SDP Final Group Report

http://github.com/sdpgroup2/sdp/

Mark Nemec Vlad Cecati Gordon Edwards Motiejus Gudenas
Paul Harris Jarek Hirniak Michael Mair Julien Mélion
Michael Rowley

1 Introduction

The purpose of this report is to describe our efforts to build LEGO Mindstorms NXT robots capable of playing Robot Foosball for SDP. We used the LeJOS Java library for all parts of the project. In our initial vision we decided to split the project into 6 components connected through our main controller with library-like interfaces. This architecture is presented in figure 1.

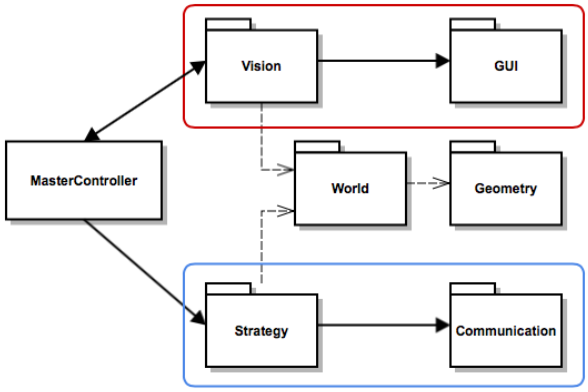

Figure 1: Diagram of main components of the project

In this report, we will outline the design of each of the components, their purpose and motivation for why we used them.

2 Team Management

2.1 Initial plan

We looked at group reports of previous year groups to find a good team management strategy. We used Github for version control, Trello for taskmanagement and Facebook for discussion. The work distribution for the first three milestones was to have most of our members working on code to pass the milestones and few people focusing on the other components which were going to be used in latter stages of the course.

This method got us through the first two milestones for which we received high marks. However, it did not provide good enough starting ground for the third milestone for which we were not able to field robots. There were multiple reasons for this.

2.2 Change of plans

Firstly, team members were working in seclusion which meant others did not know how to use the work they had done. Secondly, the code was complicated and was not tested incrementally but merged together a few days

before the milestone. Finally, most members were not clear on how to help with the project.

To attempt to improve our team organisation, we started having more regular meetings, particularly face to face. We also decided to abandon Trello, because team members were neglecting to look at it, and only used Github to manage project tasks. We therefore had a single simple way of managing the project.

3 Robot Design

We experimented with having either one or two light sensors in different positions and decided light sensors were not needed in our final design.

Many different kicker designs and positions were tested, such as kicking from the side, or, as we discovered was more effective, the front of the robot.

Firstly we tried to use a regular Lego Servo motor for the kicker, but this took up too much space and often struggled to fit within the size constraints, especially once we added the cumbersome brick. We then used a smaller, stronger motor for the kicker design to maximize torque. Using a simple gear train we dramatically increased the speed of the kicker.

The robot used a simple two wheel drive using Servo motors for accurate rotations, as speed was not a high priority. A third holonomic wheel at the back of the robot was then added to enable smooth rotations and to add stability. We experimented with using a four-wheel drive powered by two motors, gearing the wheels of sides together. However, this presented problems with turning, as having front and back wheels moving at the same speed with rubber wheels caused the robot to move and turn erratically.

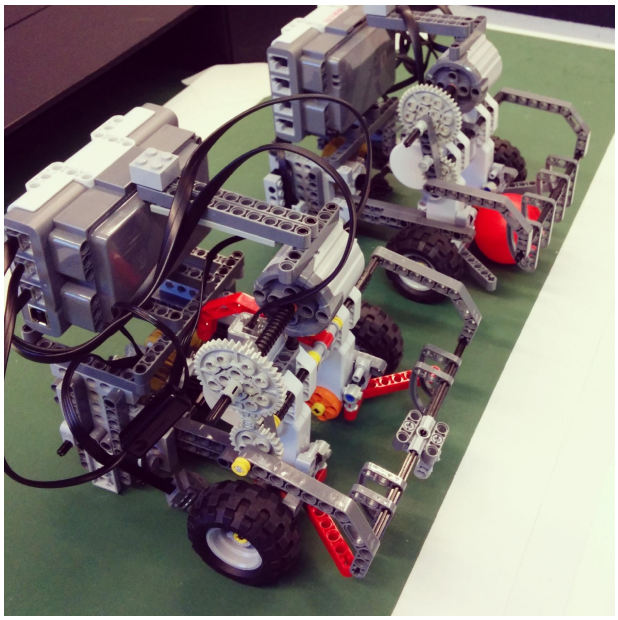

Figure 2: Final Robot design

For our final design we took some inspiration from the more successful teams. For example, we used a different kicker design, a cage, which was able to grab the ball and move with it. We added two pincer-like parts to the front of the robot to guide the ball into the middle, and had curved pieces attached to the cage, so that the ball was always aligned directly in front of the kicker, ensuring accurate kicking. We tested the positions of the pincers to make sure that the ball would always be driven to the same place of the robot when the cage closed.

4 Communication

To avoid having to build the communication system from scratch, we decided to base it on the code of a previous year's group. We chose Group 4 from 2013, as their system seemed the most straightforward.

After the PC connected to the NXT brick via

Bluetooth, commands, such as "move" and "rotate", could be sent to the robot. We represented these commands as an opcode along with optional parameters to specify, for example, movement speed. Commands chosen by the strategy system would be queued up on the PC side and sequentially sent to the robot. If a sequence of commands had to be changed, the queue would be cleared.

5 Vision

Initially, our vision system was based on Group 4 from 2013 and implemented without the use of any vision libraries. This resulted in a low frame rate, around 10 frames per second (fps), and a needlessly complicated system which was difficult to parameterise.

Hence, we decided to rewrite it, taking ideas from Group 9 from 2011, using the OpenCV library. It took us about a week to re-implement but it was well worthwhile. The new vision system was able to correctly detect the position of the ball, and the position and orientation of the robots.

Furthermore, the system was very easy to configure and was running at the optimal 25 fps. In a nutshell, the vision system is responsible for grabbing the image from the camera using V4L4J, feeding it to be processed using OpenCV, and updating the world state afterwards.

There are three main stages when processing the image:

5.1 Pre-Processing

In this stage the system performs camera lens distortion correction, which removes fisheye distortion caused by the camera, crops the image to remove unimportant sections, and applies a median filter to the image to remove noise. These steps proved to be important as without them the quality of the image was substandard and regular shapes such as the pitch were distorted.

5.2 Object Detection

In this stage, we convert the image into HSV representation. We then perform minimum/maximum thresholding on each channel. This approach proved to be effective, especially with configurable thresholds - we were able to threshold our vision system in less than 5 minutes. We also tried solutions with RGB representation and adaptive thresholding, but they did not work well because of varying lighting conditions. They were also difficult to implement correctly.

Furthermore, we perform contour extraction and calculate image moments. With these methods we were able to detect image blobs and their properties such as their centre. We also compute areas to filter out blobs that do not resemble the ball or the robots. Hence, our system avoided falsely detecting people's hands as balls during matches.

Retrieving the orientation of robots proved to be very challenging. Our system thresholds for the dots only within a small radius around the centre of the base plate. This was reliable and avoided the problem of detecting black-coloured pixels outside of the base plate as the dot.

5.3 Post-Processing

In this stage we perform perspective projection correction and convert all coordinates to

millimeters. Perspective correction was important as without it a robot and a ball with the same x-coordinate on the pitch would not appear as aligned due to the robot's greater height. After the perspective correction, the difference in x-coordinate of an aligned ball and robot was smaller than 1 cm. Converting to millimeters allowed for a more precise and unified representation of objects across both pitches.

6 Vision GUI

This is the part of our project which was visible to the user during matches. It provided information on object positions, quick thresholding, and a means of saving thresholds for each PC in a separate JSON file. The last point was important as different PCs delivered different images - contrast and brightness varied.

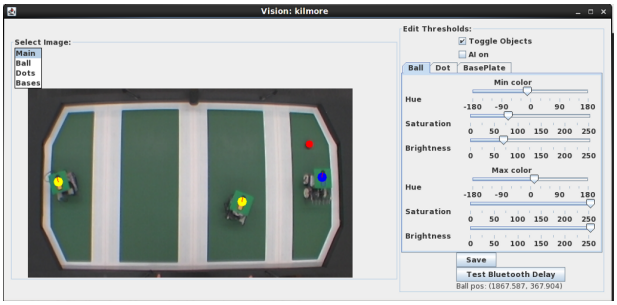

Figure 3: Vision GUI

7 Strategy

Our initial ideas for the strategies of each robot were basic. We had two distinct major states for each robot: one when the ball is in the same zone as a robot and another when it is not. When the ball was not in the same zone as a robot, the robot would try to mirror the position of the ball while remaining in its own zone. For the defender, this meant moving parallel to the goal to block shots. For the attacker, this meant trying to intercept passes from the enemy defender to the enemy attacker. Moreover, the attacker would try to grab the ball when it was in the same zone, and shoot it towards the goal.

This strategy worked well enough for the defender to block the ball, but not kick it. The attacker was able to kick, but the ball was rarely in the zone long enough to be grabbed.

To solve these problems, we altered the defender to change its behaviour when the ball was in the defending zone. Much like the attacker, it would grab the ball and pass it into our attacker's zone. We also altered the attacker to position itself facing the defender if the defender was in possession of the ball. This made it easier to receive passes. With these improvements, the ball spent significantly less time in our defender's zone while our attacker had more chances to shoot at the goal.

An additional feature we added to our offensive robot's strategy was a trick shot. It involved the attacker spinning a full 360 degrees with the ball before taking a fully powered shot. We added a random element to it as well, so that it would shoot randomly within 20 degrees of the angle from the robot to the centre of the goal. The quick spin is used to confuse and throw off the opponent's vision system (which predicts where our robot is facing using the direction vector from the plate), and the random element means that the opponent will not be able to reliably position themselves to block the shot in time (of course this strategy could backfire and shoot into the opponent as well).

We did, however, encounter quite a few difficulties with the movement and actions of the robots. Grabbing the ball was an issue if the ball was against a wall or corner. Our grabber would get stuck over the edge of the pitch when it got too close to the wall, due to its large size. Had we detected when the ball was next to the wall using the vision system, we could have had the robot position itself accordingly. However, given the simple design of our robot, and the use of a rear holonomic wheel, the actual robot movement proved to be relatively easy to implement with only two motors to program. Furthermore, the manoeuvres of the robot were consistent and reliable.

We could have also improved our robot's strategy and performance by trying more complex manoeuvres and actions. For example, we could have used the walls and bounced the ball off them for passes and shots in order to get past the opponent's robots more reliably, as it would be harder for them to prepare for an angled shot. We could have also implemented PID control or other techniques for smoother manoeuvring of the robot, which would have helped the robot to grab the ball near walls. It would have also made the robot's movement around the pitch faster and more adaptable.

We found over time that, though both robots share much of the functionality. Initially, we had entirely different code for each robot, but, ultimately, we found that it would be simpler and more manageable, if we had the same code foundations for each robot, and then added role-specific actions on top of this.

8 Lessons Learned

Our performance by the end had significantly improved because we adopted a better approach:

- **Keep it simple**: write simple, modular code. Only make changes in small increments, don't introduce a lot of new untested code in a single stage.

- **Organise effectively**: have more regular team meetings, everyone should aware of what everyone else is doing.
- **Don't reinvent the wheel**: use libraries and reuse code wherever possible - don't build from scratch unless there is a significant benefit.
- **Finalise the robot design early on**: reduce repeated testing because each new design had to be tested, and frequent redesigns wasted time which could have been better spent elsewhere
- **Test frequently**: we used some JUnit tests, but our advice would be to test the functionality thoroughly after any code changes to the robot redesigns, not just with one small-scope test.
- **Reuse existing code**: The aim of the project was to build on work that had already been done in order to fully focus our efforts on improving and elaborating on work from previous years, rather than repeating what had already been done.