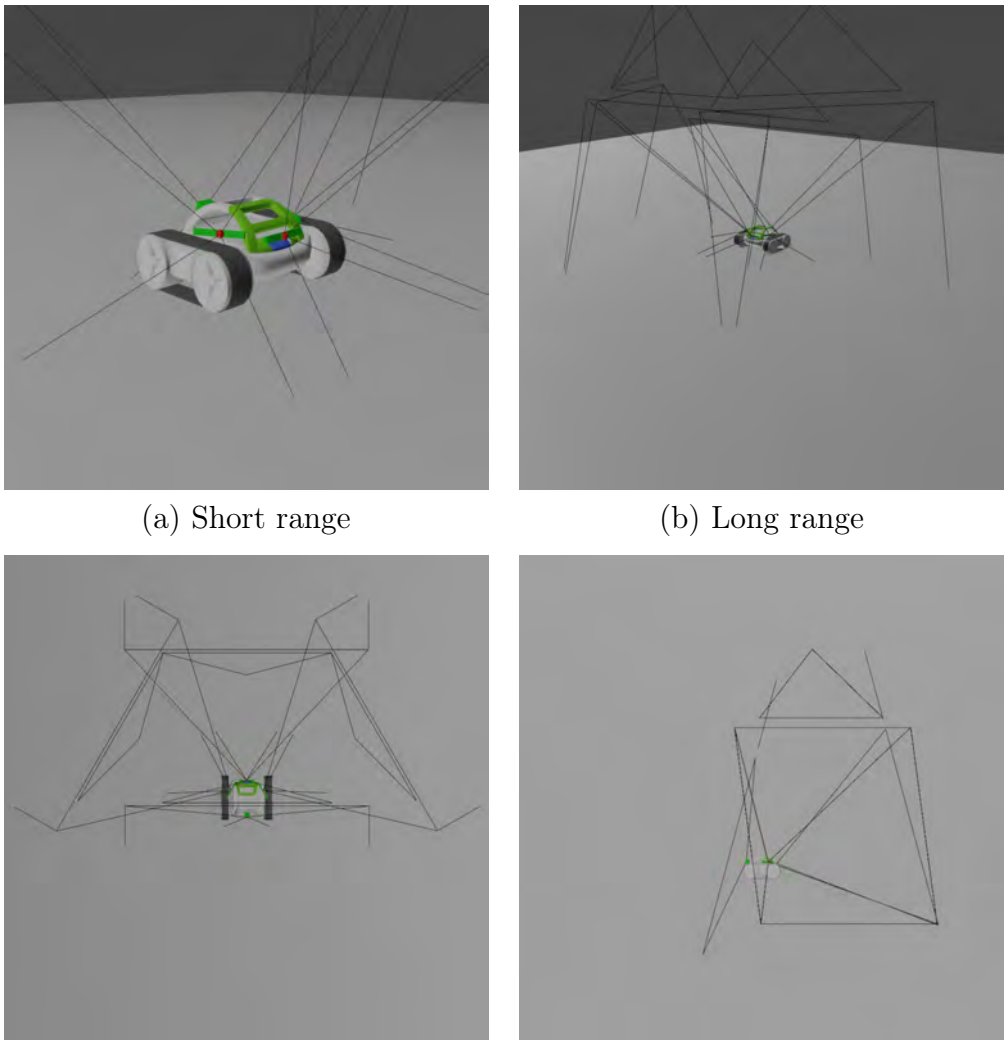
(c) From above (d) From the right

Figure A.2: Selection of cameras 1 - Asymmetric placement of cameras

Compared to the symmetric design, the region of interest is way more covered, since the left and right cameras are rotated a little bit more to the front. However, this implies to have some dead angles in the back of the robot.

1.1.3 Asymmetric placement of cameras oriented upwards

The design can be seen on Figure A.3.



(a) Short range (b) Long range

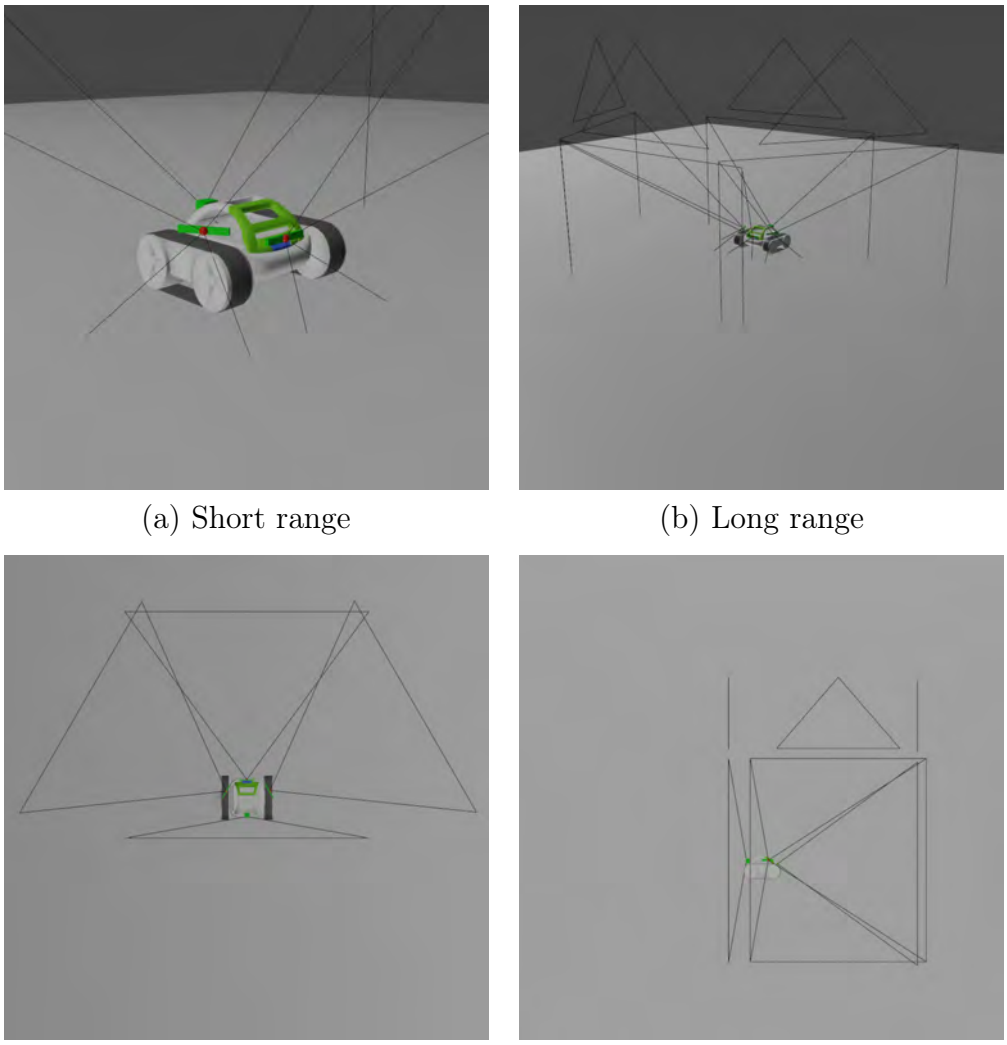
(c) From above (d) From the right

Figure A.3: Selection of cameras 1 - Asymmetric placement of cameras oriented upwards

Compared to a basic asymmetric design, where the cameras are placed oriented to the front of the Sphero RVR robot, the cameras are a little bit oriented to the top. As it could have been seen on Figure A.2-d, the cameras have a part of the image always seeing too close. This design allows to see more upwards than before, allowing to get features that are not at the ground level.

1.1.4 Asymmetric placement of cameras with bigger offset from the center

The design can be seen on Figure A.4.



(a) Short range (b) Long range

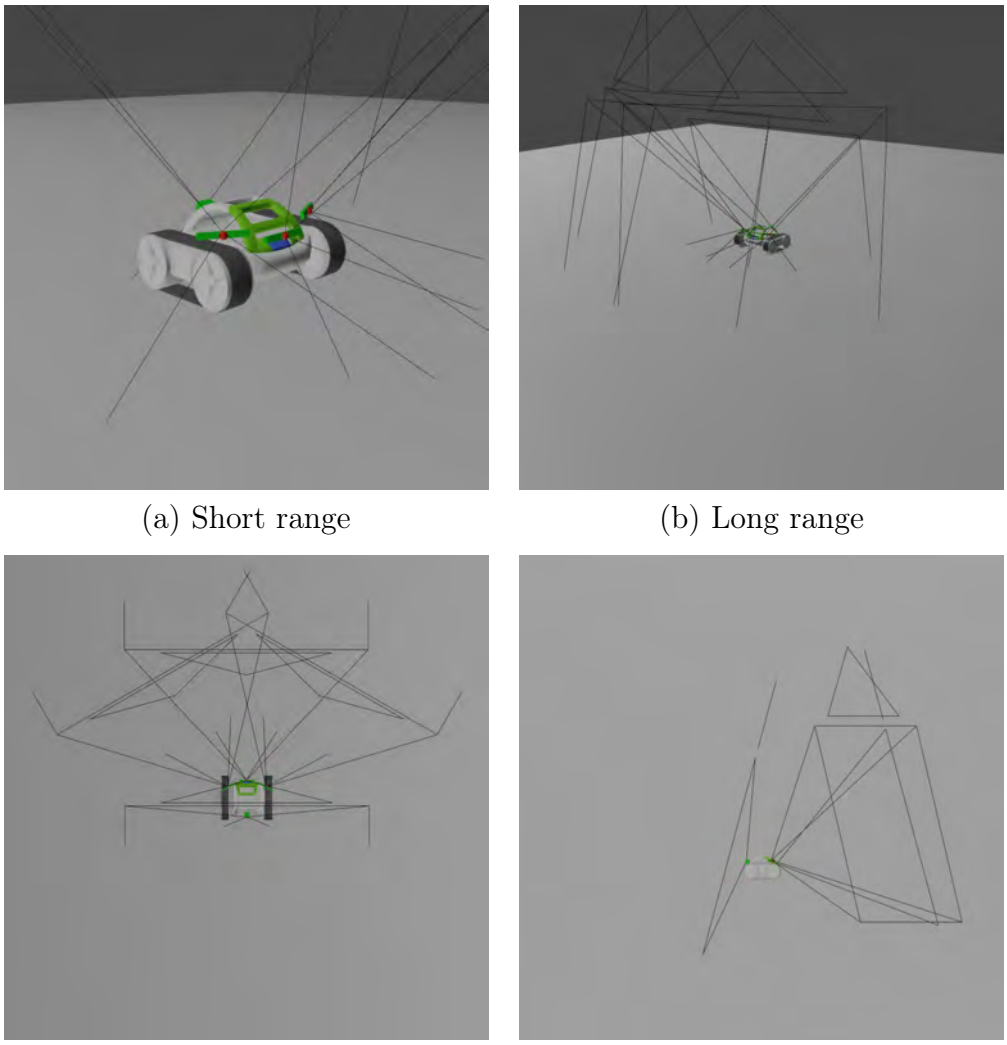
(c) From above (d) From the right

Figure A.4: Selection of cameras 1 - Asymmetric placement of cameras with bigger offset from the center

In comparison with the basic asymmetric design, the side cameras are a little shifted to the outside of the car, by being a little bit more on the wheels than in the design described by Figure A.2. One of the problems of this previous design was that since the camera was not on the edge of the car, it was keeping useless information in the images captured. Indeed, the side cameras are always seeing part of the wheels. In order to avoid that, a solution is to move the camera more on the wheels. However, this implies bigger dead angles on the very front of the robot, since the cameras are more off-centre, as it can be seen by comparing Figure A.2-c and Figure A.4-c. Moreover, it also implies bigger dead angles in the back of the car, for the same reason. Also, since the camera are above the wheels, a special support need to be designed.

1.1.5 Asymmetric placement of cameras with bigger offset from the centre oriented upwards

The design can be seen on Figure A.5.



(a) Short range (b) Long range

(c) From above (d) From the right

Figure A.5: Selection of cameras 1 - Asymmetric placement of cameras with bigger offset from the centre oriented upwards

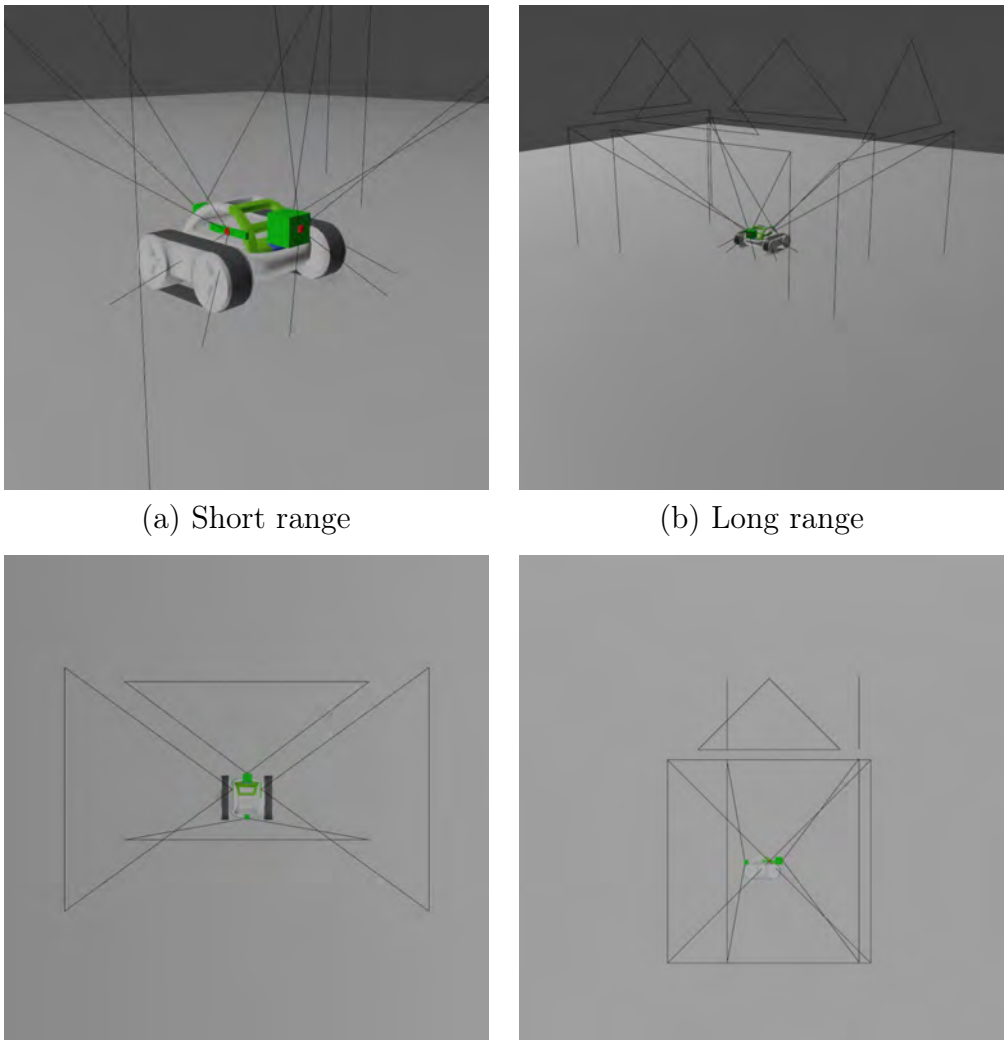
This design tries to take into account both problems stated by the design described in (c) and in (d). However, compared to the design (d), corresponding to the same design without the cameras being oriented upwards, the offset required is lower.

1.2 Selection of cameras 2

This design contains 1 *Raspberry Pi Camera Module (G)*, 1 *Waveshare IMX335 5MP USB Plug-and-Play Camera* and 2 *DFRobot SEN0286*, and is estimated to cost 113€97.

1.2.1 Symmetric placement of cameras

The design can be seen in Figure A.6.



(a) Short range (b) Long range

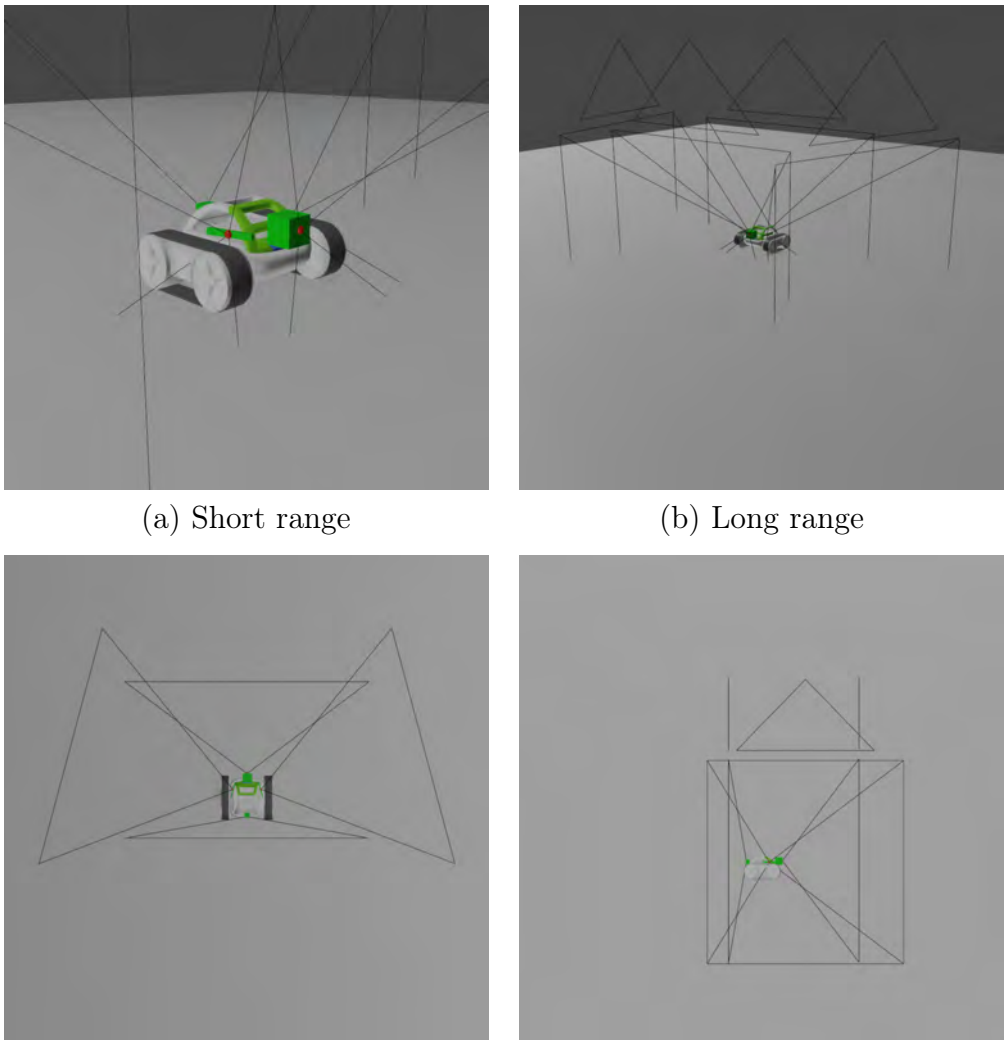
(c) From above (d) From the right

Figure A.6: Selection of cameras 2 - Symmetric placement of cameras

This is a symmetric design, similar to what was described in the first selection of cameras in (1-a), but the front camera has a wider view. This allows to have smaller dead angles on the front. However, this placement still implies dead angles on the front right and front left of the robot, even if they are much smaller. This could still be a problem since, the region of interest is mostly in front of the vehicle.

1.2.2 Asymmetric placement of cameras

The design can be seen in Figure A.7.



(a) Short range (b) Long range

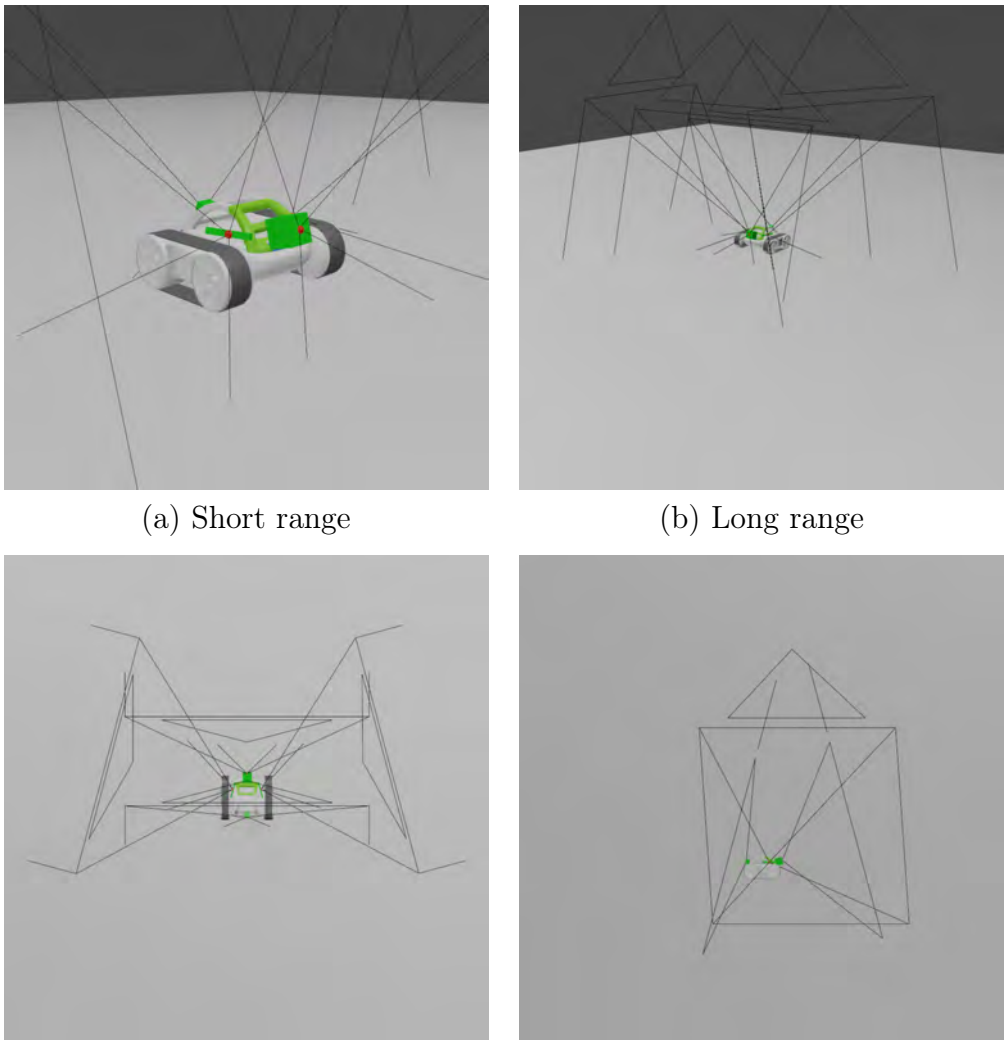
(c) From above (d) From the right

Figure A.7: Selection of cameras 2 - Asymmetric placement of cameras

This is an asymmetric design, similar to what is described in (1-b), where the front camera has still a wider view. However, the left and right cameras do not need to be as turned as before, since the front camera has a larger field of view. Hence, the dead angles implied by the similar design with the first selection of cameras, described in (1-b), are larger than the design shown on Figure A.7. Indeed, the side cameras can cover more angles in the back of the robot.

1.2.3 Asymmetric placement of cameras oriented upwards

The design can be seen in Figure A.8.



(a) Short range (b) Long range

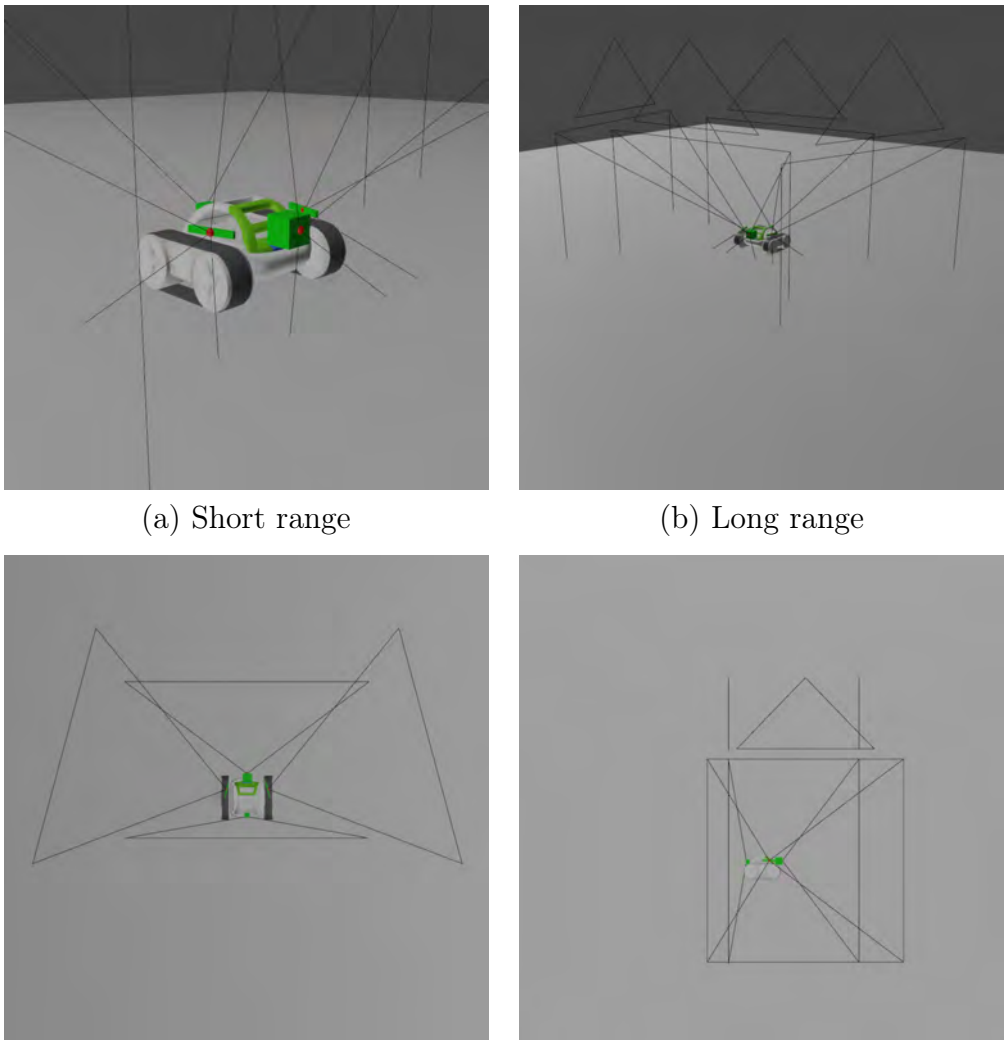
(c) From above (d) From the right

Figure A.8: Selection of cameras 2 - Asymmetric placement of cameras oriented upwards

This is an asymmetric design, similar to what is described in (b), but the cameras are oriented more upwards. The idea is the same as the one described by the design in (1-c). This way, the cameras can have more useful information.

1.2.4 Asymmetric placement of cameras with bigger offset from the centre - version 1

The design can be seen in Figure A.9.



(a) Short range (b) Long range

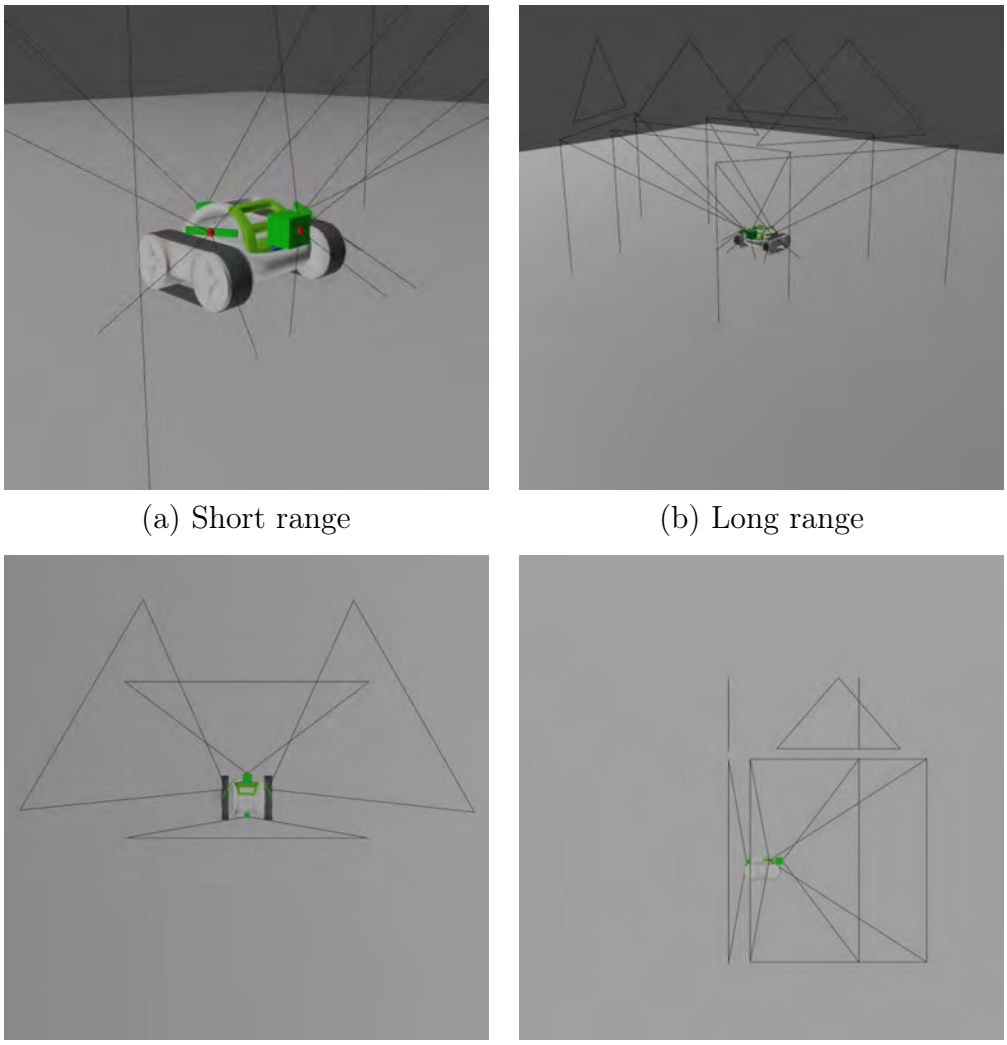
(c) From above (d) From the right

Figure A.9: Selection of cameras 2 - Asymmetric placement of cameras with bigger offset from the centre - version 1

This design of vision module is conceived the same way as the one described in (1-d). The idea is also to avoid seeing the wheels, since it doesn't give any additional useful information. However, as it can be seen on Figure A.9-c, compared to Figure A.7-c, the dead angles on a very close range are increased in the front of the robot. The back of the *Sphero RVR* robot has also bigger dead angles.

1.2.5 Asymmetric placement of cameras with bigger offset from the centre - version 2

The design can be seen in Figure A.10.



(a) Short range (b) Long range

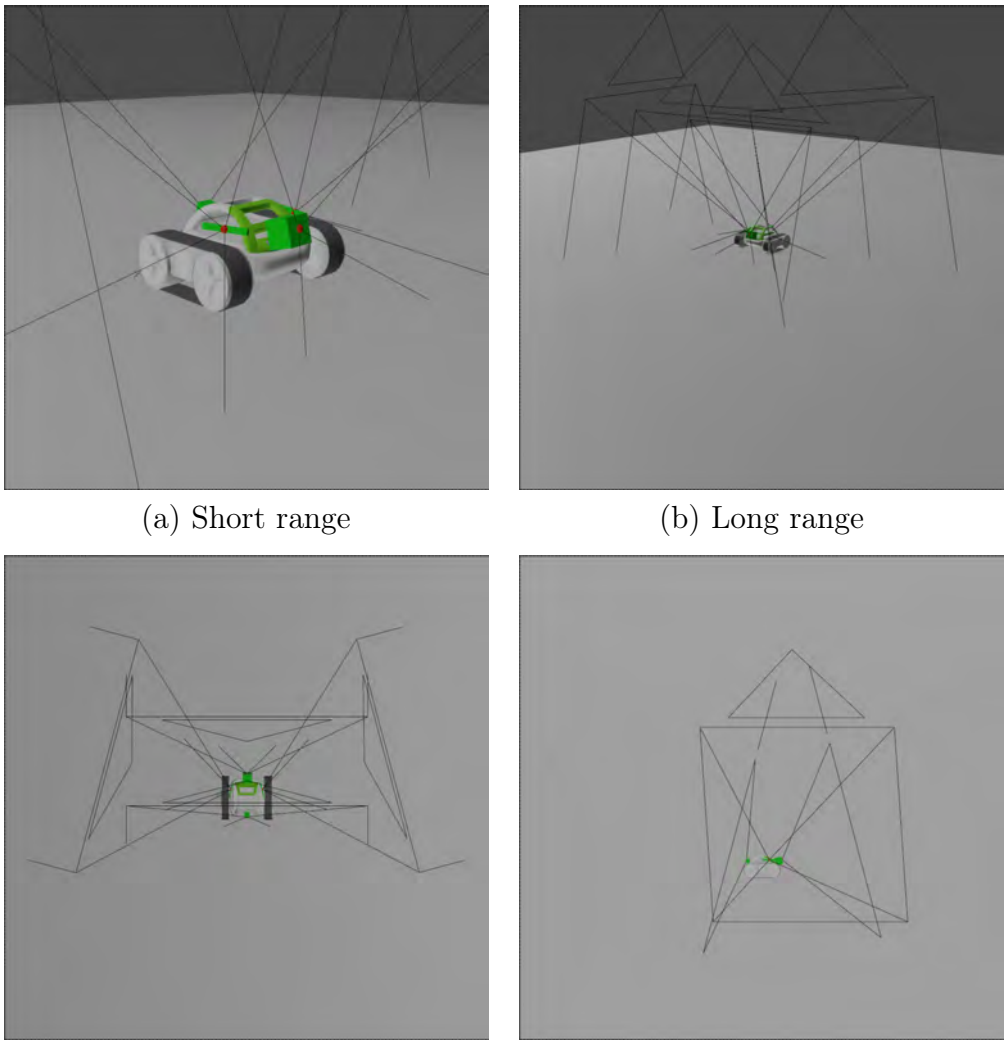
(c) From above (d) From the right

Figure A.10: Selection of cameras 2 - Asymmetric placement of cameras with bigger offset from the centre - version 2

This design is nearly the same as the one in (d). However, the side cameras are even more turned to the front. This way, the dead angles that are appearing because of the offset of the cameras are decreased. This unfortunately also means that the dead angles on the back are increased, but it is less critical since the region of interest is mostly in the front of the car.

1.2.6 Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 1

The configuration can be seen in Figure A.11.



(a) Short range (b) Long range

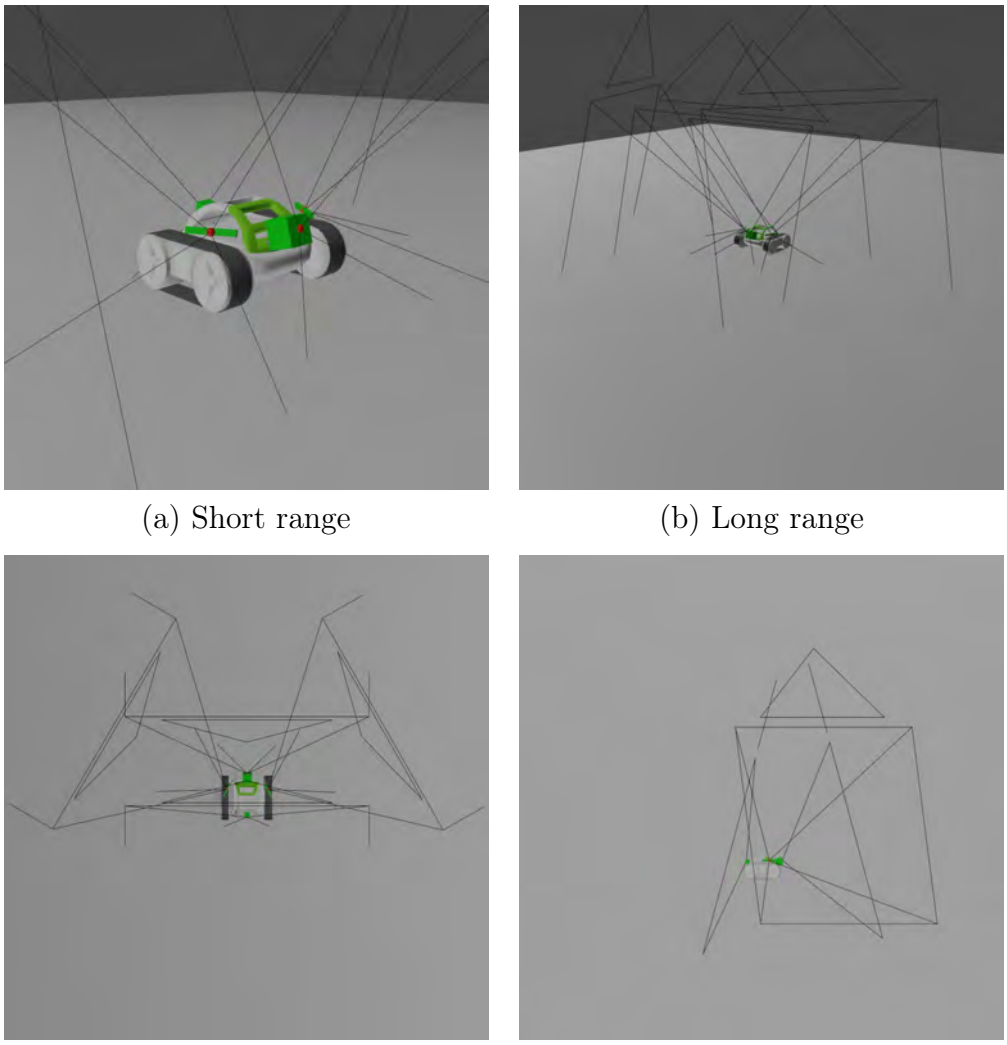
(c) From above (d) From the right

Figure A.11: Selection of cameras 2 - Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 1

This design has the same idea as what is described in (1-e). This takes into account a bigger offset and the cameras oriented upwards. Since the cameras look at a higher point than they could do with the design described by (d), the offset required from the centre is decreased.

1.2.7 Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 2

The design can be seen in Figure A.12.



(a) Short range (b) Long range

(c) From above (d) From the right

Figure A.12: Selection of cameras 2 - Asymmetric placement of cameras with bigger offset from the centre oriented upwards - version 2

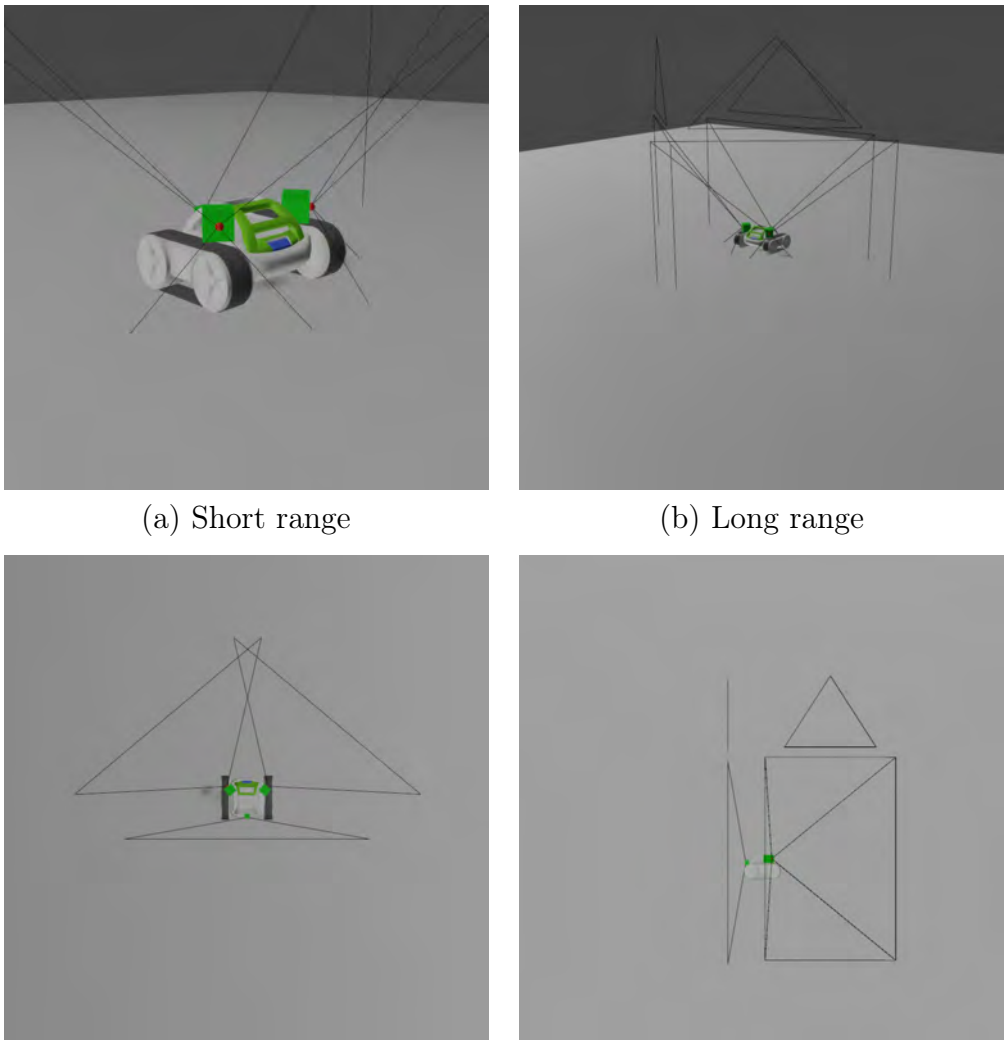
This design is nearly the same as the previous one described in (f), but it includes the idea of the design in (e), which is to reduce the dead angles in the front of the robot. This way, the vision module is able to have a better vision in the front, while having a higher view than before.

1.3 Selection of cameras 3

This design contains 1 *Raspberry Pi Camera Module (G)* and 2 *Waveshare IMX335 5MP USB Plug-and-Play Camera*, and is estimated to cost 113€42.

1.3.1 Symmetric placement of cameras

The design can be seen in Figure A.13.



(a) Short range (b) Long range

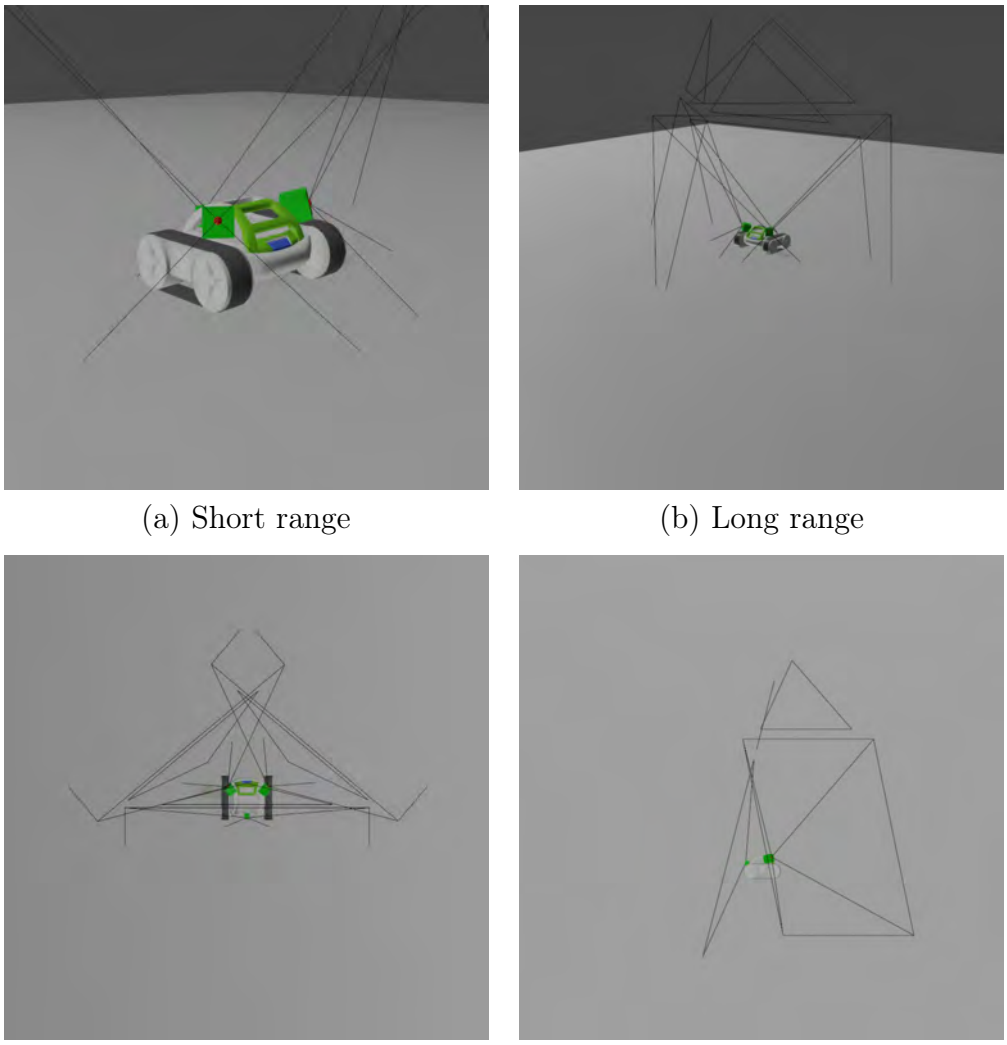
(c) From above (d) From the right

Figure A.13: Selection of cameras 3 - Symmetric placement of cameras

This design aims at having a very wide field of view with only 3 cameras. The *Raspberry Pi Camera Module (G)* is placed in the back, to allow the rest of the module to focus on the front part. The 2 other cameras are placed in a way to overlap their vision on the front part of the car. This way, the very front part of the robot is covered twice by both cameras. However, since it is on the edge of both cameras, these parts of the images have more distortions. Moreover, there are dead angles in the back of the car.

1.3.2 Symmetric placement of cameras oriented upwards

The design can be seen in Figure A.14.



(a) Short range (b) Long range

(c) From above (d) From the right

Figure A.14: Selection of cameras 3 - Symmetric placement of cameras oriented upwards

The idea of this design is the same as what is described in (1-c) and in (2-c). The goal is to avoid seeing the ground directly. It is indeed very unlikely to have an interesting object directly next to the wheels, and it is preferred to have a wider view of interest by looking a little bit more upwards.

Appendix B

LEDs clustering algorithm

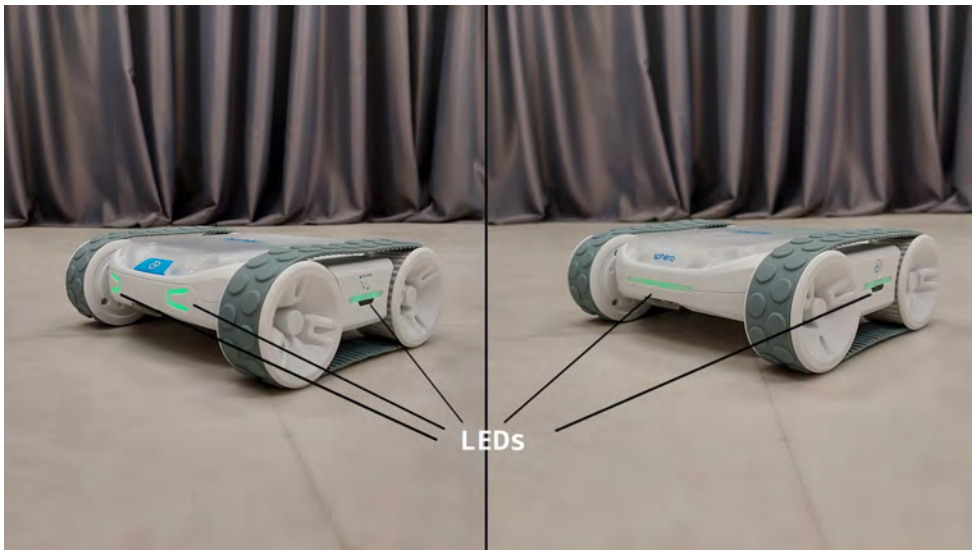


Figure B.1: LEDs on a *Sphero RVR* robot

The following algorithm is used in order to cluster the segmented LEDs. The term *eye* identifies the LEDs on the front face of the *Sphero RVR* robot, while the term *line* describes the LEDS on the side and back faces of the *Sphero RVR* robot. The LEDs are shown on Figure B.1. The LEDs clustering algorithm is described just below.

1. Choose segmented areas below a given height, since robots should be on the ground.
2. Identify if the segmented LEDs correspond to left eyes, right eyes or lines.
   (a) Compute aspect ratio of the segmented areas, which corresponds to the ratio between the width and the height.
   (b) Eyes have an aspect ratio closer to 1 than lines. It is hence possible to differentiate eyes and lines using a given threshold.
   (c) Then, the left and right eyes need to be identified.
      i. Divide the eye between its left and right parts.
      ii. Compute the area of both parts.

iii. If the left area is higher, it means that it is a right eye on the robot. Otherwise, it is a left eye.

3. Once the LEDs are identified, the matching pair of eyes can be found.

   (a) Find the closest left eye on the right of each right eye. If the matched eyes are too far from each other, then they need to be considered as isolated.

4. Clusters that include both a pair of eyes and a line can also be identified.

   (a) Compute the direction of the RVR containing a pair of eye, based on the area of each eye. If the left eye is wider than the right one, it means that that the robot is looking to the right, which means that the only possibility to add a line to the cluster is on the left of the image. Otherwise, one should only check on the right of the image.

   (b) Check in that direction if there is a line close enough and add it to the cluster.

5. Lastly, clusters of lines only can be found.

   (a) Compute the distance between pair of lines

   (b) If they are close enough, add them as a cluster.

Appendix C

Rasperry Pi setup

In order to use *ROS* for the RVR, a *Raspberry Pi* is used with *Ubuntu*. For this master thesis, *Ubuntu Server 20.04.4 LTS* is used.

Also, in order to use the cameras, they have to be enabled by the operational system. This can be done by adding the following lines in the `/boot/firmware/config.txt` file of the *Raspberry Pi*.

```
start_x=1
gpu_mem=256
```

The `gpu_mem` variable corresponds to an amount in megabytes reserved for the GPU. By default, it is set to $16mb$. However, the camera requires more memory, and that is why it has been increased. It could work with less memory, but using such high value do not bring any problems and gives adequate results.

Once it has been setup and connected, *ROS* can be installed. *ROS Noetic*[34] is chosen for this application. It is also used by the controller *Raspberry Pi*.

When working with *ROS*, a workspace need to be defined. A default name is given to it, according to the tutorials[49], which is `catkin_ws`.

The setup file for the current terminal used for running ROS also need to be sourced. In order to avoid the step of sourcing the file everytime, one could include the following command in the `~/.bashrc` file.

```
source /<path>/<to>/<workspace>/catkin_ws/devel/setup.bash
```

where of course, the `/<path>/<to>/<workspace>/` corresponds to the path to the workspace `catkin_ws`.

Appendix D

Communication with the controller *Raspberry Pi*

In order to communicate with the controller *Raspberry Pi*, an ethernet connection is used. The idea is to set the two *Raspberry Pi*s on the same network, in order to share the different topics. This is based on the procedure described by Yamamoto[50].

As said before, an ethernet cable is required to link the two *Raspberry Pi*s.

First, one should ensure that both computers have `openssh-server` and `net-tools` installed.

After that, the addresses of both *Raspberry Pi*s need to be set. Since they will be on the same network, and that they are alone, a simple address can be chosen for both of them. For the sake of this thesis, `10.10.10.1` is given to the controller computer, and `10.10.10.2` for the vision module one.

In order to do so, the file corresponding to the network configuration in the `/etc/netplan/` directory needs to be edited. In this thesis, it was named `50-cloud-init.yaml`.

```
---
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:
            dhcp4: no
            optional: true
            addresses:
                - 10.10.10.X/24
            gateway4: 10.10.10.1
            nameservers:
                addresses: [8.8.8.8, 1.1.1.1]
...
```

The `X` has to be replaced either by `1` for the file of the controller computer, or by `2` for the vision module one.

Once the changes are applied using the `sudo netplan apply`, the *Raspberry Pis* are able to communicate through the ethernet interface.

However, in order to allow the sharing of topics in a *ROS* environment, one should say who will be the master of the relation. In order to do that, one should explicitly say that the controller computer is the master. This can be done by running the following commands.

```
source /opt/ros/noetic/setup.bash
export ROS_MASTER_URI=http://10.10.10.1:11311
export ROS_HOSTNAME=10.10.10.X
```

As before, the `X` need to be replaced by either `1` for the controller computer, or `2` for the vision module.

In order to avoid doing it every time the *Raspberry Pis* boots, these commands can be added at the end of the `~/.bashrc` file.

This ends the configuration of the communication between the two computers. This allows any of the two computers to get access to any topic on the network.

Appendix E

Dataset creation protocol for robot recognition

5.1 Introduction

This dataset creation protocol has been designed in order to have a clear process to create a dataset for robot recognition, using supervised learning. This methodology can be extended to other robots than the one described here, which is the *Sphero RVR* robot.

The methodology is divided in 2 parts. First, the images need to be generated. After that, the labelling of the images is done.

5.2 Images generation

5.2.1 Configuration

5.2.1.1 Target robot and cameras used

The current configuration of the target robot and of the cameras is described below.

- Target robot: Sphero RVR
- Cameras:
  - DFRobot SEN0286
  - Raspberry Pi Camera Module (G)

For this protocol, **a high-resolution camera is also used**. This creates a universal dataset that is not dependent on the robot's camera.

The number of datasets created depends on the number of different cameras used. For instance here, the number of datasets would be 3.

5.2.1.2 Environments

Also, one should define the scope of environments in which the algorithm should operate. It can of course generalize outside the known environments, but the quality would not be

assessed by the testing dataset that would be defined once the whole dataset is determined. For this example, only one environment is considered.

- IRIDIA[17] Sphero RVR arena

5.2.1.3 Cameras placement

Lastly, one should also know where to place the cameras before starting. The placement of the cameras for this RVR robot is described using *Blender* on Figure E.1. A scene is represented on Figure E.2. The views of all the cameras are also described on the following figures (Figure E.3, Figure E.5, Figure E.4 and Figure E.6).

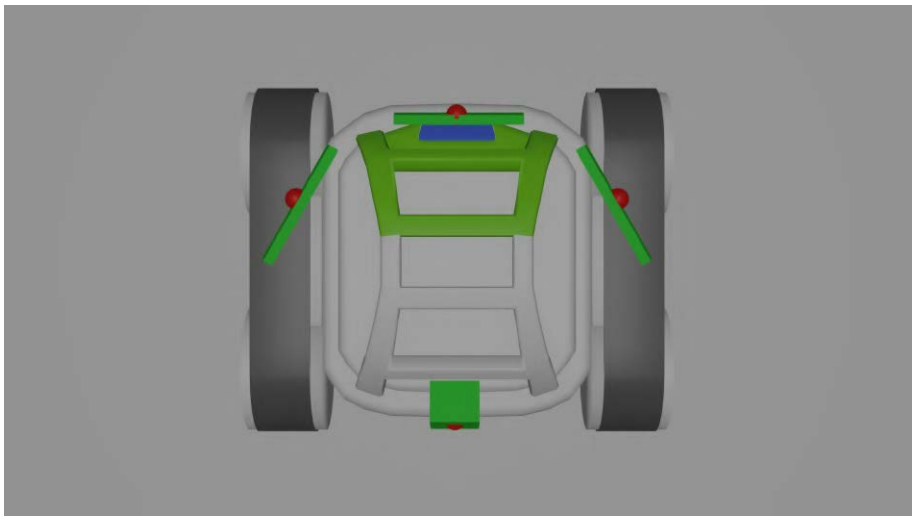


Figure E.1: Cameras

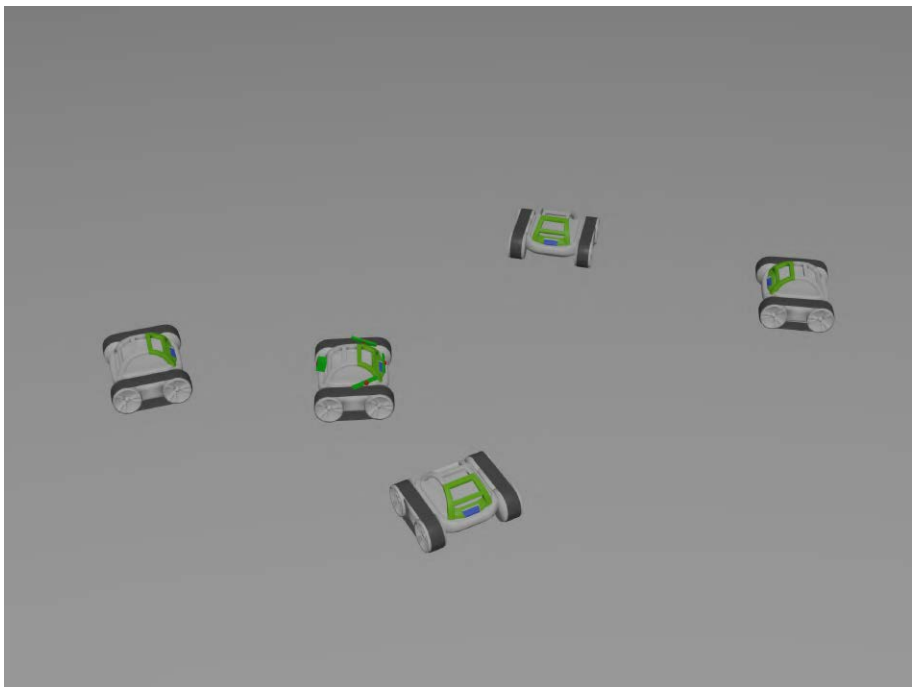


Figure E.2: Scene

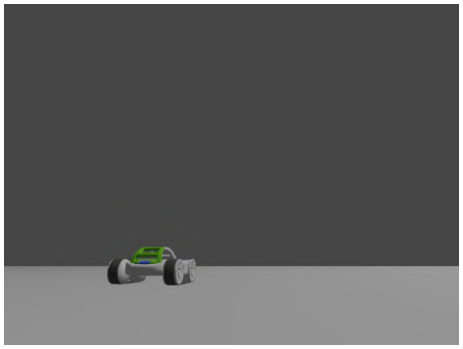


Figure E.3: Front camera view (DFRobot SEN0286)

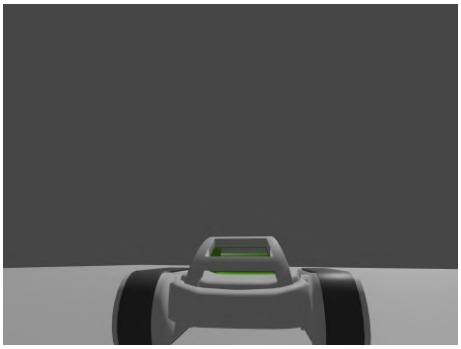


Figure E.4: Right camera view (DFRobot SEN0286)

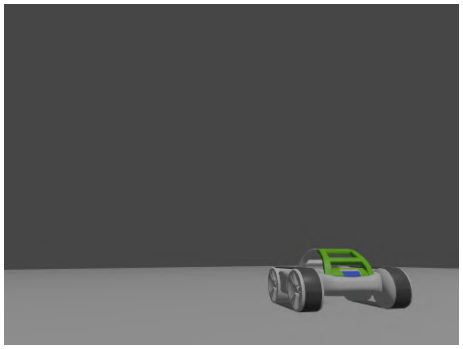


Figure E.5: Left camera view (DFRobot SEN0286)

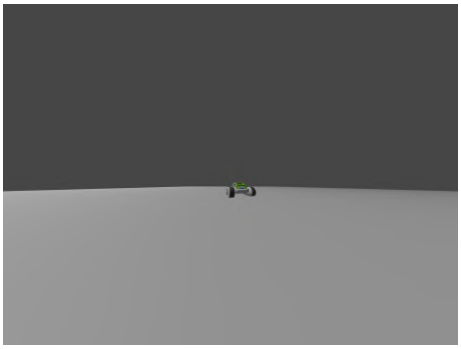


Figure E.6: Back camera view (Raspberry Pi Camera Module (G))

As it can be seen, the robots are always observed with an horizontal view. Moreover, the cameras are placed on the robot a little bit above the wheel, which corresponds to an height of $9cm$. These should be the same conditions as for the real cameras when creating the dataset.

5.2.1.4 Special requirements

When creating a dataset, one should also take into account requirements specific to its configuration and robot.
Swarm robotics
Since this protocol need to be generated in the context of swarm robotics, one could take into account the robots can have very specific configurations compared to situations for robot recognition in another context. Indeed, a very common behaviour for swarm of robots is to aggregate[37]. This is a particular behaviour that is not common to all situations outside the scope of swarm robotics. This needs to be taken into account by adding configurations where robots are aggregated, in order to allow a dataset including cases very specific to swarm robotics.

Sphero RVR
The Sphero RVR is using some LEDs, that can change colour. Since the algorithms using this dataset should be able to generalize to all RVRs, independently on the colour, one should ensure that it is taken into account, by turning them off or by using random

colours for the LEDs.

In this dataset, the LEDs are off.

5.2.2 Images capture procedure

The goal is to capture enough non-redundant images. In order to achieve this objective, the idea is, instead of taking pictures, recording videos of the robots in the desired environment. The procedure is described below.

- For each environment:
    1. Choose some spots inside the environment. The number depends on the size of the environment. The bigger the environment, the higher the number of spots.
    2. For each spot:
        - Repeat X times (X needs to be defined):
            (a) Choose a placement for the robots. The robots should be close enough to be able to be considered as relevant by the dataset. However, the dataset can challenge the algorithm, by placing sometimes the robots a little bit further. The number of robots can also vary.
            (b) For each camera:
                i. Take videos of length L around the spot. (L needs to be defined)
                ii. Keep videos without robots on the screen, in order to include images where there are no robot on the image.

In order to choose the right parameters, one should first know what is the aimed size for the dataset. According to Tatariants[46], the size can not be exactly known in advance. However, he suggests that the dataset should not be too small, arguing that for an object recognition with one class, 150 instances are enough for a dataset[46]. Nevertheless, it is quite small and we will prefer to aim for a more complete dataset, in order to avoid any issue of under-fitting. If the neural network that will train on this dataset completely overfits, one could simply reduce the number of instances taken from the dataset. In the litterature, it has been shown that in any case, the dataset should not be too small[39]. For example, Soekhoe, Putten, and Plaat have made an experience with 500 instances per class, and is suggesting it as a good number for object recognition using deep convolutional neural network. In order to have a complete dataset, the objective of 500 instances for the dataset is chosen.

Based on this first assumption, the other parameters can be chosen. Also, in an ideal situation, one should need to iterate the datasets once the neural network is trained, in order to find the perfect dataset size for the given application[11].

The number of spot depends on the size of the environment. It also depends of course, on the time allocated for the dataset creation, and the completeness of the dataset.
For this example, since the environment chosen is the IRIDIA[17] Sphero RVR arena, which is a square with a length of 3.00m, the number of spots is set to 5.

The number of times to iterate, which corresponds to the number of placements for the robots for a given spot, has to be determined by the parameters expressed just above. For this example, the number of times to iterate is set to 3 for every spot.

The length of the videos taken for every placement also need to be considered. It depends on the same parameters as described before. However, it should not be too redundant, in order to avoid overfitting of any model on the dataset. This means that it is not necessary to take the same video on the exact same spot with the exact same placement of robots. For this example, the length of the videos is estimated at 1 minute for every camera.

Lastly, one should also ensure that the special requirements are taken into account. Here, since the RVR can emit light in different colours, the robots used need to have a changing colour LED, in order to have images with RVRs having different colours for its LED.

Now that the procedure is set, and that the parameters are also, chosen, the time spent for the overall image generation procedure can be estimated in a theoretical way.

The parameters are listed here.

- Number of environments N: 1
- Number of spots S:
    1. For IRIDIA[17] Sphero RVR arena, S_1: 5
- Number of placements per spot P: 3
- Number of cameras C: 3
- Length of the videos L: 1 minute

This estimation takes into account a series of assumptions.

- The change between environment c_e is estimated at 10 minutes
- The change between spots c_s is estimated at 2 minutes
- The change of placement c_p for the robots is estimated at 2 minutes
- The change of camera is c_c estimated at 1 minute

Under these assumptions, one could compute the theoretical time T_{th} spent for the experiment.

$$T_{th} = N \times c_e + \sum_{i=1}^{N} S_i \times (c_s + P \times (c_p + C \times (c_c + L)))$$

For this example, the time spent expressed in minutes is

$$T_{th} = 140$$

This image generation, under these assumptions and with this configuration, should take theoretically 140 minutes.

5.2.3 Placement of robots

This step should be done before the experiment. It summarizes all the placements for the robots for every spot for every environment.

5.2.3.1 IRIDIA[17] Sphero RVR arena

The arena is represented by the Figure E.7.

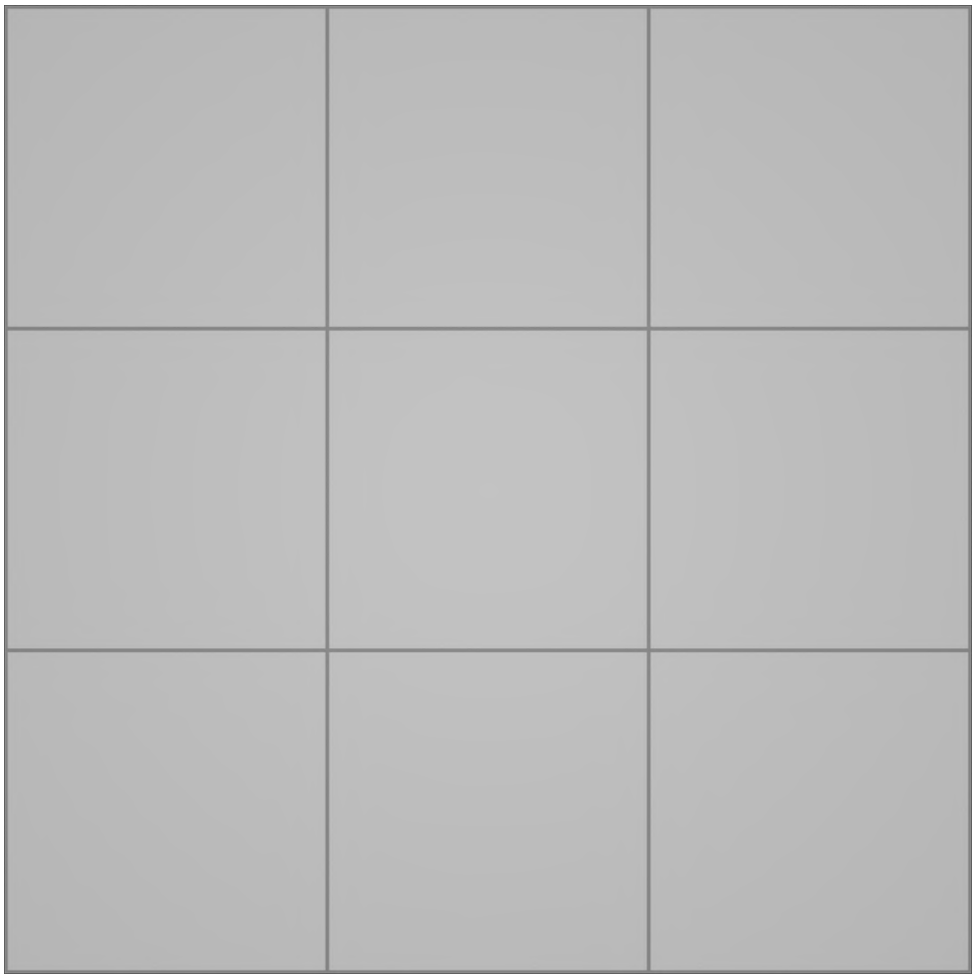


Figure E.7: IRIDIA[17] Sphero RVR arena representation

Then, there are 5 spots in this environment, and 3 placements per configuration, which means 15 different placements in the arena.

1. Spot 1:

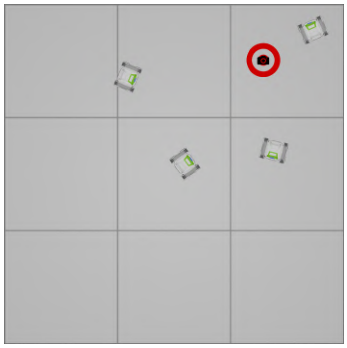


Figure E.8: Placement 1

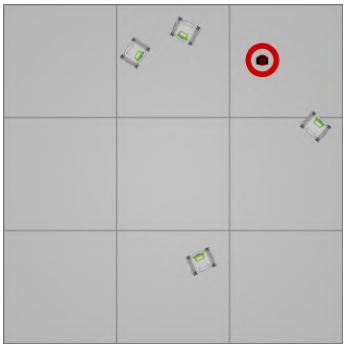


Figure E.9: Placement 2

Figure E.10: Placement 3

2. Spot 2:

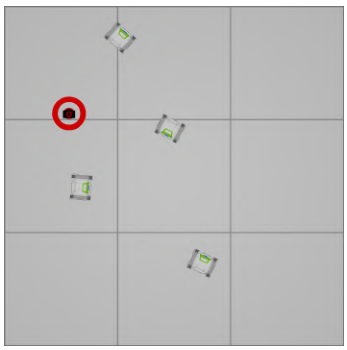


Figure E.11: Placement 1

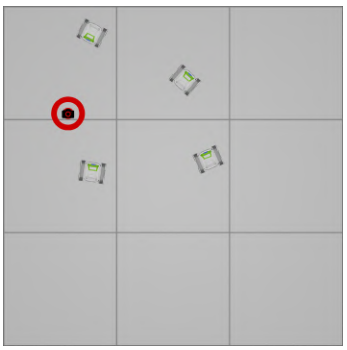


Figure E.12: Placement 2

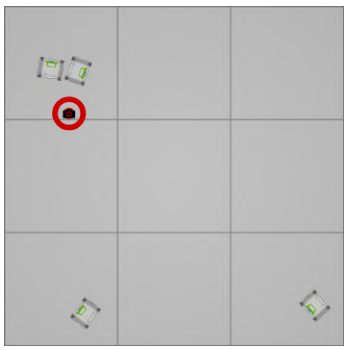


Figure E.13: Placement 3

3. Spot 3:

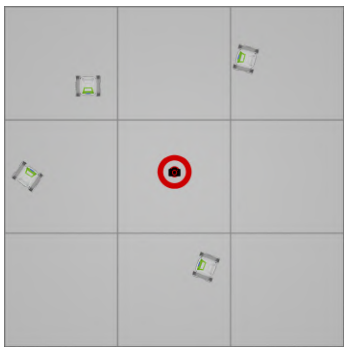


Figure E.14: Placement 1

Figure E.15: Placement 2

Figure E.16: Placement 3

4. Spot 4:

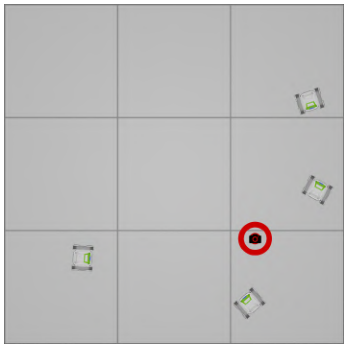


Figure E.17: Placement 1

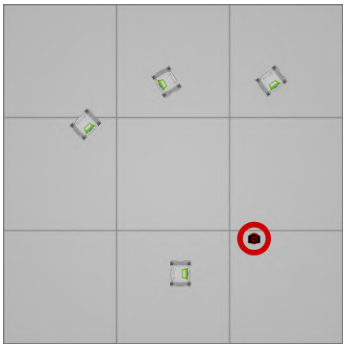


Figure E.18: Placement 2

Figure E.19: Placement 3

5. Spot 5:

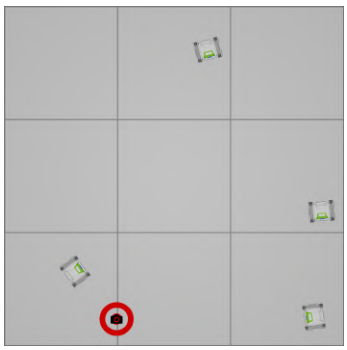


Figure E.20: Placement 1

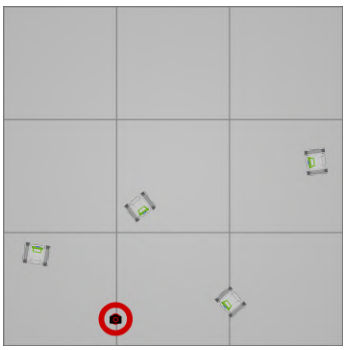


Figure E.21: Placement 2

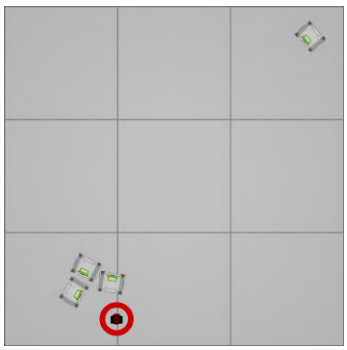


Figure E.22: Placement 3

As it can be noticed, the third placement for each spot takes into account that the robots can aggregate and can be far away from each other. This way, these corner cases, that are in fact quite common in swarm robotics, are considered.

Moreover, extra cases are considered where more than 2 robots are aggregating on Figure E.23, Figure E.24, Figure E.25 and Figure E.26.

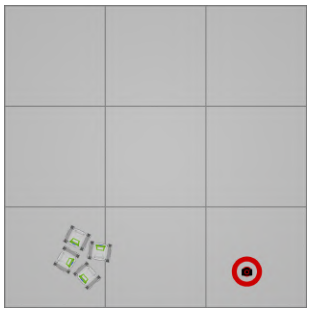


Figure E.23: Extra case 1

Figure E.24: Extra case 2

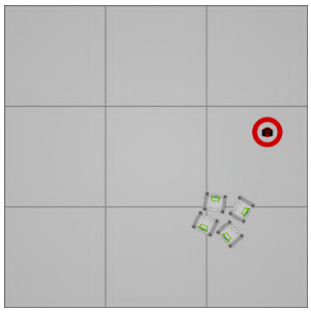


Figure E.25: Extra case 3

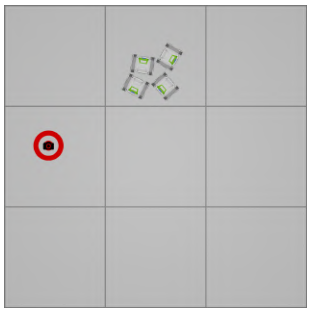


Figure E.26: Extra case 4

This, of course, adds some time to the image generation. Knowing that only the areas where the robots are aggregating is the relevant part in these configurations, only the time spent for moving the robots and the cameras is considered.

$$T_{extra} = 4 \times (c_s + c_p + c_c) = 20$$

This means that the total time for the image generation is increased by T_{extra}.

$$T_{total} = T_{th} + T_{extra} = 160$$

The total time for the image generation is estimated at 160 minutes.

5.2.4 Images from videos sampling

The videos are containing a lot of images, which are not all relevant. For example, 2 images in a row are very likely to have the nearly exact content. Hence, sampling too frequently could be a bad idea. Consequently, a rate of 2 images per second is a reasonable idea.

However, even by having a low frequence of sampling, some images will necessarily be superfluous. For example, images where there is no robot needs to be included, in order to leave cases where nothing need to be detected. However, it should not be the main part of the dataset. A filter need to be applied to remove the unnecessary images of the images generated.

With the current configuration of 1 minute per video, it means that each video generates 120 images. With the 19 different placements, it means a dataset of 2280 images, which is already very complete, according to what was discussed before about the dataset size. Indeed, the goal was 500 images for the dataset. However, one should take into account that redundant images where no robot appears need to be removed.

5.3 Image labelling

Now that the images are generated, the phase of image labelling can start. Of course, one should first ensure the validity of the images. Once this first filter is done, the labelling phase can start.

5.3.1 Labelling tool

There exists multiple tools able to efficiently label an image or a segmentation based on a stream of images.
Here is a non-exhaustive list of such tools.

- labelImg
- labelme
- CVAT
- Labelbox
- hasty
- Plainsight

All of these tools offers the possibility to create a dataset from images. It allows to easily label images and segment them.

5.3.2 Dataset extension

In order to make the dataset more robust and more complete, an idea is to artificially extend the dataset. this step is not mandatory, but it can help the generalization of the algorithm. This could include noisy, rotated images and shifted images. Another idea could be to include some distortion on the image. This way, an algorithm could more easily generalize its knowledge with other cameras.

5.4 Conclusion

To conclude, this methodology aims at helping people creating dataset for robot recognition, by describing a concrete example with the *Sphero RVR* robot.

Bibliography

[1] Sarah M. Ackerman et al. *The Swarmathon: An Autonomous Swarm Robotics Competition*. 2018. arXiv: 1805.08320 [cs.MA].

[2] André G. Araújo et al. "Integrating Arduino-Based Educational Mobile Robots in ROS". In: *Journal of Intelligent & Robotic Systems* 77 (2013), pp. 281–298.

[3] Michael Bonani et al. "The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research". In: (Jan. 2010).

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

[5] Mingxuan Chen et al. "A multichannel human-swarm robot interaction system in augmented reality". In: *Virtual Reality Intelligent Hardware* 2.6 (2020), pp. 518–533. ISSN: 2096-5796. DOI: https://doi.org/10.1016/j.vrih.2020.05.006.

[6] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org.

[7] Changquan Ding, Hang Liu, and Hengyu Li. "Stitching of depth and color images from multiple RGB-D sensors for extended field of view". In: *International Journal of Advanced Robotic Systems* 16.3 (May 2019), p. 172988141985166. DOI: 10.1177/1729881419851665. URL: https://doi.org/10.1177/1729881419851665.

[8] M. Dorigo and M. Birattari. "Swarm intelligence". In: *Scholarpedia* 2.9 (2007). revision #138640, p. 1462. DOI: 10.4249/scholarpedia.1462.

[9] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE Computational Intelligence Magazine* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[10] Marco Dorigo et al. "The SWARM-BOTS Project". In: *Swarm Robotics*. Ed. by Erol Şahin and William M. Spears. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 31–44. ISBN: 978-3-540-30552-1.

[11] Brad Dwyer. *Andrew Ng: Deploying to production means you're halfway there.* URL: https://blog.roboflow.com/andrew-ng-scale-transform-conference/. (accessed: 12.04.2022).

[12] Sergio Garrido-Jurado et al. "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming". In: *Pattern Recognition* 51 (Oct. 2015). DOI: 10.1016/j.patcog.2015.09.023.

[13] 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.

[14] *Gazebo*. URL: https://gazebosim.org/. (accessed: 27.01.2021).

[15] Alessandro Giusti et al. "Cooperative sensing and recognition by a swarm of mobile robots". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 551–558. DOI: 10.1109/IROS.2012.6385982.

[16] Ting-Kuei Hu et al. "VGAI: A Vision-Based Decentralized Controller Learning Framework for Robot Swarms". In: *Preprint* (2020). URL: https://arxiv.org/abs/2012.14906.

[17] IRIDIA. *IRIDIA*. URL: https://code.ulb.ac.be/lab/IRIDIA. (accessed: 27.01.2021).

[18] Simon Jones et al. *DOTS: An Open Testbed for Industrial Swarm Robotic Solutions*. 2022. DOI: 10.48550/ARXIV.2203.13809. URL: https://arxiv.org/abs/2203.13809.

[19] Zahi Kakish, Sritanay Vedartham, and Spring Berman. *Towards Decentralized Human-Swarm Interaction by Means of Sequential Hand Gesture Recognition*. 2021. DOI: 10.48550/ARXIV.2102.02439. URL: https://arxiv.org/abs/2102.02439.

[20] Mojtaba Karimi et al. "ReMoRo; A mobile robot platform based on distributed I/O modules for research and education". In: Oct. 2015, pp. 657–662. DOI: 10.1109/ICRoM.2015.7367861.

[21] Miquel Kegeleirs, David Garzón Ramos, and Mauro Birattari. "Random Walk Exploration for Swarm Mapping". In: *Towards Autonomous Robotic Systems*. Ed. by Kaspar Althoefer, Jelizaveta Konstantinova, and Ketao Zhang. Cham: Springer International Publishing, 2019, pp. 211–222. ISBN: 978-3-030-25332-5.

[22] Seung-Hun Kim, Chansung Jung, and Jaeheung Park. "Three-Dimensional Visualization System with Spatial Information for Navigation of Tele-Operated Robots". In: *Sensors* 19.3 (2019). ISSN: 1424-8220. DOI: 10.3390/s19030746. URL: https://www.mdpi.com/1424-8220/19/3/746.

[23] A. de La Bourdonnaye et al. "Practical Experience with Distance Measurement Based on Single Visual Camera". In: *Advances in Military Technology* 7.2 (2012), pp. 49–56.

[24] Yi Li et al. "Scale-aware Monocular SLAM Based on Convolutional Neural Network". In: Aug. 2018. DOI: 10.1109/ICInfA.2018.8812582.

[25] Alican Mertan, Damien Jade Duff, and Gozde Unal. "Single image depth estimation: An overview". In: *Digital Signal Processing* 123 (Apr. 2022), p. 103441. DOI: 10.1016/j.dsp.2022.103441. URL: https://doi.org/10.1016%5C%2Fj.dsp.2022.103441.

[26] Francesco Mondada et al. "The e-puck, a Robot Designed for Education in Engineering". In: vol. 1. 1. Portugal: IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65. URL: http://infoscience.epfl.ch/record/135236.

[27] Md. Abdul Mannan Mondal and Md.Haider Ali. "Performance Review of the Stereo Matching Algorithms". In: 2017.

[28] Nadia Nedjah and Luneque Silva Junior. "Review of methodologies and tasks in swarm robotics towards standardization". In: *Swarm and Evolutionary Computation* 50 (2019), p. 100565. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2019.100565. URL: https://www.sciencedirect.com/science/article/pii/S2210650217308398.

[29] Ajeet Ram Pathak, Manjusha Pandey, and Siddharth Rautaray. "Application of Deep Learning for Object Detection". In: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, pp. 1706–1717. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.144. URL: https://www.sciencedirect.com/science/article/pii/S1877050918308767.

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

[31] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[32] Generation Robots. *Sphero RVR Educational Mobile Robot*. URL: https://www.generationrobots.com/en/403503-sphero-rvr-educational-mobile-robot.html. (accessed: 12.01.2022).

[33] Francisco Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. "Speeded Up Detection of Squared Fiducial Markers". In: *Image and Vision Computing* 76 (June 2018). DOI: 10.1016/j.imavis.2018.05.004.

[34] ROS. *noetic/Installation/Ubuntu - ROS Wiki*. URL: http://wiki.ros.org/noetic/Installation/Ubuntu. (accessed: 14.03.2022).

[35] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. "Kilobot: A low cost scalable robot system for collective behaviors". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, May 2012. DOI: 10.1109/icra.2012.6224638. URL: https://doi.org/10.1109/icra.2012.6224638.

[36] Mateusz Sadowski. *360 camera for industrial inspection with ROS and Robosynthesis robots*. URL: https://msadowski.github.io/robosynthesis-3d-camera/. (accessed: 27.01.2021).

[37] Melanie Schranz et al. "Swarm Robotic Behaviors and Current Applications". In: *Frontiers in Robotics and AI* 7 (Apr. 2020). DOI: 10.3389/frobt.2020.00036. URL: https://doi.org/10.3389/frobt.2020.00036.

[38] Varuna De Silva, Jamie Roche, and Ahmet Kondoz. "Robust Fusion of LiDAR and Wide-Angle Camera Data for Autonomous Mobile Robots". In: *Sensors* 18.8 (Aug. 2018), p. 2730. DOI: 10.3390/s18082730. URL: https://doi.org/10.3390/s18082730.

[39] Deepak Soekhoe, Peter van der Putten, and Aske Plaat. "On the Impact of Data Set Size in Transfer Learning Using Deep Neural Networks". In: *Advances in Intelligent Data Analysis XV*. Ed. by Henrik Boström et al. Cham: Springer International Publishing, 2016, pp. 50–60. ISBN: 978-3-319-46349-0.

[40] Sphero. *Sphero Public SDK - Sphero Developer Site*. URL: https://sdk.sphero.com/. (accessed: 12.01.2022).

[41] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Noetic Ninjemys. May 23, 2020. URL: https://www.ros.org.

[42] Thomas Stützle. "ACO Algorithms for the Quadratic Assignment Problem". In: (July 2000).

[43] Austin Sumigray et al. "Improving Remote Environment Visualization through 360 6DoF Multi-sensor Fusion for VR Telerobotics". In: *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, Mar. 2021. DOI: 10.1145/3434074.3447198. URL: https://doi.org/10.1145/3434074.3447198.

[44] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: (2019). DOI: 10.48550/ARXIV.1911.09070. URL: https://arxiv.org/abs/1911.09070.

[45] Ying Tan and Zhong-yang Zheng. "Research Advance in Swarm Robotics". In: *Defence Technology* 239 (Mar. 2013). DOI: 10.1016/j.dt.2013.03.001.

[46] Maksym Tatariants. *Small Dataset-Based Object Detection: How Much Data is Enough?* Accessed: 12-04-2022. URL: https://mobidev.biz/wp-content/uploads/2021/09/small-dataset-based-object-detection_-how-much-data-is-enough_.pdf.

[47] Juan Carlos Trujillo et al. "Cooperative monocular-based SLAM for multi-UAV systems in GPS-denied environments". In: *Sensors* 18 (Apr. 2018), p. 1351. DOI: 10.3390/s18051351.

[48] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big Data* 3.1 (May 2016). DOI: 10.1186/s40537-016-0043-6. URL: https://doi.org/10.1186/s40537-016-0043-6.

[49] Brennan Yamamoto. *ROS network setup between two devices via ethernet cable*. URL: https://github.com/brennanyama/RobotOperatingSystem/wiki/ROS-network-setup-between-two-devices-via-ethernet-cable. (accessed: 21.03.2022).

[50] Brennan Yamamoto. *ROS network setup between two devices via ethernet cable*. URL: https://github.com/brennanyama/RobotOperatingSystem/wiki/ROS-network-setup-between-two-devices-via-ethernet-cable. (accessed: 21.03.2022).

[51] Yuchao Zhang et al. "Sambot II: A self-assembly modular swarm robot". In: *AIP Conference Proceedings* 1955.1 (2018), p. 040156. DOI: 10.1063/1.5033820. URL: https://aip.scitation.org/doi/abs/10.1063/1.5033820.