Grabber Designs and Software Solutions for the KIPR Wallaby

David Fischer, Alexander Brenner, Sascha Nesterovic, Karl Forstner, Marko Miletic, Jan Giefing and Rafael Schreiber
Technical Secondary College
Department of Computer Science
2700 Wiener Neustadt, Austria
Corresponding author Email: fischer.david@student.htlwrn.ac.at

***Abstract*—Grabbers are one of the most essential parts in winning a Botball tournament, and so is the software running on the bots. In this work, we want to introduce different simple but expandable grabber designs and software programmed with the current Botball kit in mind. We wanted to make using everything mentioned as easy to understand and implement as possible so newcomers will have an easier time getting into Botball. Early on having these kinds of resources as a team can help eliminate various issues and speed up the development process overall.**

I. Introduction

Participation in a Botball tournament requires a Botball kit, which is a carefully selected collection of LEGO and metal parts, sensors and servos. The kit's main components are:

- Two KIPR Wallaby controllers
- The iRobot Create 2®Programmable robot
- LEGO Technic bricks as well as some metal parts
- Various motors and Sensors

A combination of all these parts will allow one to build a robot, which should be able to complete a majority of the tasks on the game table. Before building the robot, the developers and engineers should work together closely to plan a basic structure with a simple and service friendly design that is able to complete the tasks it gets assigned. In our opinion, and also from our own experiences, solving problems during the building phase is much easier than during the programming phase, therefore teamwork and clear communication during this period is crucial. The Botball kit provides a wide range of possibilities concerning the bot's construction, so there are always multiple options to complete the different assignments on the game table. Of course, it is often difficult to come up with creative designs for these tasks, therefore we constructed three different types of basic grabbers and demonstrated their integration onto the bots. Troubleshooting during development is also challenging, that's why we designed a way to check a servo's integrity using only the Wallaby's integrated gyro sensor. In order to allow quick development and testing for these purposes, we developed remote-compile, an efficient and easy-to-use interface for compiling and running code on the Wallaby. For an explanation of its functions and other features see section IV.

II. Grabber Innovation and examples

Since the game table varies every year, we constructed and tested three simple but efficient grabbers which fit most of the game table's reoccurring objects, like the botguy and basic cubes. These grabbers are very straightforward to build and are perfect for Botball beginners. Most of the grabbers require a servo, cogwheels and some LEGO parts, which are all contained within the standard Botball kit.
We made blueprints for the grabbers in the LEGO Digital Designer (LDD) [1], a program, which can be downloaded from the official LEGO website and makes for a great and easy-to-learn tool for computer aided design (CAD) of LEGO structures.

A. Gravity grabber

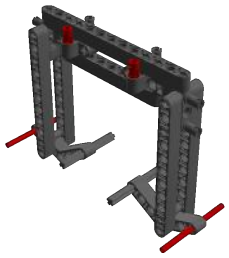

Fig. 1: CAD model of the gravity grabber. Since LDD doesn't feature rubber bands the spots which are meant to be connected are marked in red.

There is a limited number of servos in the Botball kit, therefore we felt the need to develop an entirely servo-less but still functional grabber. This is why we constructed the gravity grabber as an alternative to the usual grabber concepts. When grabbing from above, the grabber's left and right bars are pressed inwards, so that the object can fit into the grabber. After the object has moved to a position where the rubber bands are able to pull back the two bars, it is locked in place and the grabber won't be able to drop the object. This design has some obvious flaws, but it's perfect for placing the botguy in the baskets from 2018's game table. Its size is about 3.1 by 5 inches and its span length measures about 4 inches.

See Fig. 2 and 3 to see a demonstration of the grabber's grip on the botguy and a small cube.

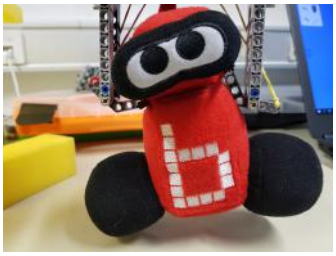

Fig. 2: The gravity grabber grabbing a botguy.

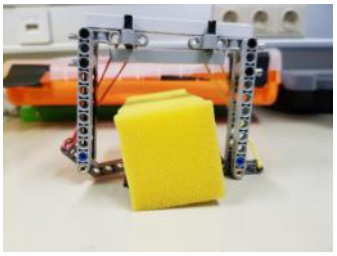

Fig. 3: The gravity grabber grabbing a small cube.

B. Horizontal grabber

Fig. 4: CAD model of the horizontal grabber from behind. The servo should be mounted at one of the cogwheels in the center.

Our horizontal design is a basic grabber constructed to perfectly fit the botguys shape. It is one of the most basic but efficient grabbers, which makes it very common at Botball events. It is best used from the side around the botguys neck, where it excels at providing a very secure grip. The grabber consists of four cogwheels and one servo which is mounted at one of the middle two wheels. This grabber's weakness are poms since the only option is pushing them and picking them up reliably is almost impossible. To pick up cubes, this design would have to be scaled up which would reduce the amount of grip on the botguy. Its size is about 4.5 by 3.1 inches, its span length is about 7.5 inches.

For our setup see Fig. 5 and 6, which show the grabber's grip on the botguy. Considering that the servo position ranges from 0 to 2047, the optimal amount of servo strength to achieve the tightest grab is about 78 percent or a servo position of 1600.

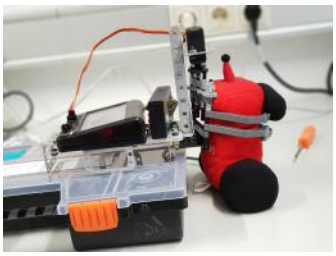

Fig. 5: Horizontal grabber grabbing a botguy.

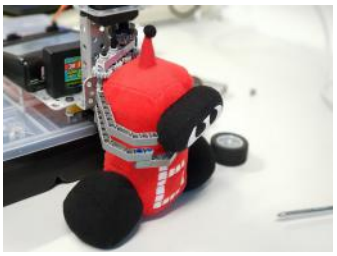

Fig. 6: Front view.

C. Vertical grabber

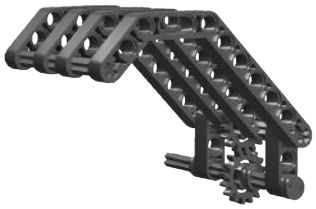

Fig. 7: CAD model of the vertical grabber. Servo should be mounted at the bottom cogwheel.

A vertical grabber constructed to fit most of the basic cubes and poms on the game table. This grabber is built to push and drag around objects like poms and cubes. Its concept is similar to an excavator since with the grabber's claw-like design it is best to lower the claw on top of the objects and drive backward to move them to a different location. This works best on cubes and small groups of poms. Its size is about 4.8 by 5 inches, its span length is about 3.1 inches, and the optimal grip is achieved by pushing the grabber on the floor.

See Fig. 8 and 9 for a demonstration of the grabber's grip on poms and a small cube.

Fig. 8: Vertical grabber with poms.

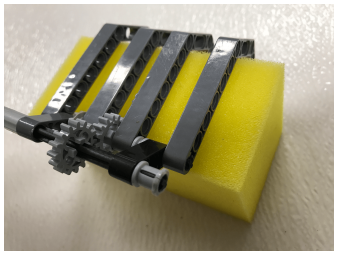

Fig. 9: Vertical grabber with a cube.

D. Using grabbers

Since these highly customizable grabber designs can be added to almost any bot or on any claw arm, we hope this section can, above all, help new teams get an idea on what makes a good grabber. Blueprints for all of the grabbers can be found online [2].

III. The Wallaby's gyro sensor

The Wallaby contains an integrated gyro sensor [3] which measures values in a 3-dimensional space in the directions x, y, and z. This dataset of three values provides an estimate on how the bot is currently moving. KIPRs libwallaby includes four functions that return the gyro's output.

- `int gyro_calibrate()`: Initiates the gyrometer's calibration
- `signed short gyro_x()`: returns the current movement of the x-axis

- `signed short gyro_y()`: returns the current movement of the y-axis
- `signed short gyro_z()`: returns the current movement of the z-axis

See Figure 10 to get an idea on how the gyro sensor responds to motion on the y-axis.

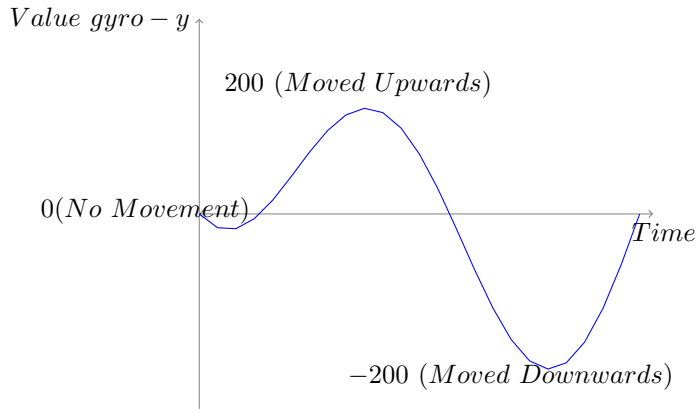

Fig. 10: Graph showing the gyro's output when first moved upwards and then downwards along the y-axis.

Even though the gyro sensor has no obvious applications in Botball, there are multiple uses for it. Our basic idea was to develop code that can detect a claw arms fluctuations and check if the servos used to mount the arm are still working smoothly. To confirm the validity of our consideration, we performed the following experiment using a basic claw arm.

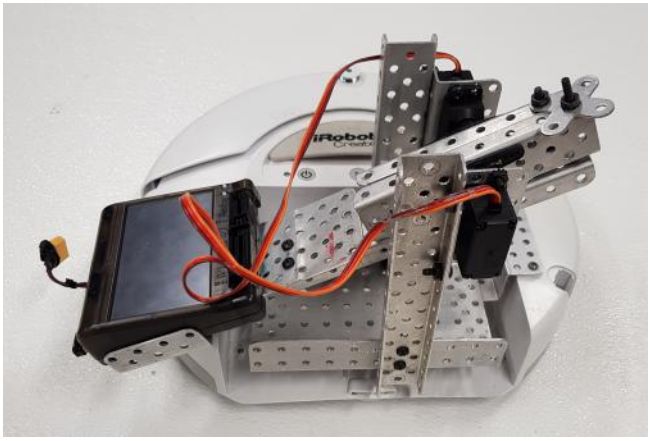

Fig. 11: iRobot Create and Wallaby with the gyro setup in place. Moving the claw arm makes the gyro return data.

The claw arm in Fig. 11 is the same our team used during the ECER 2018 Open competition. To complete the setup, we would have to place an igus chain inside the claw arm between the servos and mount a grabber on the chains tip. By reading the gyro sensor's y-coordinate output, the bot can calculate if the claw arm has moved upwards or downwards using a simple function. If the y-coordinate is static between -30 and 30, then one may assume that the servos are defective and have not fulfilled their tasks. When the Wallaby moves upwards, the y-coordinate changes to up to 500, moving the Wallaby downwards changes the variable to -500. If the Wallaby is motionless, the y-coordinate varies between about -30 and 30.

To demonstrate this, we wrote the following code snippet, which moves the claw arm and checks the servo's integrity afterward using data from the gyro sensor.

```
// this program assumes that the Wallaby
// is mounted on the claw arm
int maxValue = 500;
int minValue = -500;

int minStand = -30;
int maxStand = 30;

int servo0 = 0;
int servo1 = 1;

int timeInMs = 1000;

int main(){
  servoDown(2, 3);
}
// changes the claw arms servo position
bool servoDown(int position0, int position1){
   while (gyro_y() < maxValue && gyro_y() >
       maxStand){
      //changing servo position
      set_servo_position(servo0, position0);
      set_servo_position(servo1, position1);
      msleep(timeInMs);
      if(gyro_y() > minStand && gyro_y() <
          maxStand){
         // if the Wallaby thinks that there
         // was no movement false is returned
         // to indicate the broken servos
         return false;
      }
   }
   return true; // if the Wallaby moved
}
```

A solution like this can also be applied to a wire rope hoist construction to determine the Wallaby's current position without using a click sensor. This method also has its downsides, like the added amount of tasks for the Wallaby to complete, which might seem insignificant, but when added up can end up being a lot of time.

IV. REMOTE-COMPILE

A. Introduction

Currently testing and running all the different programs written for the Wallaby is a bit tedious and slow, therefore in this section, we want to introduce remote-compile: a compact program that allows one to remotely compile and run a program on the Wallaby.
We took heavy inspiration from the fl0w project [4] and got some help from its developer Philip Trauner early on. Since a

system like this wasn't finished in time to be implemented in fl0w we decided to write one ourselves. We have developed remote-compile to solve Harrogate's [5] biggest problems:

- Speed: The Wallaby is not the fastest controller, that is why we decided to write a lightweight program to compile our projects. In our opinion, the current system is flawed and puts too much load on the Wallaby, which slows it down significantly.
- Individuality: Programmers want to work in their favorite IDE, since it might feature more functionality than Harrogate, like typing suggestions or debugging.
- Customisability: Harrogate only has one interface and is unable to interact with other programs. With remote-compile, the client can be written in any language because of its fully featured and documented API.
- Security: Harrogate doesn't support any authorization, as a consequence, everyone in the Wallaby's network cloud access the Wallaby's projects and has the possibility to harm or mess with them.

That's why remote-compile was developed with speed, functionality, simplicity, security and cross-platform compatibility in mind.

B. Implementation

Remote-compile uses a self-developed protocol called RCCP (Remote Compile Communication Protocol), which uses TCP as the transport protocol. During development, our main goal for RCCP was speed and expandability. RCCP never leaves OSI layer 4, the transportation layer so it can achieve fast and reliable communication even in unstable networks. We used TCP, since it guarantees, that no data gets lost and manages broken packets by re-sending them. The remote-compile server runs on the end device (e.g. Wallaby) which can be reached via a simple TCP connection. RCCP, the backend protocol of remote-compile, works by sending a "UNIX style" command from the client to the server and making it respond with a JSON string.

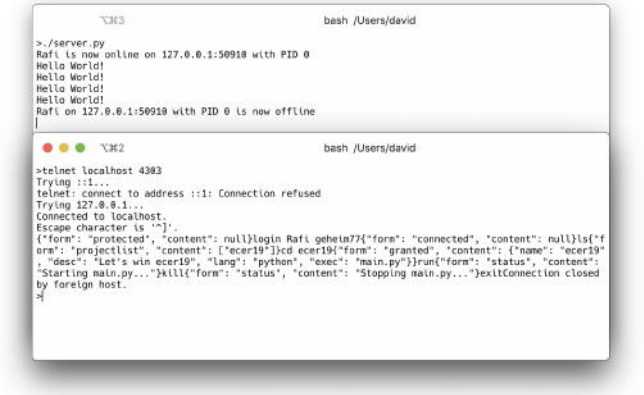

Fig. 12: Telnet login example with both programs running on localhost.

The picture above shows the login process between a client and a server running on the same machine. This style of communication allows an efficient integration into other programs. The JSON file format especially offers enhanced compatibility because almost every programming language provides standard libraries for parsing JSON data.

C. Functionality

remote-compile currently supports C, C++, and Python code on all major platforms (Linux, macOS, Windows). On-demand, additional languages can be implemented, thanks to the very expandable and well documented RCCP protocol. You can also protect access to the server with a password for enhanced security. Likewise, remote compile is not just limited to Botball, it may also be used for other types of projects on almost every platform.

D. Possibilities

We think using remote-compile on the command line currently isn't that great of an experience, that's why we also wrote a fully featured Atom plugin [6] which interfaces with remote compile without the users having to establish a connection themselves. The plugin uses 3 simple shortcuts and makes running programs on the Wallaby easier than a regular ssh connection would. The complete documentation can be found on the Atom-remote-compile repo [7].

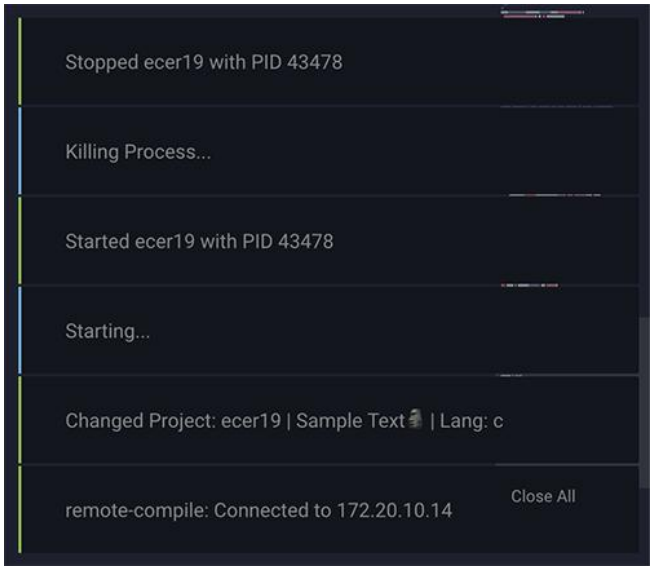

Fig. 13: The Atom remote-compile plugin establishing a connection, changing the directory automatically and running the project (notifications read from bottom to top.)

Thanks to the remote-compile API [8] a frontend implementation like this can be written for any editor or platform. More about how our client and server gbased communication using RCCP works can also be found in the Documentation [8].

E. Future Development

Multiple additional features are planned for the future such as:

- end-to-end encryption: So remote-compile can feature even more security. This is currently in development and is planned to be finished in Q2 of 2019.
- More plugins for editors: Since the Atom plugin is finished we have moved on to developing a plugin for VS Code.
- File transfer: This would fix the fact, that one still has to rely on an editor plugin, like remote-FTP or an SFTP connection to the Wallaby, to move code onto the Wallaby.

V. Conclusion

We hope that this publication will help new teams to get a quicker start into Botball and provide them with valuable resources to support their bot's construction. Sometimes getting creative is a hard task and that's exactly why we wrote this paper. As a team, we have had these kinds of problems in the past and getting over them definitely was a challenge. Having issues, especially early on as a team, may lead to more obstacles later so having a basic idea about how to start can help a lot. Having a plan in mind before starting construction is vital and our grabber presets, unmodified or not, could be a great addition to bots on this year's game table.

Acknowledgements

The authors would like to thank Dr. Michael Stifter for his support throughout the work on this paper and Mag. Johann Punz for proofreading our final draft.

References

[1] L. A/S, "Lego digital designer." https://www.lego.com/en-us/ldd, 2004.
[2] break, "Grabber build guides." https://konst.fish/grabbers, 2018.
[3] B. KIPR, "libwallaby gyrometer." http://files.kipr.org/wallaby/wallaby_doc/group__gyro.html, 2017.
[4] S. S. N. K. N. L. C. Z. S. Z. Philip Trauner, Christoph Heiss, "fl0w - a development environment for the kipr wallaby." http://webspace.pria.at/ecer2017/papers/Paper_17-0603.pdf, 2017.
[5] KIPR, "Harrogate github." https://github.com/kipr/harrogate, 2018.
[6] D. Fischer, "Atom remote-compile package." https://atom.io/packages/remote-compile, 2019.
[7] D. Fischer, "Atom remote-compile repository." https://github.com/konstfish/atom-remote-compile, 2019.
[8] R. Schreiber, "remote-compile documentation." https://frontend.works/remote-compile, 2018.