NAMeC - Team Description Paper Small Size League RoboCup 2020 Application of Qualification in Division B

J. Allali[1,2], J. Bezamat[2], P. Felix, S. Loty[3], V. Mignot[4], X. Muller[1], R. Pigret-Cadou[1], T. Saliba[2], E. Schmitz[2]

[1] LaBRI, Université de Bordeaux, France
[2] ENSEIRB-MATMECA, Bordeaux-INP, France
[3] CATIE, France
[4] Bordeaux Ynov Campus, France
julien.allali@enseirb-matmeca.fr (corresponding author)

Abstract. This paper presents the progress of the NAMeC team's work since 2019. On the hardware side, we have reworked the communication between the main computer and the robots, added a developer mode and integrated various electronic modifications. On the software side, the work environment has been simplified, we have added a more efficient logging system, and changed the AI architecture to optimize resource consumption and make it easier to register the different modules of the system.

1 Introduction

In 2019, we participated for the second time at the RoboCup SSL in division B under the name NAMeC (Nouvelle-Aquitaine Mécatronique Club). The first time that we participated is under the name AMC (Aquitaine Mecatronique Club).

This year improvements were mainly focused on reforming the basics. In the firmware, we reworked the communication protocols. We also implemented a mode in which the robot communicate with a different master than usual for maintenance/security purposes.

The 3D models of the robot were transferred on a different software in order to improve the access of the mechanics to other people in the team introducing branches and git-like approach.

As for the software, the basics has been simplified in order to ease the understanding of the code for new people joining the team. On one side, the installation and execution of the software has been encapsuled with Docker. On the other side, the development framework has also been reworked with a brand new architecture.

Globally we wanted to better visualize what robots were doing and what the software was processing. That's why a better logging system has also been added to the firmware and a new packet appeared on the communication protocol. A new interface will be developed later in order to synthetize and display these new data.

2 Firmware & Electronics & Mechanics

During the Robocup of 2019 in Sydney, some issues emerged concerning the communication between the robots and the main server. The will of this year was to improve the communication protocol and to improve a new feature that has been implemented in the rush of the competition of 2019.

2.1 Rework of communication pattern

The first thing we reworked on our communication is at a pretty low level. Before the beginning of the competition last year, we observed a strange behavior on our communication packet when the robot was charging its capacitors. In fact, it appeared that the electromagnetic perturbations due to the charger board were highly disturbing the RF communication. Our solution, which could help some teams having the same problems and needs a fast fix was to interrupt the charging each time the robot needed to answer to an RF communication. That way, the integrity of the communication packet was kept intact and the robot was still capable of charging (a bit slower). This fix isn't a long-term solution but it helped us at the last minute last year. This year, we are conceiving a new charger board which will not interfere as much with the communication. Moreover, the cable management will be reworked in order to separate the power from the logical signals.

The protocol of communication is also under rework. We are introducing new kinds of packet in order improve the monitoring of the robots and add the possibility to custom the configuration of the robots. Whenever a packet is received, the robot answer to the master. This answer will depend on the nature of the receied packet. These new packets are the followings :

- Configuration Packet : It includes all the limits of the robots (power, speed, acceleration) and enable/disable some functionalities (chip-kick, kick, dribbler, charger). This is useful in case a malfunction is detected on the robot. We can ask for a replacement and set limitations on the robot to prevent further damages.
- Ping packet : Used only for testing the communication. It includes a timestamp to do some workbench on the communication. It is not used during match.
- Command Packet : It is the most used packet. It is sent at a precise frequency. This packet sends the order to the robots. Speed in x, y and theta(rotation), kicks orders, charges orders, dribbler orders. The robot answer with its odometry and its status.
- Diagnostic Packet : A table of error has been decided with some errors code. This is used to monitor the state of the robot. It will allow us to detect errors during a match and send adequate orders to prevent other errors.

A new interface is developed in order to display the status of each robots and this way monitor everything from the side of the field.

We are also reworking the way of sending signal to the robot. We kept the same antenna and chip (nRF24L01+) but we are trying to use the Enhanced Shockburst protocol to implement a 6 simultaneous communication between the master and the robots. As we have 3 antennas per robots, we will be able to speed our communication and improve the robustness.

2.2 Match/Developer mode

As this team is a bit new. A lot of maintenance is needed between the games. A lot of testing on the field is also a necessity. The problem that occurred during Sydney was the main server was communicating with the robots on the test field and on the robot under maintenance. This situation could lead to some problems as the robots may charge their capacitors during a test. If someone doing the maintenance has their hands in the robots at that time, a big electrical shock can occur. Moreover, if the main server sends a movement order to a robot on a table, the robot may fall and destroy itself. To prevent all of this from happening the feature Match/Developer has been implemented.

The idea is simple, when the robot is on but in maintenance, the frequencies for communication are different from when it is supposed to play a game. That way, the main server can only communicate orders with the robots on the field. The protection of the robots has been modified in order to be removable easily. Some magnets has been inserted on the top. That way, when the protection (with the color pattern) is on the robot, the mainboard, equipped with Hall sensors detect the magnet and switch in match mode. If the magnets are not detected, the robot stays is developer mode. Moreover, there is 4 hall sensor on the mainboard so the ID of the robot is also represented by the location of the magnets. Thanks to that, the shield can be interchanged on the robots without modifying the code of the robot.

2.3 Electromagnetic noises solutions

As said before, the communication was higly disturbed by electromagnetic noises emitted by the charger board. That caused not only problems on the communication but also on the Hall effect sensor from the dribbler motor. The board and the cable from the charger board emmit high frquency noise going from -600mV to + 600mv. As theses cables were disposed all inside the robot, there are a big chance that these noises highly impacted the performance of the robot. The orange cable on the figure 1 are the cables delivering the power to the coil and the capacitors. The aluminium pack is the charger board. The board was placed as seen on figure 2 (on the middle plate). This cable management is obviously not optimal.

As a first test, we rolled the board and the cable with aluminium in order to imitate armored wire. We noticed a reduction of the amplitude of the noise to -100mV/+100mV. From that, we decided to rework the board and change the cable management.

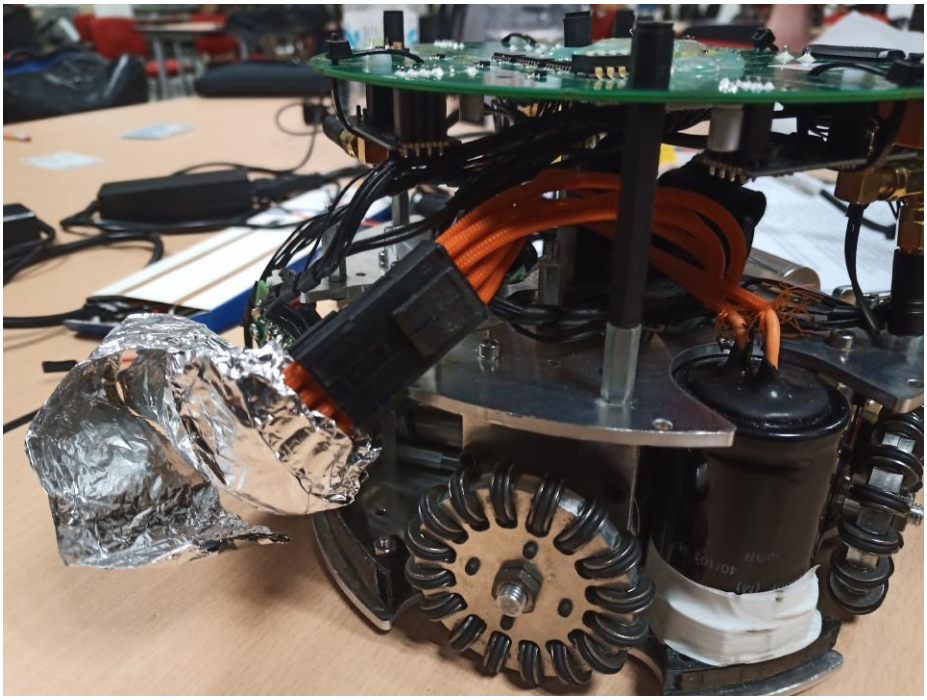

Fig. 1. Picture of the inside of the actual robot

The charger board will be modified by adding frequencies filters in the output of the boost in order to reduce the electromagnetic noises. As the schematic isn't finished yet, we can't detail the specifics. The board will also be placed far away from the mainboard. The new location will be in direct contact with the bottom plate (see Figure 3).

The main change is the cable management. First of all, as the cables were CPU power cable, they were not properly armored, we will swap them with armored ribbon cable. Then these cables will be placed inside the bottom aluminium plate as it can be seen in the figure 3. That way, the noise will be reduced by the new cable and the plate which is already at the ground. Moreover, the ribbon wire will be covered with aluminium tape to reinforce the noise reduction. This modification will remove all power cable from the inside of the robot deporting it only in the bottom plate. This feature will also help the maitnenance of the robot as we will be able to dismount the bottom plate without unplugging the boards and removing cables.

2.4 Mechanics

This year, all the mechanical parts and assemblies has been transposed into Onshape [2]. Before, that, everything was on Autodesk Inventor. As we were many

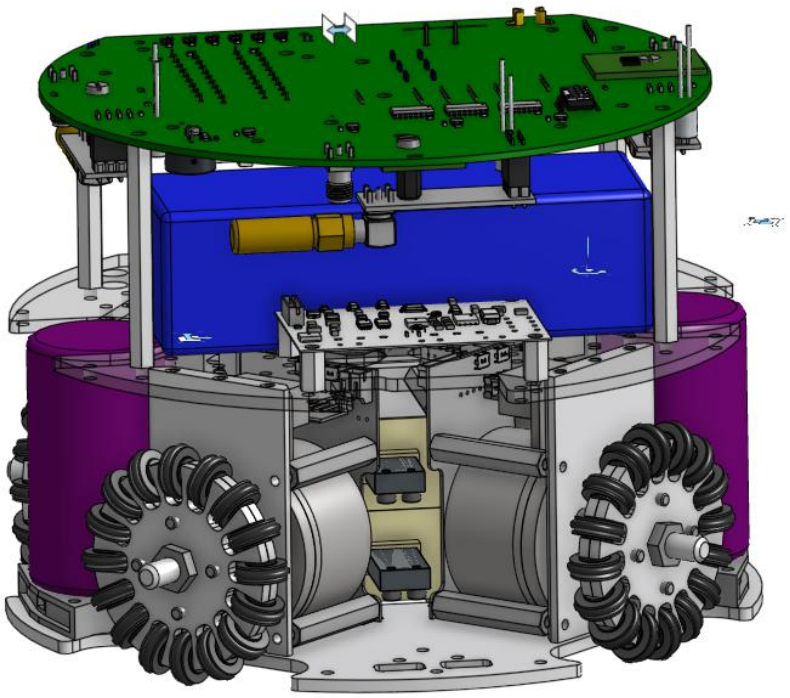

Fig. 2. Old version of the robot

wanting to modify the mechanics of the robot, Inventor wasn't satisfying. In fact, the file management is quite complex and reacts poorly on git. That way, if someone modified a part and somebody else the same on their local repository, the push/pull/merge would not work or, even worse, happen horrendously. Moreover, Inventor works mainly on Windows and as everybody on the team is on Linux, it implies to have a virtual machine or a dual boot.

Onshape works on internet browser so no need to swap on windows to work on the mechanics of the robot. Only a good internet connection is needed. Moreover, versions are easily made and any changes made to the robots is directly reflected online. That way, if somebody is working at the same time on the robots, the work would be updated with the latest version. Furthermore, the transition from Inventor to Onshape is not that difficult as it included some tools allowing to convert the file easily. The electronics circuits can be difficult to translate so converting them as a .stl is a better idea. Another quality of Onshape is that models can be exported in .urdf quite easily. This format can be imported in

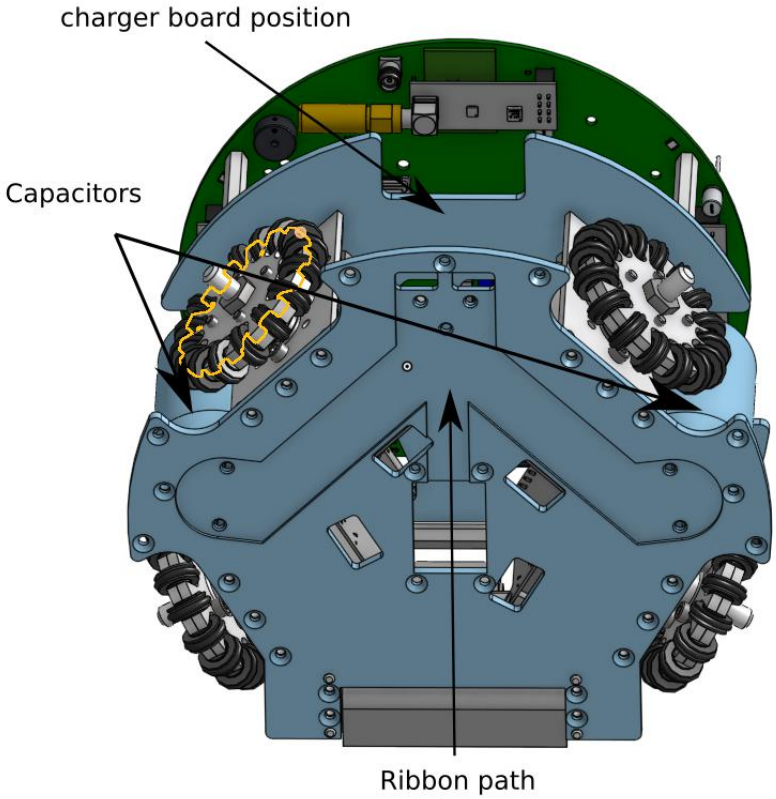

Fig. 3. Bottom plate for nesting the ribbon

Pybullet or V-rep for simulations. All our 3D models can be consulted on onshape at this url [1].

3 Software

This year our efforts in the software development are concentrated on robots security and on code maintainability. During the 2019 edition our robots haven't suffered any major breakage but at the cost of underusing their maximum mechanical capacities. Our objectives are to improve control of the AI to then allow to increase robots capacities and speed. Notably with improvements in obstacle avoidance. For this purpose but also to facilitate the work of future teammates, we improve the understanding and maintainability of the code. We are implementing a more modular code structure, in particular with a task oriented architecture. To clarify and simplify the use of our softwares, we encapsuled most of our code with Docker. As we wanted to have more control and monitoring over the robots and the software, a new log system has been also implemented.

3.1 Docker and environment tools

We used Docker to isolate grSim and the game controller inside containers. The first goal was to launch all the SSL stack on others Linux distributions and to simplify the setup of our stack for matches as well as for new students. Moreover, we used containers to launch grSim on a server without X graphical interface. The medium-term objective is to propose a continuous integration pipeline allowing to automatically test the robots' behaviors in different situations (such as shooting at goals or facing a simple defense) in order to determine the sustainability of the developed strategies, at least in simulation. There are a number of constraints that are slowing us down, especially concerning the generation of test environments so we are currently developing a middleware that allows us to prepare the configuration of the robots in the grSim simulation from a configuration file, and run our AI inside. We want this system to allow us to launch several matches simultaneously with several variations of our AI, if possible accelerated, and then feed back the information for each match, especially the winner. This will greatly facilitate simulation testing and allow us to make our AI evolve faster.

3.2 Task Oriented Architecture

The core parts of our software was not clear at all, however this year NAMeC team is mainly composed of students. A new way to organise our code is introduce this year to simplify the way we initialize and create our AI. Before, many data were dispatched and copied in various places. It was difficult to understand and maintain certain parts of the code without massive changes. As an improvement we centralize our data in memory combined with a simple task system.

Each circle and it's associated segment in figure 4. defines an API that is implemented as data structure. For example, to treat all inputs of the system, a pipeline composed of various tasks is executed at each new incoming message. These tasks will analyse, filter, extracts and store informations at different level

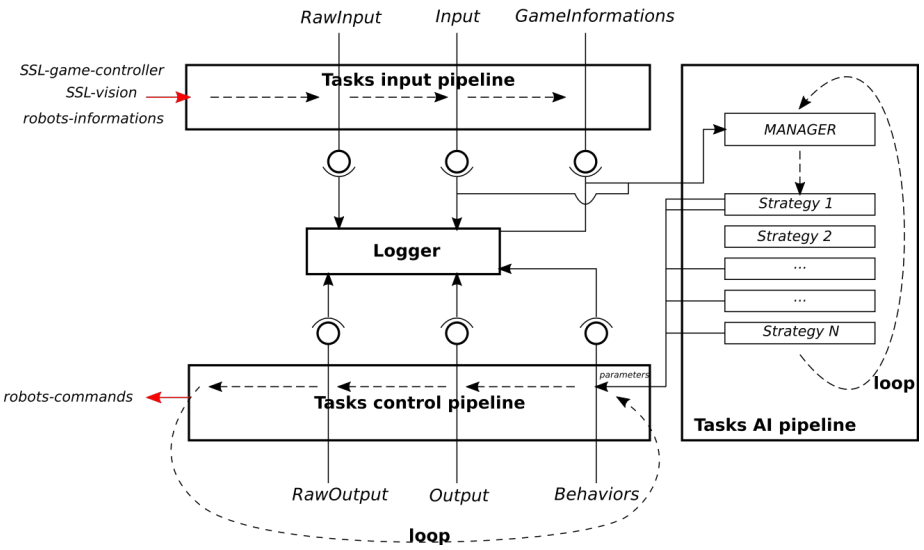

Fig. 4. New dataflow of the AI

of complexity from raw information received by the system. The AI will futher manipulate the real state defined in *Input* and take decision from internal custom informations store in *GameInformations*.
For our AI, we keep our architecture based on *Behaviors*, *Strategies*, *Manager* explained on our previous TDP but refactor it to take advantage of the tasks system. All the data, input and output are centralized to provide a simple way to interact with those data and create tools that take advantage of these characteristics (e.g. 3.3). Also the aggregation of data and the logic is clearly split.

The idea behind the task system, named *task manager*, is to simplify the way we initialize the program and organize the execution flow. This system executes each tasks synchronously in a given order in loop. Tasks are automatically removed when terminated.

Basically we have several types of tasks :

input/output : Retrieves protobuf data from multicast interface, sends commands to robots...
filters : Computes accurate information from raw data that will be use further by algorithms. (e.g. robot's velocity, acceleration...)
manager : Computes all robot's commands at a given game state. It corresponds to the artificial intelligence implementation.
checker : Analyses commands and global data structure to detect and correct errors/invalid commands.
thread launcher : Launches a task in a thread and close the thread at the end of the task or the program.

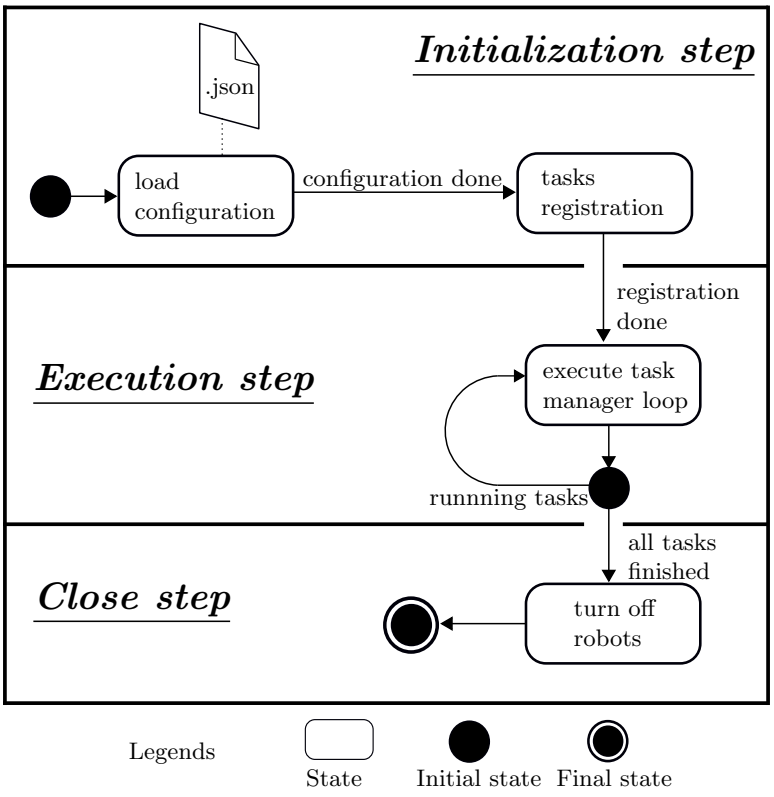

Fig. 5. Simplify execution of the main program

tools : All tasks that doesn't directly interact with the system. (e.g. benchmark task, logger task...)

We can easily select, change, compare and test all tasks thanks to this new architecture.

As shown in Fig. 5 there are 3 main step when the main program is launched. The most important one is the initialization step. During that phase, we want to have a complete control of which algorithms and *manager* are used according to our configuration. The code below Fig. 6 show how straightforward it is to create and understand our program initialization.

Fig. 6. Tasks registration

```
void tasksRegistration() {
    if ( config::simulation ) {
        tasks_manager.addTask(new GrSimIO(), 10);
    } else {
        tasks_manager.addTask(new SSLVisionInput(), 10);
        tasks_manager.addTask(new RobotsIO(), 11);
    }
    tasks_manager.addTask(
        filter::factory::get(config::filter_name), 21
    );
    tasks_manager.addTask(
        manager::factory::get(config::manager_name), 31
    );
}
```

As mention previously (3.1), we plan to create a more advanced test environment. We hope that all changes mention in this section will help to reach this objective.

3.3 Log system

We introduce a new logging system for our main program. Instead of only store all protobuf packages to replay them later, we decide to store the current memory state periodically using 'mmap' system call. This method take advantage of our data centralization that is inline in memory. That will permits to create new tools further without the danger of disturbing the main program. As long as the students work with the memory mapping they can't manipulate critical internal memory of the main program but they have access to it in read-only.

We have the possibility to save the state of the memory at any point in the code. That way, we can isolate the parts we want to test or debug which is a real benefit.

4 Conclusion

To recap this year, we improved the accessibility of the software and increase the robustness of the electronics. We hope that this changes will help newcomers to understand the project quicker and be effective as soon as possible.

For the next months, we will improve the tuning of the motion control system thanks to a real study of our robot's physics capability. On top of the motion control system, we will introduce global navigation algorithms. Currently, we only have a local navigation algorithm for obstacle avoidance but it lacks of efficiency on some configuration. The data used by the artificial intelligence will be pre-process by a new filter pipeline. Also we will integrate more tools as extension of our visualization software and increase it's performance.

The last two months before the Robocup will be used to develop new behaviors base on all previous improvement. We bet on our better robustness to perform better than the previous year.

References

1. https://cad.onshape.com/documents?column=modifiedAt&nodeId=fc1069ea056701d7518d57d9&resourceType=folder&sortOrder=desc
2. https://www.onshape.com/
3. Boussicault A., Felix P., Hofer L., Ly O., N'Guyen S., Passault G., Pirrone A. : AMC - Team Description Paper, Small Size League RoboCup 2018, Application of Qualifcation in Division B, (2018)