Enseirb-Matmeca

TMJI

Rapport Projet S9

Robot pour être vrai

Authors :
Nolwenn Gangloff
Léo Grelet
Quentin Harscoët
Florian Marguerit

Supervisors :
Myriam Desainte-Catherine
Jaime Chao
Jean-Michaël Celerier

Academic year 2016/2017

3 février 2017

Table des matières

Introduction

The project *"Robot pour être vrai"* is one of the different half-year projects which were proposed to the members of the TM option. This project, developed in collaboration with the Robotic option covers different fields : robotics, image processing, network, AI, etc...

This project aims to include Metabots in a artistic spectacle : *la Compagnie Lubat* wants the Metabots to dance on scene, controlled by a performer. It can be split into three main parts : the first one is more focused on image processing and aims to detect the Metabots. The second one is focused on software development and the control of the Metabots. Then, a mechanical and hardware part focuses of the Metabots.

We conducted the first two parts, whereas the other part was completed by a team in the Robotic option.

This report presents the work we accomplished. It is divided into four parts.

The first one is an overview of the project : we present the client, the goal, the technologies provided and the main structure.

The second part deals with the control application. We discuss the algorithms and the software.

In the third part, we talk about the detection application and explain how the image processing functions provide information for the control application.

Then, the last part revolves around the limits and encountered problems.

I Overview

This first section revolves around the details of the project : the client, the technologies et the structure.

I.1 The client : *La Compagnie Lubat*

The subject of this project was suggested by the members of *La Compagnie Lubat*, especially Jaime Chao who was the one who showed us the project and his demands.

La Compagnie Lubat is an artistic company which works on the improvisation. It uses Metabots during their shows. Metabots are small robots with four legs which can move around the scene. The company contacted us so that we add new behaviours to their robots. The company wants to have the robots to be able to move by themselves, or to be controlled, on the scene.

The aim of the project was the following : to control a fleet of Metabots in order to add them in their spectacles. The direction we chose is to detect the Metabots on the scene, compute the new positions and send the orders to make them move.

Thus, a video camera must be placed on the top of the scene, in order to take pictures of the scene. These pictures are used to compute the positions of the robots. These positions are sent to the application "MetaManager" which updates its database accordingly. Then the application computes the orders for the bots. These orders are sent thanks to the Xbee chip installed on each robot.

I.2 Technologies and equipment provided

In order to complete this task, *la Compagnie Lubat* gave us some tools and technologies.

We were able to use and to study some Metabots, to test their movements and their limits. A team of interns, which worked last year for the company, developed an application called "MetaManager". This application allows the person who uses it to take control of the Metabots and to send orders to them. We were given other applications, technologies and documentations such as a MAX/MSP portion of code, Iscore which is a multimedia interactive sequencer, and a database composed of robot movements. However, we only used the application MetaManager.

I.3 Project structure

The global project is divided into three big parts.

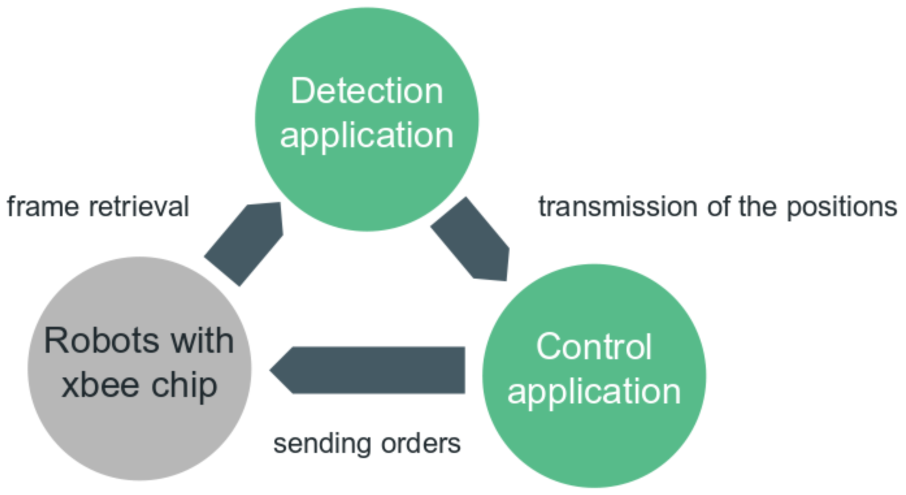

FIGURE 1 – The structure of the project

One part is focused on the Metabot detection using a video camera. Another one deals with the movement algorithm for the bots. And the last one is about the communication between MetaManager and the bots. Our TMJI team worked on the two first parts, and the Robotic team worked on the last one. However all these parts are linked together and are dependant on each other.

We divided our team in two sub-teams. Nolwenn and Léo worked on the image processing part, whereas Quentin and Florian worked on the application MetaManager and the robot behaviours.

II Controlling the Metabots

The MetaManager application already provided a first version of the Metabot control. Nevertheless, the app was recently developed and had some bugs. So the first weeks of work were dedicated to bugfixes. It was also a way to familiarise ourselves with JavaScript and to learn NodeJS, which neither of us knew well.

II.1 Objectives

The goal was to implement different fleet movement algorithms for the Metabots. The first which was implemented was BOIDS. Then another algorithm similar to BOIDS but with a leading bot for the fleet was developed. Other algorithms with a more random behaviour could have been easily implemented.

II.2 BOIDS algorithm

The BOIDS algorithm was invented by Craig Reynolds in 1986. The name come from "Bird-oid object", objects that behave like birds. This algorithm is very famous and has been already used in many fields such as Movies (Jurassic Park, the Lion King), video-games (Heavy rain), etc... It represents the movement of a fleet of birds. this algorithm works with three main forces.

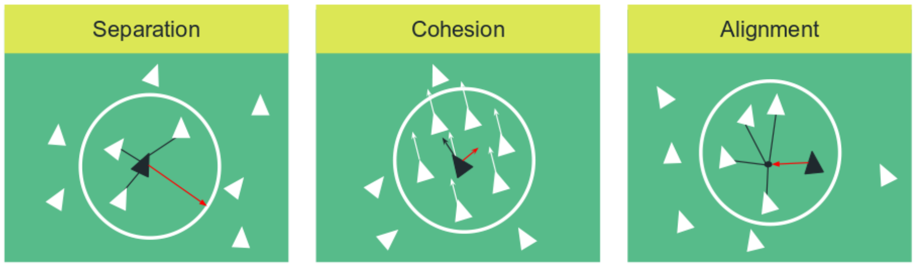

FIGURE 2 – The different forces

- A Alignement force : When a boid is too far away from the others, it comes closer to them.
- A cohesion force : When a boid is within a group of boids, it will adapt its speed and direction to the other boids next to him.
- A separation force : When a boid is to close from other boids, he will change his direction in order to escape from the group until it is in a reasonable distance from them.

Each force has its efficiency radius. It means that the influence of the others robots won't be considered if the robots are too far away or too close.

II.3 The Metamanager Application

The application Metamanager was created last year. It is developed with NodeJS using the javascript langage. It allows a user to have a basic control on the Metabots such as connecting them to the application and sending orders to them. But this application was not complete, indeed

the AI features was missing.
As we weren't able to have several Metabots at the same time, we worked only with simulations in the application itself to create and test our algorithm. The simulation allowed us to visualise how the Robots would perform with our algorithm with something else than raw numbers, thanks to a little 3D scene implemented with Three.JS.

II.4 Improving the Application

When we have received the application, it was still very buggy. The first weeks of work were dedicated to debugging the application. It was tough because Javascript and NodeJS were totally new for the team. The main problems remained in the fact that old structures were still used in the last version of the application. It generated big bugs due to a lot of undefined variables.

Once these problems solved, the team was working on AI behaviour. The application was meant to use AI, so a feature was created in this way. The class "Supervisor" in the class diagram was meant to contain the behaviour algorithm. We first implemented the basic BOIDS algorithm for the robots, then we have implemented a second algorithm based on the same behaviour than BOIDS but with a added feature. The feature was that the user can control one of the boids and lead the fleet with it.

The first algorithm was not exactly BOIDS since the BOIDS algorithm does not take into account any borders. Our algorithm is meant to be used on a scene, that's why we needed to modify the original algorithm to prevent the Metabots from falling of the scene.
So we had to add new forces to change the movement of the robots around the scene. This work was a bit complicated since we tried our best to keep the "natural motion" of the robots. As such, we had to make them smoothly return inside the borders of the scene when they were outside of it. Simply inverting their velocities would make them behave like a ball bouncing of a wall and that was not the intended behaviour.

The second algorithm is closely the same as the BOIDS algorithm, but it adds a new force. The new force is a specific attraction force which gather all the robots near the leader. The leader doesn't receive any order or force, the only way for him to move is an specific order from the user. Nevertheless its presence and position are still taken in account to compute the forces for the other robots.

We could have implemented other algorithms easily, but as soon as we finished the first two algorithms we started working on another part of the application : the communication between the image processing program and MetaManager.
This two different programs which are coded in different languages communicate with a tool from the library zeroMQ.

The operation of this application works in four different parts :

- First the application receive the data from the image processing program. it contains the new positions of the Metabots. The data file is a JSON format.
- The database of the position is then updated.

- the algorithm is launched, the new forces are computed for each bots.
- The news forces are sent the robots thanks to the Xbee chip.

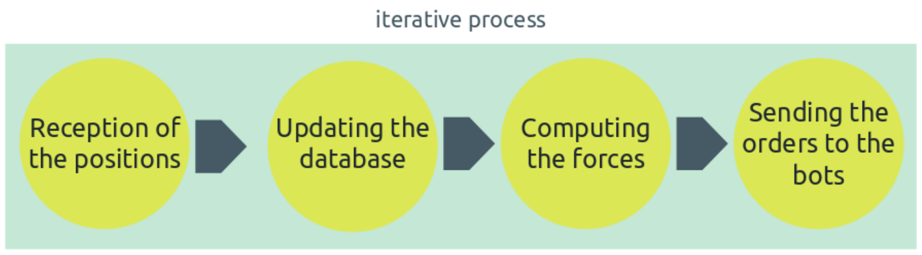

FIGURE 3 – The operating system of MetaManager

MetaManager Utilisation

In order to use the application, and run a simulation with the algorithm, you'll have to follow the following actions.

- First you have to install MetaManager, once you have done it, you run the application : npm start.
- Then add different Metabots in the "entities tab" (for example 3), you can after that edit the position of the Metabots on the scene because they have all the same position by default.
- Once you have generated your Metabots, you can generate a specific type of algorithm to control them. Switch to the "supervisor tab", name your supervisor and choose its type of algorithm. Save it, and select it in the drop down.
- Then you can switch to the "scene tab" and watch your square bots moving around a scene.

If you want to test the algorithms on real Metabots, you have to :

- launch the Bluetooth research.
- connect the bots to application and then link them to the entities you have created as before.
- run your algorithm, and watch your robots move.

III Detecting the Metabots

This third section talks about the detection application, which is needed to update the positions stored by the control application. Thus, we will first see the goals and principle of this application. Then, we will talk about the environment, with the technologies and the structure of the application. Lastly, an image processing section will present you the main algorithms used for detecting the Metabots.

III.1 Goals & principle

Depending on the characteristics of the floor material, there may be some errors between the predicted positions and the real ones, which will lead to unintentional collisions. Thus, a detection application is needed to ensure that the real positions are taken into account while sending new orders to the Metabots. This real-time detection allows the coordination of the fleet while avoiding collisions.

A first attempt was made by the previous developers to detect the Metabots. It used QRcodes stuck on top of the Metabots while a camera was positioned above the stage. This solution had some flaws : it required a high-resolution camera, a short distance between the camera and the Matabots, and a high luminosity. Furthermore, when there are many Metabots, for example more than 10, the time needed to get the QRcodes and compute the positions was too high. Lastly, from a esthetically point of view, this wasn't a viable solution. : *la Compagnie Lubat* didn't want the Metabots to have big black and white stickers.

Thus, we had to think about a new solution : we decided to use infrared technologies :

- **An infrared camera :** positioned above the stage, this camera retrieves the infrared frames of the scene. The intensity of the lights should not impact the infrared frame and the quality of the signal.
- **3 infrared LEDs on the Metabots :** positioned on the Metabots, these LEDs are aligned and are used to compute the position and the direction of the Metabots. Indeed, if we use one LED, we will not be able to know where the Metabot is going, we will only be able to know where it is located. Thus, the two closer LEDs indicate the head of the Metabot. The LEDs will not be visible to the spectators.

Here is a representation of the system :

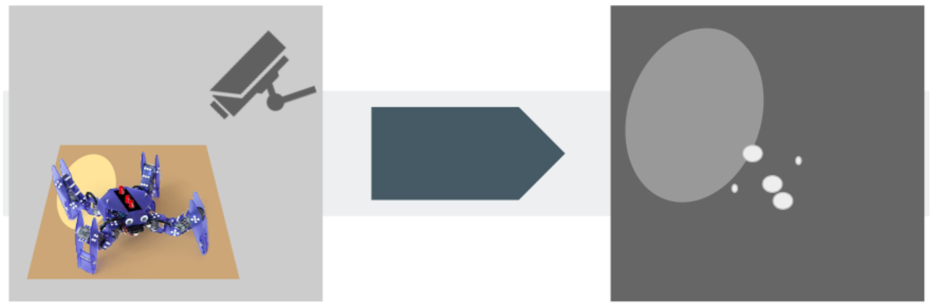

Figure 4 – The detection system with the infrared LEDs and the infrared camera

As we can see, the obtained frame may have some artifacts. These artifacts can be corrected through image processing, as we will see later.

Combined with the infrared technologies, we decided to use the language *C++* and the *OpenCV* library. This is a library of programming functions mainly aimed at real-time computer vision we used to process images in order to retrieve the Metabot positions.

FIGURE 5 – C++ language

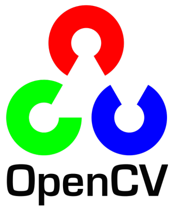
FIGURE 6 – OpenCV library

III.2 Algorithm

The algorithm of our detection application is divided into 2 parts. Here is a diagram of this algorithm.

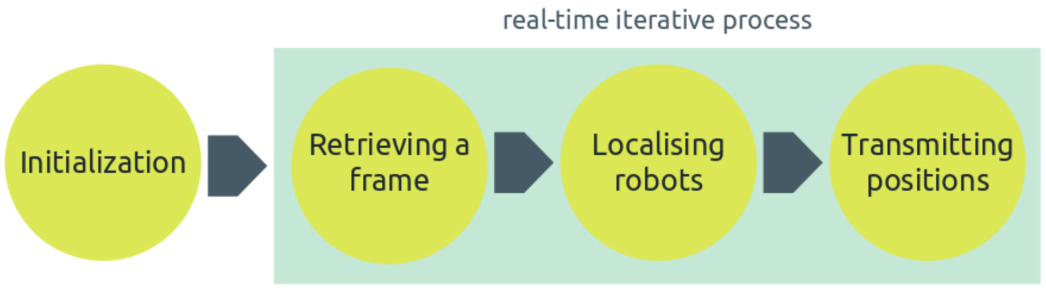

FIGURE 7 – The detection algorithm

- **Initialisation :** the first step is called only once during the application. In this part, we initialise the camera and the Metabots. The user is asked to turn on one Metabot at a time to let the camera detect it and get its position. The user then turns if off and turns on the next one. Once all the Metabots are detected, the initialisation part is done and the user can start the real-time detection.

- **Real-time iterative process :** This step can begin once the initialisation is done. This is a real-tile iterative process composed of 3 main functions : *Retrieving a frame*, *Localising robots* and *Transmitting positions*. In the first function, the camera retrieves a infrared frame. Then, thanks to image-processing functions, we can detect the LEDs on the frame. By comparing the Metabot positions on the previous frame with the LEDs positions, we can find which Metabot the LEDs belong to. Then, we can transmit the positions to the Control Application, formatting the data in JSON format and using a network library called *ZeroMQ*.

III.3 Image processing

The Detection Application mostly uses image processing functions. In this part we will describe these functions in detail.

III.3.1 Frame retrieval

The first function involves the infrared camera : indeed, in the first place we need to retrieve an infrared frame in order to determine where the robots are.

However, this image contains artifacts, due to spotlights lightening the scene, or even infrared light from the LEDs reflected on some parts of the robot.

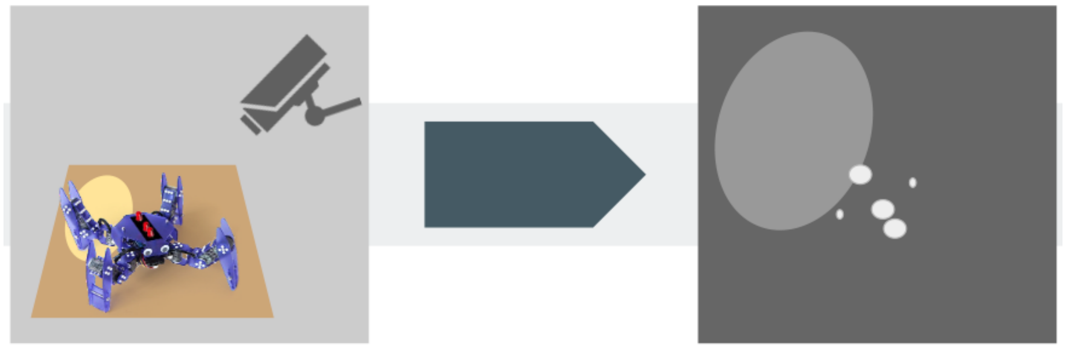

FIGURE 8 – Frame retrieval concept, with artifacts (a spotlight and infrared reflection on screws)

Thus, in order to obtain a usable image, some operations must be applied :

- **Thresholding :** At first, thresholding the image lead to a black and white picture. By choosing a threshold value high enough, we can erase a lot of the unwanted data, for example the spotlight. By this operation, we can't erase all the artifacts : their luminosity is too close to the LEDs one, and if we take a value high enough to erase these artifacts, we might erase the LEDs.
- **Eroding :** This second operation allows us to erode all the stains. Thanks to that, we can erase the artifacts we could not erase with a threshold operation. Indeed, these artifacts won't be as large as the LEDs. Because of that, after applying several erosion operations, the stains will disappear. We just must be careful not to erase the LED stains while doing these operations.
- **Dilation :** Lastly, this last part is used to dilate the stains to make them have the same size they had before the erosion.

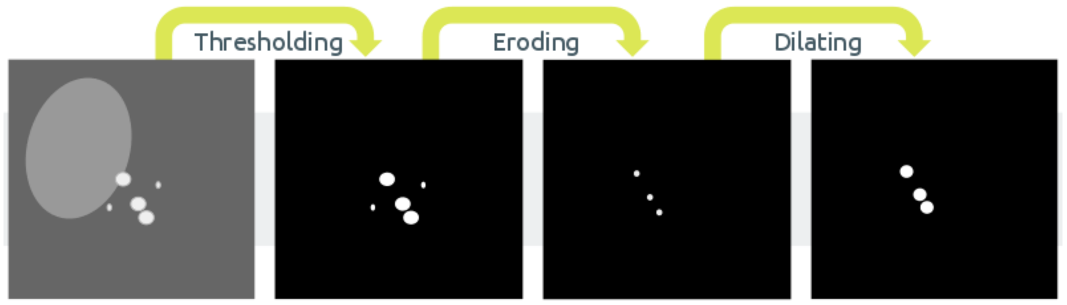

FIGURE 9 – Operations applied to retrieve a usable frame

III.3.2 Localising robots

Once the frame is usable, we can now detect the robots.

At first, we have to make groups of three LEDs to determine in which robot a LED belongs. This is done by getting all the positions of the LEDs and computing for each LED, which LEDs are closer to it.

Thanks to that, we can get temporary Metabots, with their position and direction. The last step consists in attributing these temporary Metabots to the real ones.

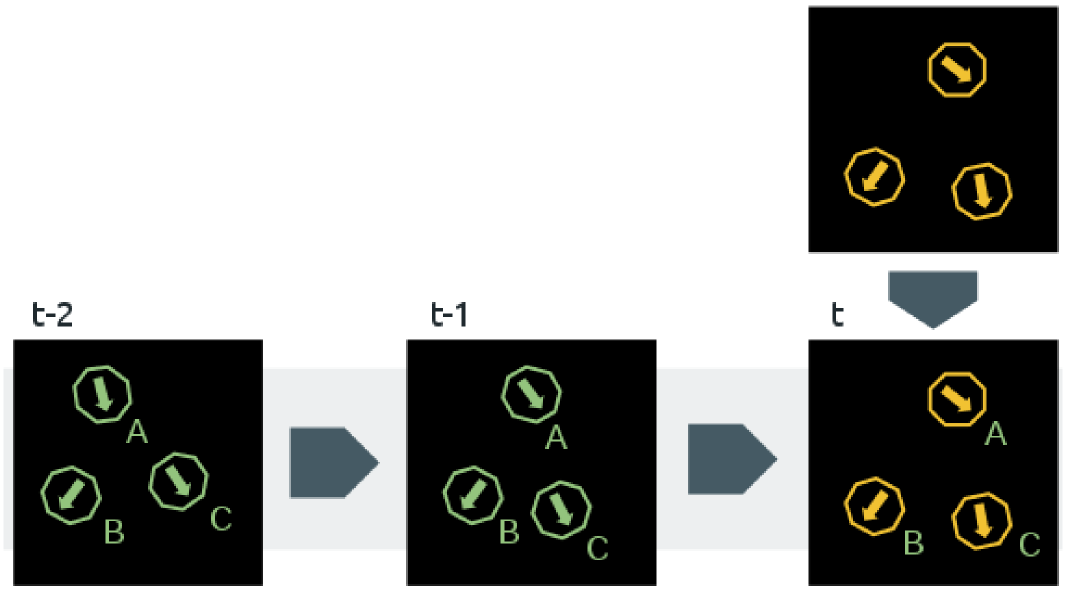

FIGURE 10 – Matching the temporary Metabots with the real ones

To attribute the temporary Metabot positions, we compute for each couple of temporary Metabot and real Metabot the distance between them. Then, we sort all these distances by an ascending order. While the number of Metabots we processed isn't equivalent to the number of Metabots, we take the smallest distance and update the Metabot position and angle with the value get thanks to the frame.

Once we update all the Metabots, we format the new positions in Json and send them to the control application.

III.4 User manual

This last part gives you some instructions in order to launch our application. At first, we will talk about the needed libraries to make the executable file, then we will see how to communicate with the applications.

III.4.1 Installation

Here are the dependencies of our application. Those libraries need to be installed before using the detection application.

- **Cmake :** CMake is an open-source, cross-platform family of tools designed to build, test and package software. These tools are necessary in order to install Opencv and the other libraries needed.
- **Packages v4l2ucp, v4l-utils and libv4l-dev** : These packages, which can be installed through *sudo apt-get install v4l2ucp v4l-utils libv4l-dev* are needed for the camera connexion. Once these packages are installed, if Opencv was already installed it must be recompiled.
- **Opencv :** OpenCV is an open source computer vision and machine learning software library we use to compute the Metabot positions. The version we used is **opencv-2.4.13**, available here `http://opencv.org/downloads.html`.
- **ZeroMQ :** ZeroMQ is the networking library we used to communicate with the control application. The version we used is the **4.2.0** version, which is available here : `http://zeromq.org/intro:get-the-software`. In addition to that library, we also used **cppzmq** available here `https://github.com/zeromq/cppzmq`.

III.4.2 Orders

The user can communicate to the application via the terminal. After having compiled the files, thanks to the command *make*, the application can be launched with :

```
./main
```

Then, the user must initialise the Metabots. It is done by entering : *init* and *bot*. Once the initialising part is launched, the user must turn on one Metabot at a time, and enter *ok* when the Metabot is ready. Once all the Metabots are initialised, he has to enter *ok* again and can now start the detection with the command *start*.

IV Limits and continuation

This last section deals with the encountered limits and the possible continuation regarding the project.

IV.1 Encountered problems

During the different phases of development, we faced different problems. Regarding the MetaManager, working on a complex application in a new language was difficult. It took us time to understand and memorise all the stuff we needed to know.

Another problem is the material. In fact, we needed some specific devices such as "infrared leds" and a "infrared camera". Jaime agreed with this idea, but he didn't manage to find the necessary funds to buy them. Therefore one part of our work was tested on premade files.

The last problem we faced is that the Robotic team didn't have the same schedules as us, so our work was totally desynchronised from theirs.

IV.2 Possible improvements

- **Controlling the Metabots :** About the application MetaManager, if we had have more time we would have implemented more algorithms, make the BOID algorithm more robust and more customisable. We also wanted to add the feature that allows the user to control one of the boids, in order to control the fleet.

  Due to the fact that our work the the robotic team's work weren't synchronised, we left the implementation of the ZigBee protocol.

- **Detecting the Metabots :** Regarding the detection application, some improvements could have been made. Indeed, the application isn't robust is all the situations : when a Metabot is hidden by an obstacle in a frame, our application may not correctly work. The attribution of the LEDs to the Metabots is not enough robust.

  Moreover, for now, the user can only see in the initialisation part how the retrieved frame will look like (after the erosion and dilation operations). But he can't change the degrees of erosion and dilation in this part. In order to change these values, he has to modify them in the code, and recompile the program. This is not really user-friendly.

  Lastly, our application should be tested with real data : an infrared camera and infrared LEDs positioned on robots.

- **Conceivable improvements** We could have implemented other features such as beat tracking, a visualisation of movement with Iscore, and a new set of movement for the bots.

V Conclusion

Thus we were able to see in this report what the subject and context were. We explained how the main structure of our project worked : thanks to two applications working together to control and detect the Metabots.

Two sections were dedicated to these two applications, each of them describing the purposes and main algorithm.

Then, in the last part, we saw which were the limits and continuations of this project.

We have nearly completed all the tasks that La Compagnie Lubat asked us to do. The detection application can help prevent collisions, while the control application send orders to the bots, following a Boids Algorithm. However, the user can't control the leader through a controller. This feature is not yet fully implemented.

Nevertheless, even if we couldn't implement all the features we wanted to make, this project was very interesting. We had a lot of fun while working on it. However we were disappointed because we weren't able to test our code and applications on the Metabots and with an infrared camera, because of the lack of means.

Due to the fact that we had several projects at the same time, we couldn't do as much things as we originally wanted to do. Moreover, the robotic team didn't manage to work with us due to their schedules, so we weren't able to merge our work.

This project was interesting, but many external factors prevented us to fully take advantage of it.