Project report DEIS

Simon Brunauer, Harald Lilja, Anton Olsson

January 2020

1 Introduction

In the course *Design of Embedded and Intelligent Systems* the students of the course are posed with the problem of solving cyborg terraforming on a small scale. The idea behind the project is that we will have robots mimicking biological life, in our case shrimps and insects. This is due to animals ability to create their own living environment. The students of the course will create robots that will in part be controlled by said animal life. Looking at this project on a larger scale the robots could with a larger budget and more research help create living environments on foreign planets. The students will also have to create a robot that has the ability to interact with other robots in situations such as driving behind a leader, overtaking and other driving tasks. Most of the problems was shown with several robots driving in the environment at the same time.

1.1 Tasks

- Driving tasks - This consists of problems such as switching of lanes and turning on intersections.
- Cyborg terraforming - Detecting biological motion and having the robot perform movement in a corresponding manner.
- Platooning - Collaboration between two or more robots to communicate and synchronize behaviour. Switching of leader, following the leader and side-formation are some of the tasks to be performed.

These are the main problems to be solved and will be answered in the result section with images and corresponding texts.

1.2 Robot and environment

The robot constructed for the course is based on a Sparkfun Redbot [1]. The basic setup of the robot uses infrared sensors pointed towards the ground to detect lanes. Encoders attached to each wheel to be able to do odometry. A front facing camera for sensing of the environment such as walls, other robots or possible hazards. Each wheel has a corresponding motor and this is all connected through a micro-controller and a small computer. The micro-controller is a Arduino Uno. The micro-controller is in charge of the low-level tasks while the computer, a Raspberry pi 3, is in charge for the higher level of computation. The raspberry is running on a version of ubuntu xenial tailored for using Robot Operating System(ROS)[2]. The image can be retreived from [3]
In addition to all this the robot was further improved by adding some additional things. The first being a 3D-printed shovel to be used in possible terraforming tasks. To protect the computational unit from outside environmental problems, such as dirt and rain. A final sensor was added to help with sensing distances to object, this is a Sonar-sensor.
The robots' GPS-coordinates in the environment is calculated from a overhead camera mounted over the arena. To differentiate between the robots each group is also handed two identification-spirals. When mounted on each robot the camera will be able to detect and identify each robot through a script run on a server which in turn will return the ID and position of each spiral in the environment [1].
This setup was built two times as each group of students received two kits of robot parts and can be seen in Fig. 1.

[1] https://www.sparkfun.com/products/12649
[2] http://wiki.ros.org/
[3] https://downloads.ubiquityrobotics.com/pi.html

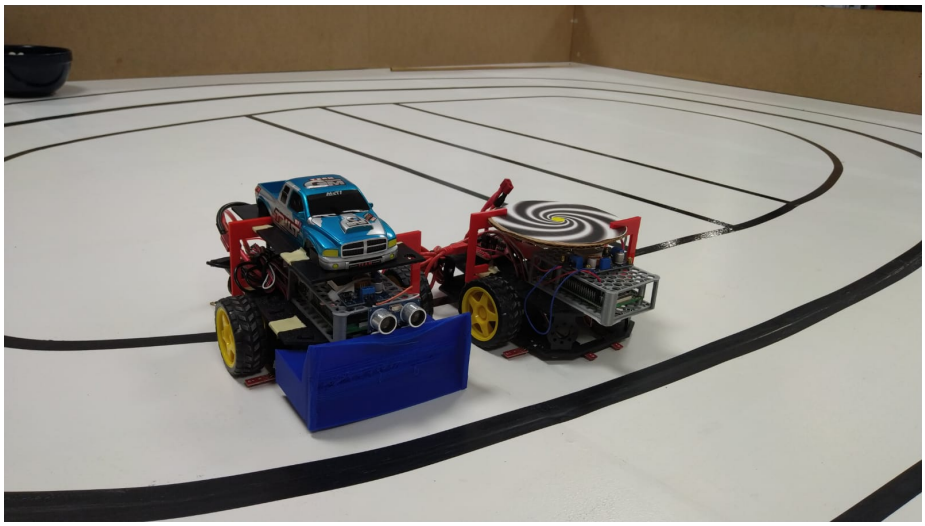

Figure 1: Image of the two Robots constructed in the course.

The map is a rectangular area constrained by walls on all sides. The ground is supplied with tape-stripes to mark the drive-able area as well as divide it into two lanes. The stripes in the middle of the circle is a intersection where a robot can enter while it is not driving. This arena can be seen in Fig. 2

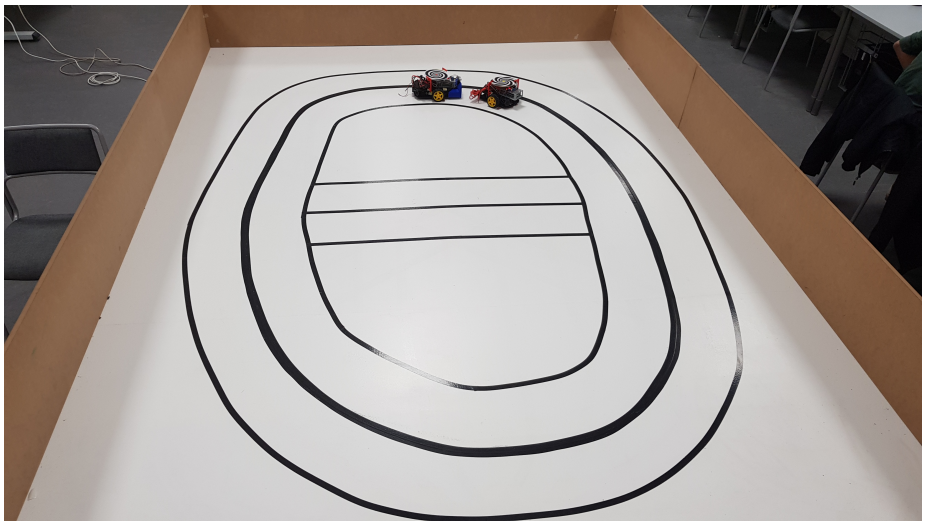

Figure 2: Image of the arena used as environment in the course.

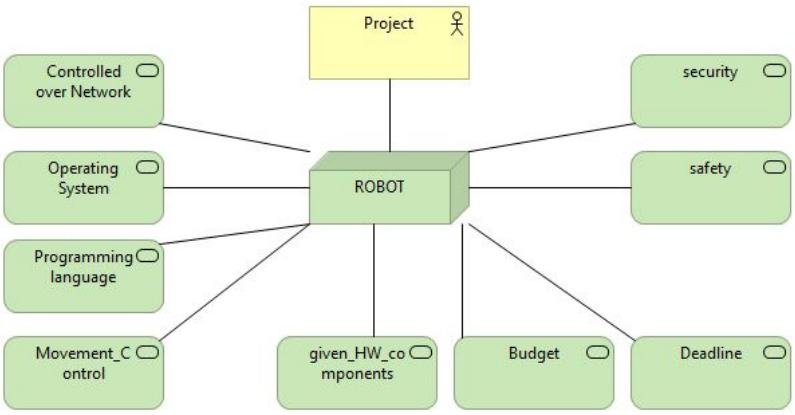

Figure 3: High level-requirements

1.3 Requirements

The following segment explains the high- and low-level requirements. The high-level requirements consists of using the right operating system, that the robot is supposed to be controlled over the network and we should be able to control the movement of the robot. These are just a few of the high-level requirements which can be seen in Fig. 3 The low level requirements consists of using ROS, using either CPP or Python, having an emergency stop and staying within the budget. All of the low-level requirements we have can be seen in Fig. 4.

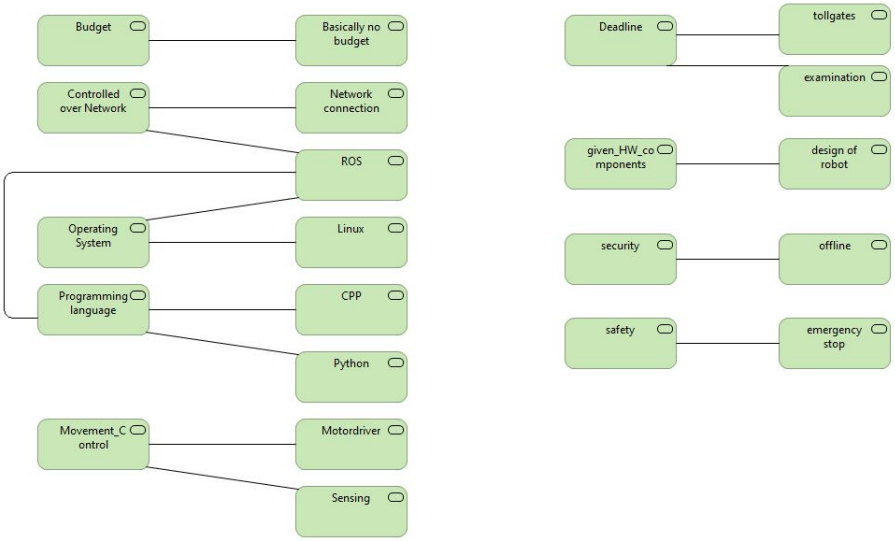

Figure 4: Low level-requirements

2 Method

The final version of the code consists of 4 different files. One file is running on the arduino and takes control of the motors, a second one is the main controller, which sends corresponding actions to the arduino. The other two programs generate the input for the controller. The shrimptracker tells us where the shrimp is and the commandsender sends different commands to the controller.

2.1 First Model

Fig. 5 shows the first model we planned for our software. There we planned to split up the software into different parts. First of all we wanted one part, that is used for the whole communication. Then we planned an own software-part for image processing, as well as for track sensing. The odometry should also run on its own.
The biggest software part was planned to be the controller, which should interact with all the other parts over interfaces. It should be used for all the decisions regarding the state of the robot and should control its movement.

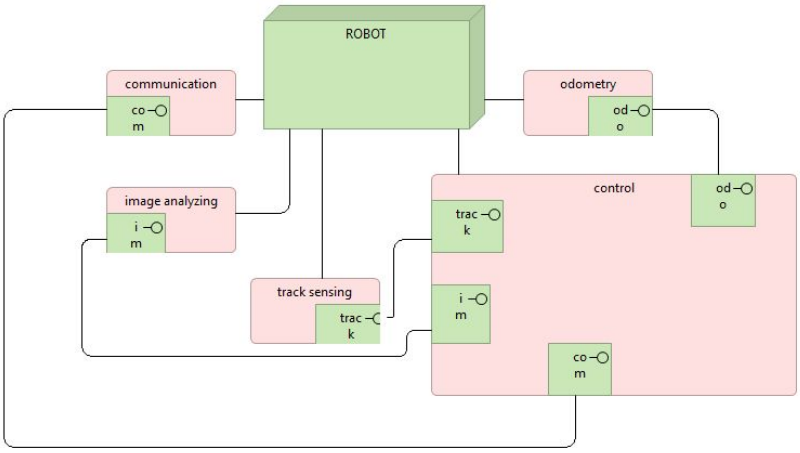

Figure 5: Platform Independent Model of software for first attempt

2.2 Arduino

The *actuator.ino* is running on the arduino. We implemented rosserial on this one to enable the arduino to listen and publish to rostopics. The program is responsible for linefollowing. Therefore, it listens to the linefollowing topic and does the linefollowing with according speed. In this mode it looks at the optical sensors and if one of the outer sensors detects a line, it corrects the speeds applied to the motors to turn it and keep inside the lane.
The actuator also has two topics for the motors. With those speeds can be applied to each motor individually, to control the robot manually, or via a shrimp. Next to this it has a topic for lanechange. The corresponding function will then turn the robot to the direction of the lane we want to change to, until it detects the line with the corresponding outer sensor. Then it is moved straight for a bit to get into that line and afterwards it is turned again into the opposite direction.

2.3 Command Sender

The commandsender lets us control the robot manually. Based on the key pressed on the keyboard of the PC, an action will be sent to the controller via the robotname\action topic. This allows us to switch between the different modes, which are manual control, side formation mode, autonomous mode and shrimp tracking. It also allows us to send the corresponding commands for all the other actions, like lanechange and intersection.

2.4 Controller

The controller is the brain of the whole project. It controls the whole movement based on the inputs from the command sender and the shrimp tracker. It has mainly four callbacks and therefore listens to four topics. It is able to publish to the topics, to which the arduino listens to, to control the movement. It can also publish to the feedback and action topic, which were used for communication amongst the robots. There are two different controllers for the two robots. They are basically the same, but use different speeds for some tasks, as the motors from the two robots react different to speed values and thy have a slighlty different setup based on the beginning states and names of the robots.

2.4.1 gps_cb

The first callback is the gps_cb. This one listens to the gps server. It reads the current position of the robot, based on the number of the spiral. Those coordinates are also used to calculate a polar angle between robots in the same platooning group. This is then needed to keep the distance in line-following-mode and side-formation-mode, as based on that brakes are applied to the corresponding robot, by reducing its speed. This also happens inside this function. The angle of the robot is calculated inside this function. For this, the robot should be equipped with both the spirals. This angle is needed for the shrimp tracking. The way this is done is using the IDs of the spirals, putting spiral ID 1 on the nose of the robot and spiral ID 2 on the tail. This way the angle of the robot can be calculated in relation to the camera mounted over the arena. The following equations is used:

$$\delta x = x_{nose} - x_{tail} \tag{1}$$

$$\delta y = y_{nose} - y_{tail} \tag{2}$$

$$\theta = tan^{-1}(\frac{\delta x}{\delta y}) \tag{3}$$

An explanatory figure can be seen in Fig. 6.

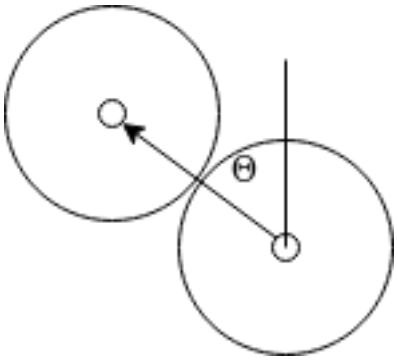

Figure 6: Calculation of pose

2.4.2 shrimp_cb

The next function is the shrimp tracking function. It is also a callback, which listens to the shrimp topic. From this one it gets the position of the shrimp. It then calculates the direction it has to go and the distance it has to go. Unless a new coordinate is sent in the meantime, it also reaches the corresponding point on the table. As soon, as new coordinates are received, it tries to go to those.

2.4.3 feedback_cb

The feedback callback function listens to the feedback topic. Only the controllers can post to this one. It is just used for communication between the robots and therefore only used by the task of changing the leader. This can only be started by the command sender of the leading robot. The action callback of this one then sends a message on the action callback to the following robot. The following robot will then send either "OK" or "NOTOK" to the feedback and states within this message, which robot should be the receiving one. "OK" means, that a switch can be done and the leading robot will then be stopped by feedback callback. If it receives "NOTOK" it remains in exactly the same state as it is now and prints, that the change of the leader cannot be done right now. Once the following robot has overtaken the leading one, he changes it state to leading and sends a "DONE" on the feedback topic. The previous leading robot then changes it state to following and is accelerating again to follow the new leader.

2.4.4 action_cb

The last important callback of the controller is the action callback. This listens to the action callback and therefore receives the information from the commandsender and where needed from the other robot. It is basically used for controlling the different movements of the robot, or setting the robot into the correct mode. The decision what is done is based on the message, but before

anything is done, it is checked, if the message is meant for this specific robot and only then processed.
The first important task of the action_cb is to do laneswitching. Therefore, it looks on the message it received and then calls the sends the corresponding value to the lineswitch topic of the arduino. Befor that, it needs to stop the linefollowing, if it is in one of the autonomous modes, and afterwards activate it again by posting speed values to the linefollow topic.
It also provides the part set role. This does the leader switch. It first looks from whom the message was received. If it was from the command sender, it will check if the robot is the leader and then either sends the leader change message to the follower, or will tell the user, that a leader change cannot be done, as this robot is not the leader. So only the leader can initiate a change of leader. The left part for the leader in the leader change is handled by the feedback callback. If it is not the leader, it will inform the leader, that a change can be done, or that it cannot be done via the feedback callback. If it can be done it will accelerate and drive for some time, until it is in front of the other robot. It then informs the other robot, that the change was done and updates the robots state.
With set speed the robot can be moved forward in manual mode, by applying the received speeds to the motors. Therefore, it can also move the robot backwards, to the left and to the right.
Set mode allows the user to set the mode of the robot. It can either be set into listening mode, shrimp mode, line following mode, or side formation mode. The listening mode hereby represents the manual mode. The line following mode and the side formation mode represent the autonomous mode. They only differentiate as more robots would either drive behind, or next to each other. The shrimp following mode allows the robot to follow the shrimp. They can be changed at any time by the commandsender, except if the robot is in a critically movement at that time.
The last important part of the action callback is the intersection. Here the robot will be stopped. Then it will be turned by 90° into the intersection and follows this one with linefollowing until the end. Last it turns out of the intersection by 90° again.
All the other parts in this function were only used for testing, or are not used at all. Some message values were already reserved for future applications.

2.5 Shrimp tracker

The image analysis part in charge of tracking the shrimp is built using image analysis library openCV [2]. OpenCV consists of many methods useful for image processing. Before all the tracking of the object is conducted the image from the camera is adjusted to mimic the size of the map. The first part of the shrimp tracker consists of converting the RGB-values from the raspberry-camera to HSV-values. This is because HSV is more suitable for object detection than RGB-values. An easy way to look at it is that hue represent what color, saturation is the amount of white that the color is mixed with and value represent the amount of black the color is mixed with. This is very applicable to the problem at hand for this project, as the shrimp mostly consists of a uniform color and the background is mostly white. Therefor the first problem to solve is to remove all the colors from the image except for the color of the shrimp. We used a lower boundary of $(H, S, V) = (0, 21, 75)$ and a upper limit of $(H, S, V) = (20, 150, 220)$. These values have to be adjusted on a to day basis due to different impact of light and shadow.

The next part is to remove holes and imperfections from the image to further improve the object detection. To do this we use erosion and dilation. Erosion and dilation will give different result based on which order it is conducted. To make something smaller you use erosion and dilation is to make something bigger. Using erosion and the dilation will result in the removal of small objects while dilation and then erosion will result in the removal of small holes. What we used was to first erode and then dilate to remove any possible objects still found in the image.

Following this approach the next part is to take the contours of the remaining objects still left in the image. Based on the largest contour in the image we make the assumption that that will be the object we are tracking, in our case the shrimp.

The remaining part is to publish the center of the contour in (X, Y) through ROS. This is done once every second so it's then up to the robot to try to get to the position on the map that corresponds to the center of the contour in the image.

For visualisation the image fitted to mimic the environment is shown to the user with a yellow bounding circle around the tracked object, in our case the shrimp. From the center of said circle a red line is drawn to visualize the path the shrimp has taken. This can be shown in Fig. 7. The shrimp started out in the middle and then went to the bottom right after that it started running laps around the arena, which can be seen with all the red lines. The red lines fade after a couple of seconds so the snapshot shown in Fig. 7 shows how fast the shrimp can move but still be tracked.

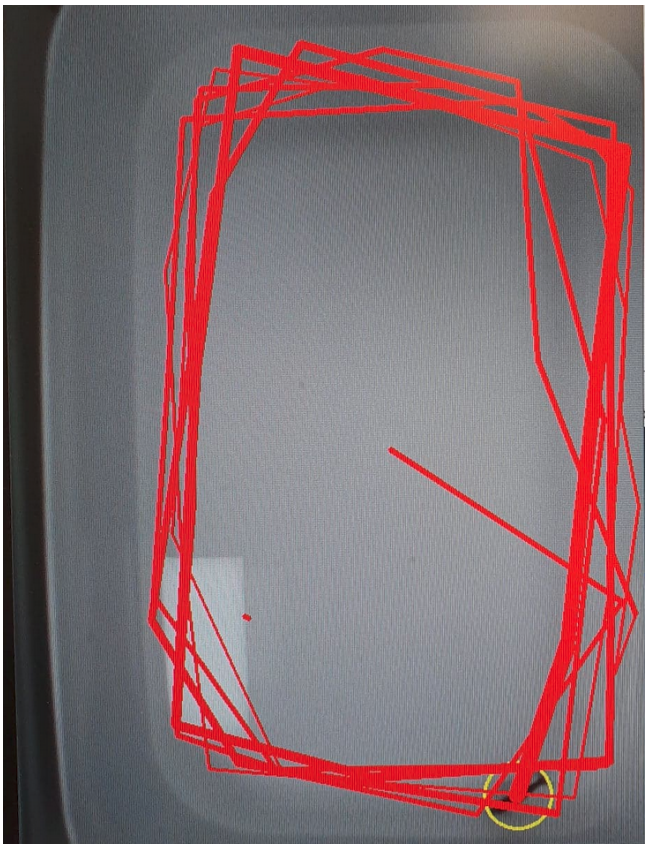

Figure 7: The image from the raspberry-camera fitted to mimic the environment. Shrimp in frame surrounded by yellow bounding circle with a red line drawn from the center of said circle.

2.6 Final Model

Our final software model differs slightly from our planned one seen in Fig. 5. The image processing part remains an own part, which is the shrimp tracker now. The track sensing runs inside the arduino code. It could not be moved anywhere else, as its response would be to bad then and we would move outside the track, when we are following the lines. As the encoders were quite bad and too many information was lost to rely on the odometry, this part was completely removed in the final version. The communication is no software part on its own, as all the parts need to communicate with each other. Therefore, it was moved to the individual software-parts.
Fig 8 shows the communication between the different parts of the project. For the whole communication ROS was used, as it was the easiest and best working version. Initially we used an own serial communication between the controller

and the arduino. There a lot of information was lost and it was therefore replaced by rosserial.
The communication between the raspberry and the shrimp tracker also just uses one rodtopic, which is named /robotname_shrimp. Rostopics also allow the communication to the Server and between the controllers.
The different topics, which were used, can be seen in Fig. 8.

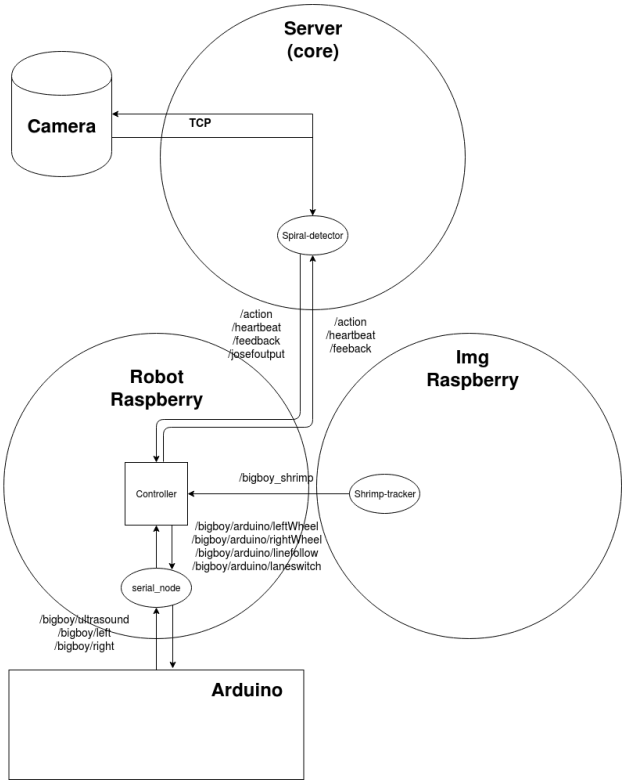

Figure 8: Communication overview

3 Results

The following part explains our result with text. All of our results are hard to show in text therefor we provide a link to a drive where videos of the results can be found [4].

3.1 Driving tasks

Basic driving tasks were completed early such as lane following and turning on intersections. The biggest hurdle was the switching of lanes. Problems arose with lanes being of different width and robots needing different power to switch lane depending on where the robot was positioned in the lane. This was finished when the values was tuned correctly. When these parts was on the arduino alone they worked correctly but gave us problems in other parts of the program. After the remake of the code it works correctly with a good flow between the different parts of the code.

3.2 Cyborg tasks

The shrimp tracker was completed early on with the basic cyborg part. This part consisted of a video with the shrimp in where it swam around in a white box. If the shrimp was on the right side of the image then the robot span on the spot to the right and if the shrimp was on the left side of the box the robot span on the spot to the left.
When implementing the extended cyborg part we used live footage of the shrimp and the robot was able to go towards where the shrimp was in image when the message was sent. As shown in Fig. 7 the shrimp was running laps around the white box which made it impossible for the robot to follow the shrimp 1-to-1 but as soon as the shrimp stood still the robot drove to the corresponding part of the arena to where the shrimp was in the image.

3.3 Platooning tasks

The final part of the project was the platooning tasks, side-formation and follow the leader was solved pretty quick since they didn't require much of communication between the robots. The hardest part for platooning was the inter-communication between the robots for the changing of leader. We solved the problem of changing of leader in the side-formation mode.

3.4 Requirements

We fulfilled all of the requirements set out in the beginning, which can be seen in Fig. 3 and Fig. 4.

[4] `https://drive.google.com/drive/folders/14NMTmbMo2ZO8c_-W1nDg7VdcChlM1xS8?usp=sharing`

4 Problems and discussion

While working on the project, two major problems occurred. The first one was, that the WiFi in the project room often stopped to work properly. That also impacted the other groups. This problem was solved by replacing the WiFi router with another one.
Our second problem was the communication between the raspberry pi and the arduino board. For this communication we first implemented a serial communication, that sends strings with the commands and attributes over the USB connection from the controller to the arduino and the other way round. In this version of the communication, packets were quite often lost and it was not that stable. This lead the robot to crash and not react anymore, or just a few times. Therefore, we came up with the solution to implement rosserial for this part of the communication. This allowed us to communicate with the arduino over rostopics, but we also had to redo nearly the whole code running on the arduino. As we had to do this, we also moved nearly the whole "intelligence" to the controller.
Also some smaller problems were seen while working on the project. One of the problems, that occurred, was, that on our second robot the motors were uneven. This lead the robot to move to one side, where it should move straight. This was solved by simply adding different power to the motors.
We also tried to move the code for the linefollowing from the arduino to the raspberry. This lead the robot to move over the lines and therefore the linefollowing did not work anymore. The reason for this behaviour was, that the communication took to long. The arduino had to send the data from the optical sensors via a topic to the controller and then the controller had to send the correct actions to the two topics of the motors. Because of that experience, every part of the code, which needs immediate action on detection of the optical sensors, was moved to the arduino again.
The combination of bad encoders and the rather slow communication between controller and arduino lead to the result, that we had to remove our odometry. An estimation of our position, based on the encoder values, did not work properly. The rosserial is limiting the amount of possible topics for the communication between arduino and controller, as it takes a lot of memory from the arduino and adding more topics caused stability issues.

5 Workload

Throughout the project the group have been using git and GitHub for version control and collaboration. This repository is publicly available at [5] where each project member is named Anguse(Harald), LazioSweden(Anton) and Simbru96(Simon).

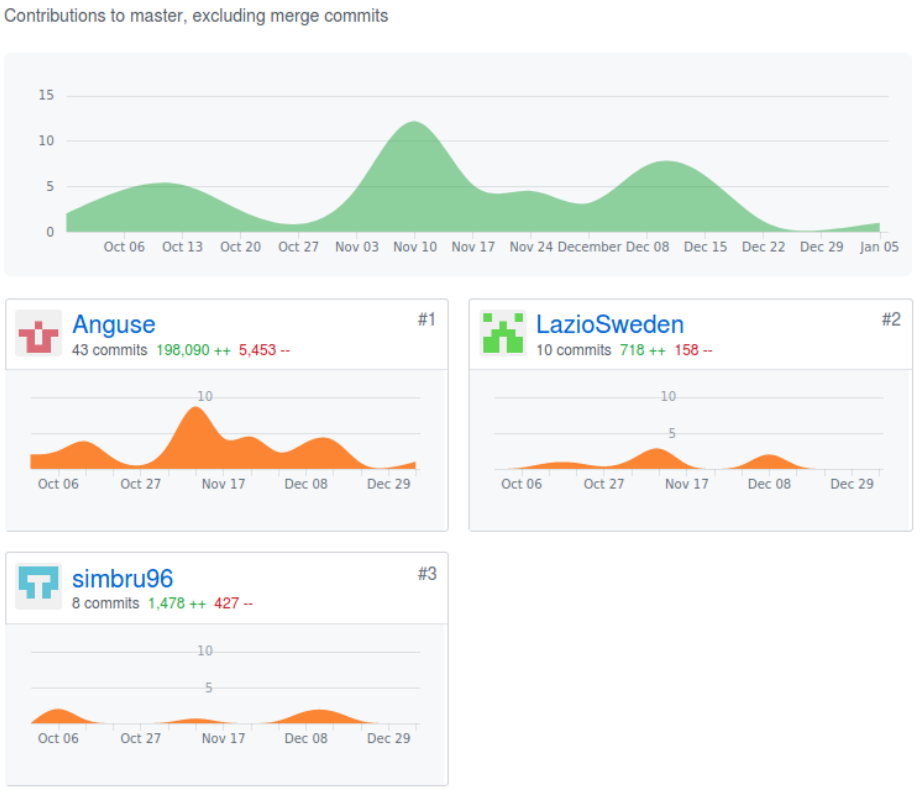

Figure 9: Workload

[5] https://github.com/Anguse/deis-labs

6 Conclusion

We successfully obtained the results needed for grade 3, we had hoped for a higher grade but due to the course running in parallel with the master thesis for the final half time sadly became a problem but we are satisfied with our work. Initially we thought that the shrimp tracking would be the part that would take the most time but it became the other way around.
The shrimp tracker was the first task to become completed, the tracking algorithm was completed in the first half of the course while the part for the robot to follow the shrimp was completed once the solution of using two spirals on top of the robot to calculate the orientation instead of using odometry completed the shrimp tracker.
The driving tasks was the second part to be completed, since we had to complete them twice due to the change of approach.
Platooning was the hardest part of the course with the complexity of the different platooning task taking up the most time.
For future work we would have liked to implement more cyborg terraforming tasks, which was required for a higher grade. Looking back a different approach for work distribution would have been preferred with controller taking the largest amount of time. We would also have liked to worked more with other groups but since we all had different approaches for communication between the robots with some groups using ROS and others implementing LCM made it troublesome when on a deadline.
The biggest take-away from the course was learning ROS, which became both the biggest asset but also posed some problems which we haven't seen before. We think that for future iterations of the course more time on the lectures should be focused solely on ROS.

References

[1] Carl FR Weiman and George Chaikin. Logarithmic spiral grids for image processing and display. *Computer Graphics and Image Processing*, 11(3):197–226, 1979.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.