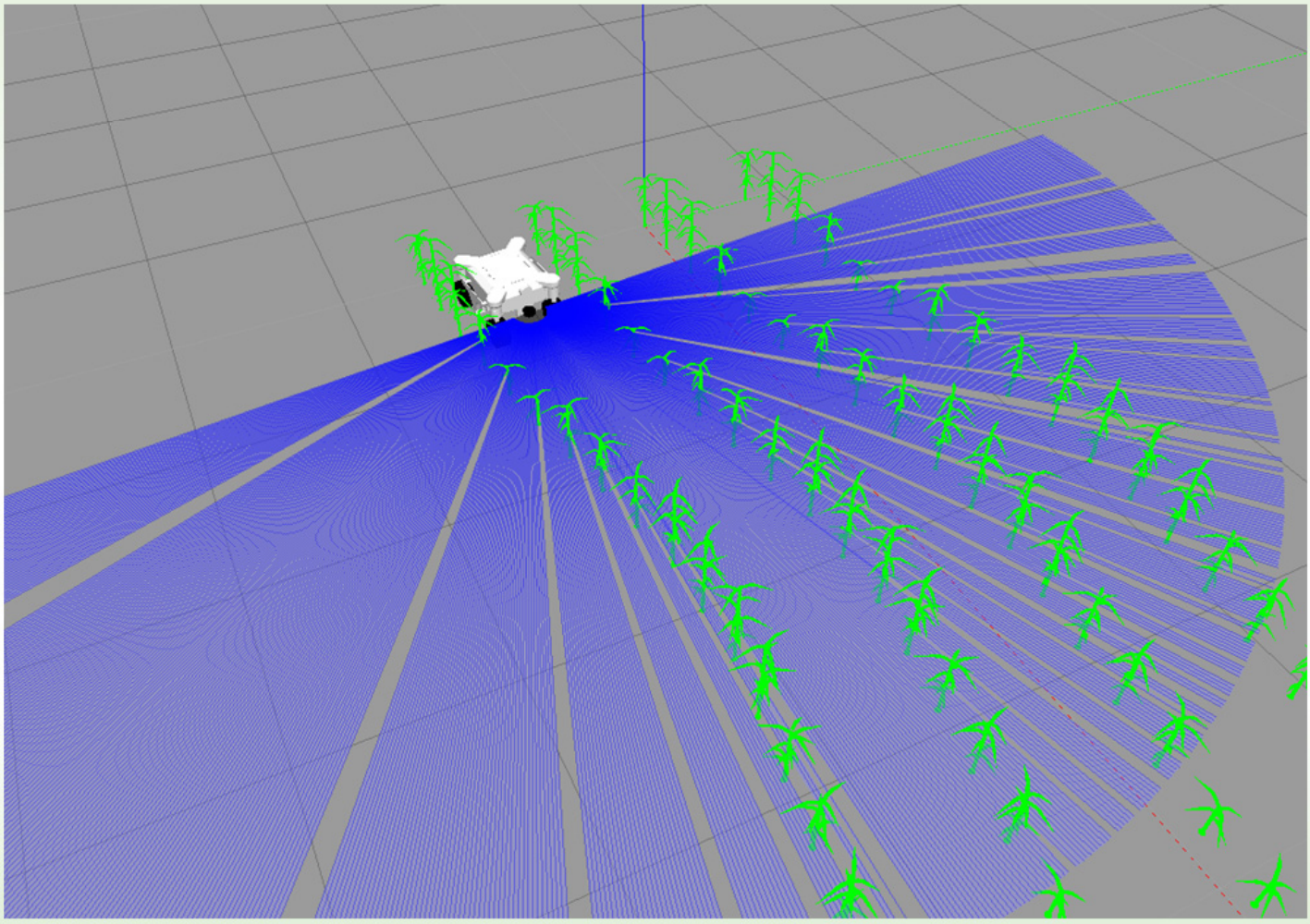
www.fieldrobot.com/event

Field Robot Event

International Contest

18th edition

Proceedings 2021



DLG Feldtage
Meet the crop professionals

Proceedings of the 18th Field Robot Event 2021

International Contest on Virtual Fields
June 8th – 10th, 2021

Conducted in conjunction with the DLG-Feldtage / DLG Field Days
Broadcasted by DLG-Connect

Editors:
Hans W. Griepentrog, Sam Blaauw, Thijs Ruigrok, David Reiser

Date: January 2022

Publisher: University of Hohenheim
Technology in Crop Production (440d)
Stuttgart, Germany

Contact: Hans W. Griepentrog

E-Mail: i440d@uni-hohenheim.de

File available on this webpage: http://www.fieldrobot.com/event

Sponsors

WAGENINGEN
UNIVERSITY & RESEARCH

UNIVERSITÄT
HOHENHEIM

Preface

Due to the corona pandemic we were forced to cancel the FRE 2020. It was a hard decision, but in almost all concerning countries the situation for preparing for the event was extremely difficult. Traveling within Europe was also a very difficult matter. However, in 2021 we were very happy to inform our FRE community that we will conduct a Field Robot Event. Although the unfavorable situation continued we said we do not want to give up and after many discussions we decided to plan and offer a virtual and remote event in 2021.

Modelling and simulation are an important issue and topic within robotic sciences. Therefore, we decided to conduct the contest in a ROS/GAZEBO environment. Some teams had some experience already but others none. Hence, we tried to guide and support those teams and provided information how to prepare for the event.

The tasks for 2021 were quite similar as for a physical event: Basic and advanced navigation, two agricultural applications and the freestyle. In order to make it easy for some teams to start we provided a standardized virtual robot model including sensing which should be used by all teams. Having the same machine model, the emphasis was on programming a code to control the virtual machine in a particular virtual environment. The environments were defined and realized for each task by the organising team.

However, during planning the event many expected that the performance of the standardized machine would not differ much between the codes provided by the robot teams. But the machine behaviors during the live event were indeed very different in reacting on the same environment and its components. Conclusion was, that the code had a huge influence. An international jury committee was as usual assessing all task performances and decided about the awards.

The general feedback about the virtual event in 2021 was very positive from the public and from the participating teams. The internet broadcasting via the DLG connect platform had more than 500 visitors per day. Watching the robot actions closely and listening to comments from robot experts were obviously attractive to visitors. But after a final discussion, there was a consensus that having a physical event with all the personal communications and interactions should be aimed at again.

Thanks to all sponsors. Special thanks to Sylvia Looks from CLAAS Foundation, Peter Groot Koerkamp from EurAgEng, Freya von Czettritz and Andreas Steul from DLG.

The editors and organizing committee

Hans W. Griepentrog, Sam Blaauw, Thijs Ruigrok, David Reiser

January 2022

Index

Participating Teams

Task Description

0. General information

Remark: The organisers tried to describe the tasks and assessments as good and fair as possible, but all teams should be aware of that we might need to modify the rules before or even during the contest. These ad hoc changes will always be decided by the jury members and cannot beyond disputed.

The organisers expect that the event is held in an “Olympic Manner”. The goal is a fair competition, without any technological or procedural cheating or gaining a competitive advantage by not allowed behaviours and technologies. Any observed or suspected cheating should be made public immediately to ensure fairness for all. The jury members are obliged to act always as neutrals. All relevant communication will be in English. For pleasing national spectators in the country where the event is organised, the contest moderation could partly switch to the host language.
All participating teams are obliged to contribute to the event proceedings with an article describing the machine and software in more details and their ideas behind it or development strategies in general. This publication is perceived well within our agricultural robot community. Furthermore, this shall document the state-of-the-art and make it easier for new teams to come to the event and being able to cope with the challenges. Increasing knowledge and skills for developing and operating agricultural robots is an overall aim of the event.
Furthermore, each registered team must provide an introductory video of max. 2 min time about team, machine, strategy, institution and country. This will be shown during the contest as well as for the FRE website.

General rules

Competition environment and conditions

The competition will be conducted virtually in a simulation of ROS Gazebo. It is an open-source 3D robotics simulator. Gazebo supports codes for sensor simulation and actuator control. It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It can model sensors that “see” the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc. Gazebo has been used as the simulation environment for a number of technology challenges and competitions like DARPA Robotics Challenge, NASA Space Robotics Challenge, Virtual RobotX Competition and others*.
The operating system and ROS version used are fixed to avoid compatibility problems (Ubuntu 18.04, ROS-melodic).
There will be a specific competition environment provided for each task. Furthermore, for the development and testing of the robot software there will be environments published via links on the FRE webpages that already give an impression about the challenges and features

in each task. The final and beforehand not published contest environments will vary (!) but will be in principle the same as the test environments. Their specific properties will be defined by the organisers.
A machine model for the tasks 1 and 2 to be used by all teams will be provided by the organisers. It is compulsory for the teams to use this machine and model in task 1 and 2. This requirement shall ensure fair conditions for all teams, because the focus shall be on controller software including sensing and converting perception information into robot navigation or motion. For the other tasks 3, 4 and 5 the teams are welcome to use their own virtual and real machines but teams are also welcome to use an extension of the standard model.
Prior to the contest the organisers distribute Docker containers. One to run the competition environments in a Gazebo simulation, another to integrate teams robot code to. (picture 1 at the end of this text) These two containers communicate, allowing just the exchange of sensor messages and speed commands.
During the contest teams must have uploaded their robot container to the organisers one hour before the task starts. The organisers need this time to prepare for the setup of the task run.
During conduction of the task the teams robot container will communicate with the original Gazebo container. Object position information provided by gazebo will be blocked and cannot be used for the competition.
Five tasks will be prepared to challenge different abilities of the robots e.g. in terms of positioning, navigating, sensing and actuating in a direct or indirect relation to agricultural applications.
The scoring, ranking and awarding will always be team and not machine related. Teams are not allowed to come with more than one machine or code per task.
The simulated crop plants in the environments will be maize with a mean height of 30 – 40 cm and leaves pointing to random directions from plant to plant – each plant might look different or even be dynamic in terms of random leave motion. There will be 11 rows of equal length. The mean row width will be 75 cm (varying 70-80 cm) and mean intra-row spacing 16.7 cm (plant density 8 plants/m2). The plant spacing within the rows will also realistically vary.
The drive paths of the robots shall be between the crop rows and not above rows.
The use of GNSS sensor packages is not allowed except for the Free Style Task 5. The focus for the other tasks in terms of localisation shall be on relative positioning and related sensor-based behaviours.

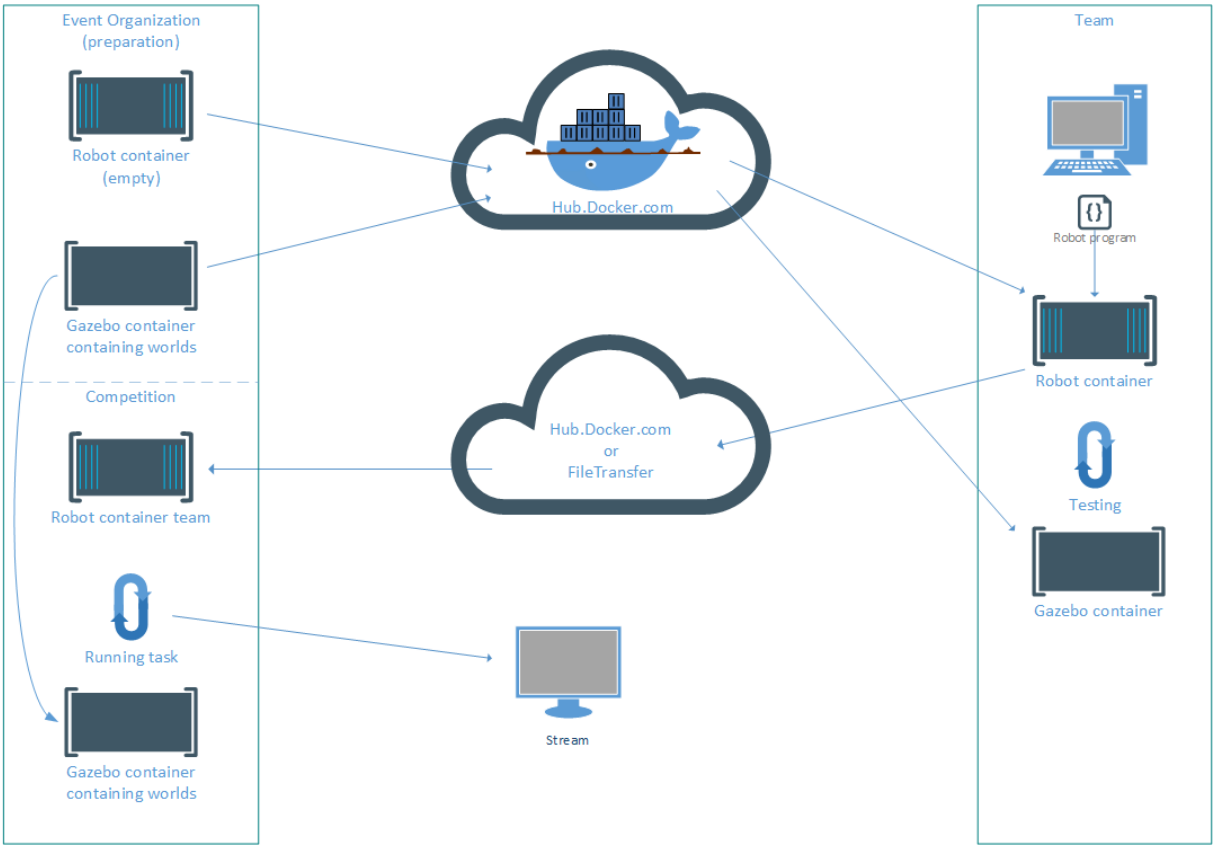


Figure 1 – Schematic overview of the use of containers

General robot requirements

Autonomous Mode

All robots must act autonomously in all tasks, including the freestyle, if possible. Therefore, teams cannot interact with their machines during the run.
There will be an automatic stop of the gazebo environment (freezing) after the run duration (e.g. 3 or 5 min). During the run there will be a screen display of distance and errors e.g. number of damaged plants. This shall also make it more exiting for the participants and the spectators. The final assessment or conclusion about e.g. penalty will be done by the jury.

Positioning and starting

Before the start the robot is placed in front of the beginning of the first and second row. The starting point will be a marked with a white line. No part of the robot is allowed to cross the white line at the start position. After the start signal, the robot must start within one minute. If the robot does not start within this time, it will get a second chance after all other teams finished their runs.

Start & Stop and Second Trial

There will be no possibility for the teams to use a stop button intervention during the run. But it is up to the teams to decide once to stop within the first row or within the first 30 secs of run. A new start e.g. with a slightly different starting position or orientation has to follow immediately.

Scoring and Awards

The performance of the competing robots will be assessed by an independent expert jury. Beside measured or counted performance parameters, also creativity and originality will be evaluated (e.g. task 5 free style). There will be an award for the first three ranks of all task 1 to 5, but only task 1 to 4 together will yield the overall competition winner. For the overall competition ranking points will be given as follows:

Rank	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	etc.
Points	30	28	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	etc.

Participation results in at least 1 point for the given task. No participation results in 0 points. If two or more teams have the same number of points for the overall ranking, the team with the better placements during task 1 to 4 will be ranked higher. There will be no negative point sum e.g. due to penalties during the task runs.

1. Task ‚Basic Navigation‘

General description

For this task, the robots are navigating autonomously through a maize field. Within three minutes, the robot has to navigate through curved (!) rows (picture 2 at the end of this text). The aim is to cover as much distance as possible. On the headland, the robot has to turn and return in the adjacent row. This task is all about accuracy, smoothness and speed of the navigation operation between the rows.
The starting is on the right side (first turn is left). This is not a choice of the team but of the officials. A headland width of 2 meters free of obstacles (bare soil) will be available for turning operations. The headland will be indicated by a fence or ditch or similar (3D object).

Virtual Field Environment

Random stones are placed along the path to represent a realistic field scenario. The stones are not exceeding 25 mm from the average ground level. The stones may be small pebbles (diameter <25 mm) laid in the ground and large rocks that protrude (maximally 25 mm) from the ground. In other words, typical outdoor abilities as defined by machine ground clearance and to climb over small obstacles are required.
There will be no gaps in row entries as well as at the end of the rows. The ends of the rows may not be in the same line.

Rules for robots

Each robot has only one attempt. For starting, the robot is placed at the beginning of the first row without crossing the white line. The maximum available time for the run is 3 min.

Assessment

The distance travelled during task duration is measured. The final distance will be calculated including especially a bonus factor when the end of the field is reached in less time than 3 min. The final distance including a bonus factor is calculated as:

S_{final} [m] = $S_{corrected}$ [m] * 3 [min] / $t_{measured}$ [min]

The corrected distance includes travelled distance and the penalty values. Travelled distance, penalty values and performance time are measured by the jury officials.

Crop plant damage by the robot will result in a penalty of 2% of total row length distance in meter per damaged plant. (This year example 10 x 5 m = 50 m max. distance, means a penalty of 1 m per damaged plant.)

The task completing teams will be ranked by the results of resulting total distance values. The best 3 teams will be rewarded.

Points for the overall winner will be given as described under awards.

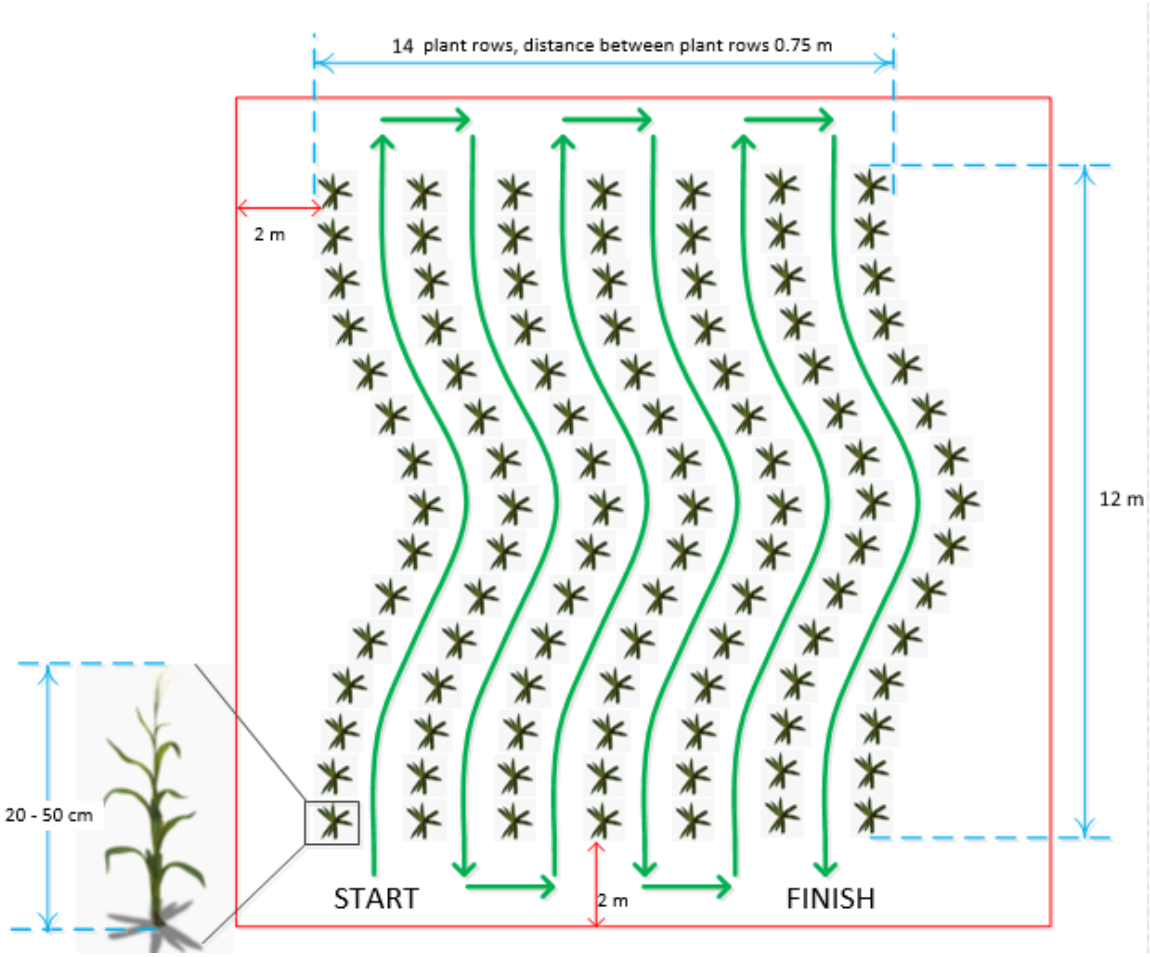


Figure 2 – Concept of field structure for task 1 (example)

2. Task ‚Advanced Navigation‘

General description

For this task, the robots are navigating autonomously. Under real field conditions, crop plant growth is not uniform. Furthermore, sometimes the crop rows are not even parallel. We will mimic these field conditions in the second task.
The general rules here are the same as in task 1. No large obstacles in the field, but more challenging terrain in comparison to task 1.
The robots shall achieve as much distance as possible within the duration of the task while navigating between straight (!) rows of maize plants, but the robots have to follow a certain predefined path pattern across the field (picture 3 at the end of this text).
The robot must drive the paths in a given order provided by the organisers. The code of the path pattern through the maize field is done as follows: S means START, L means LEFT hand turn, R means RIGHT hand turn and F means FINISH. The number before the L or R represents the row that has to be entered after the turn. Therefore, 2L means: Enter the second row after a left-hand turn, 3R means: Enter the third row after a right hand turn. The code for a path pattern, for example, may be given as: S – 3L – 2L – 2R – 1R – 5L – F.
A file with the code of the path pattern to be followed is made available to robot at the start.

Virtual Field Environment

Random stones are placed in the same way as within task 1.
Additionally, at some locations, plants will be missing (row gaps) at either one or both sides with a maximum length of 1 meter. There will be no gaps in row entries as well as at the end of the rows. The ends of the rows may not be in the same line.

Rules for robots

Each robot has only one attempt. For starting, the robot is placed at the beginning of the first row without crossing the white line. The maximum available time for the run is 3 min.

Assessment

The distance travelled following the given path during task duration is measured. (As soon as the robot leaves the specified path, the distance measurement will stop.) The final distance will be calculated including especially a bonus factor when the end of the field is reached in less time than 3 min. The final distance including a bonus factor is calculated as:
Sfinal [m] = Scorrected [m] * 3 [min] / tmeasured [min]
The corrected distance includes travelled distance and the penalty values. Travelled distance, penalty values and performance time are given and measured by the jury officials.
Crop plant damage by the robot will result in a penalty of 2 % of total row length distance in meter per damaged plant. (This year example 10 x 5 m = 50 m max. distance, means a penalty of 1 m per damaged plant.)
The task completing teams will be ranked by the results of resulting total distance values. The best 3 teams will be rewarded. Points for the overall winner will be given as described under Awards.

Figure 3 shows an example of how the crop rows and the path tracks could look like for task 2. Be aware, the row gaps and the path pattern will be different during the contest.

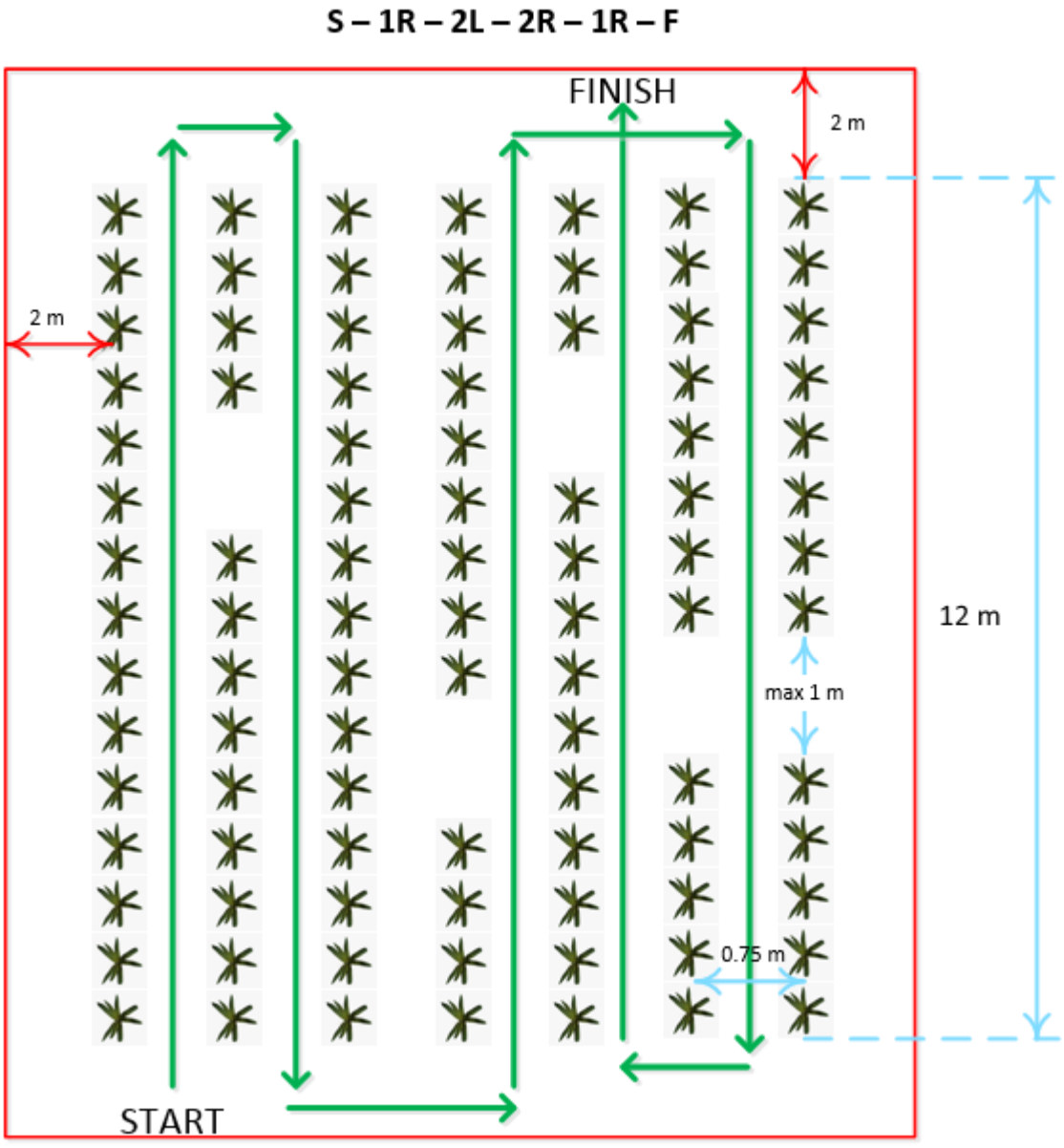


Figure 3 – Concept of field structure for task 2 (example)

3. Task ‚Field Mapping‘

General description

The robots shall detect objects as weeds and beer cans (example for waste) and map or geo-reference them. The coordinate system shall be locally in horizontal field dimensions. The reference point will be pillars with a QR code. Task 3 is conducted in an environment similar to task 2. Nevertheless, good row navigation is required. There will be nine (9) objects in total distributed across the virtual field.
The robot has to generate a file (*.csv) with classified objects and their coordinates relative to the given reference point. Each object should be reported on one line in the file including the coordinates x and y in horizontal plane in meters with 3 decimal points and its kind (table 5).

Virtual Field Environment

Objects are realistic weeds and cans e.g. of beer with different brands and colours. The objects will be placed randomly across the field. So they can be in between and in the rows. No objects are located on the headlands.

Rules for robots

Each robot has only one attempt. For starting, the robot is placed at the beginning of the first row without crossing the white line. The maximum available time for the run is 3 min.

Assessment

The jury assesses the detection during the run:

Detected object during run (true positive)	5 points
Detected object during run (false positive)	-5 (minus) points

And assesses the classification and accuracy of mapped objects:

x: Euclidean distance to object of the same kind*	points
$x \leq 2$ cm	15
2 cm $< x \leq$ 37.5 cm	15.56 – 0.2817 * x
$x >$ 37.5 cm (false positive)	-5

*(distance error to the nearest object of the same kind)

Crop plant damage by the robot will result in a penalty of 4 points per plant.

The task completing teams will be ranked by the number of points as described above. The best 3 teams will be rewarded.

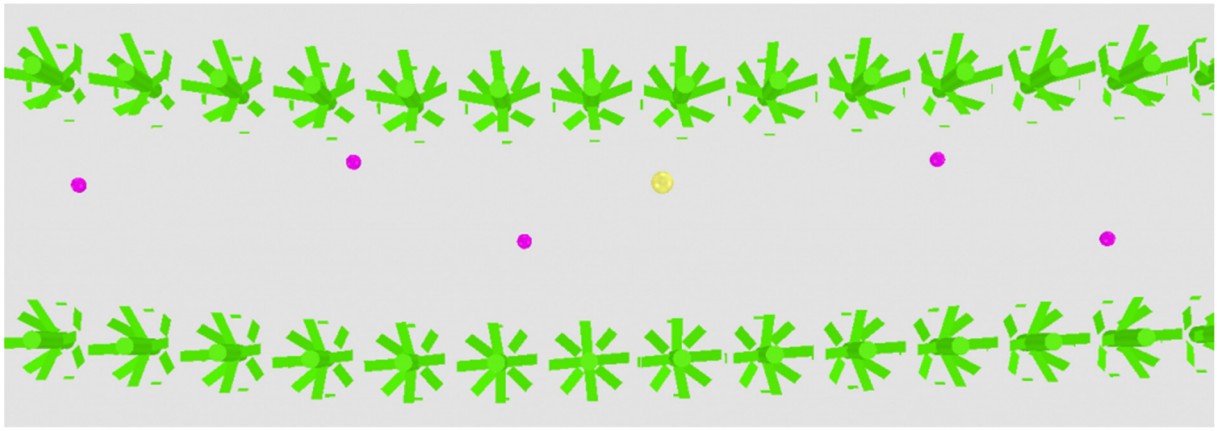


Figure 4 – Example of weed and obstacle locations for task 3 and task 4.

X	Y	Type
2.645	3.583	weed
3.804	3.537	weed
4.894	3.562	waste

Table 5 – Example for a map file, recorded in Task 3

4. Task ‚Removal of Objects‘

General description

For this task, the robots are navigating autonomously. The robots shall remove 10 objects and place them outside the crop area on the headland. Task 4 is conducted on the area used in task 2 and task 3 with straight (!) rows. The exact relative position coordinates of the objects will be provided beforehand as a realistic scenario using a drone with a positioning system and a camera. The object type will also be provided. The waste / cans (5 objects) should be placed on one headland and the weeds (5 objects) on the other headland. There will be an indication of the field reference points using a QR code and a pole.

Virtual Field Environment

The weeds and beer cans will be randomly distributed across the field. So they can be in between and in the rows. No objects are located on the headlands. The weeds and cans will not be connected to the soil, but will differ in size and mass.

Rules for robots

Each robot has only one attempt. For starting, the robot is placed at the beginning of the first row without exceeding the white line. The maximum available time for the run is 5 min.

Assessment

The Jury or the simulator registers and assesses the number of objects where they are remaining after the run:

Object picked up	3 points/object
Object delivered to the right headlands	6 points/object
Object delivered to the WRONG headland	3 points/object

The robot is allowed to push the object to the headland, but without a clear act of picking up, it will only earn points for the delivery. Crop plant damage by the robot will result in a penalty of 2 points per plant. The total travelled distance will not be assessed. The task completing teams will be ranked by the number of points as described above. The best 3 teams will be rewarded.

5. Task ‚Free Style‘

Description

Teams are invited to let their real robot perform a freestyle operation at their home institution. The explanation as well as the performance shall be transmitted online via internet to the jury and the spectators. One team member has to explain the idea and the machine. Comments during the robot's performance are also welcome.
Creativity and fun are required for this task as well as an application-oriented performance. The freestyle task should be related to an agricultural application. Teams will have a time limit of five minutes for the presentation including the robot's performance.

Assessment

The jury will assess by points P the

- P_1 : agronomic idea (originality)
- P_2 : technical complexity
- P_3 : robot performance

Points P will be given from 0 (insufficient) to 10 (excellent) for each criterion (P_1, P_2 and P_3). The total points will be calculated using the following formula:

Pfinal points = $P_1 + P_2 + 2\ P_3$ (double weighing on performance)

The task 5 is optional and will be awarded separately. It will not contribute to the contest overall winner.

Robots

BETEIGEUZE - KAMARO

Johannes Bier [1], Thomas Friedel [1], Johannes Barthel [1],
Edvardas Bulovas [1], Leon Tuschla [1]

[1)] *Kamaro Engineering e.V., Germany*

1 Introduction

We enjoyed helping and supporting to create the competition environment for this year's online FRE. We contributed a map generator, and some 3D artwork that we had previously used in our own simulation. We are glad we were able to convince the organizers to use realistic plant models, so that navigation and image recognition tasks could be as close to a real-world scenario as possible, given the resources. Having a standardized robot model was probably a good rule, so that the barrier of entry was lower for teams with no prior experience regarding simulation, and nobody could use an unrealistic virtual robot that would have made the tasks easier. The Jackal robot is quite like our own robot "Dschubba" that we used at the FRE in 2019, so it was comparatively easy to adapt our existing software stack.

2 Navigation in a virtual field

The robot software is based on the ROS middleware and is this implemented as multiple ROS nodes. Our main components are:

- a *row_crawl* node that performs in-row reactive driving and end-of-row detection,
- a *turn* node that takes control when switching from one row to another,
- a *state_machine* node,
- an *object_detection* node for weed/litter detection (Task 3).

2.1 Row Crawling

The only sensor used for traversing the row is the laser scanner. For driving through the rows, we search the filtered point cloud for an obstacle-free cone of configurable length, usually 1 – 2 m, in the filtered laser scan. The search is started from the robot's driving direction. The found cone of free space is visualized by two red lines in Figure 1: RViz visualization of our row_crawl node. This gives us a line in the direction of the row. We shift this line to the left/right by an amount calculated from the ratio of the distances to the plants on the left/right. We clamp the distances to a maximum value to not be too confused by missing plants. These distances are calculated by growing a rectangle on the side of the robot until more than three points are inside it (the lime green rectangles in Figure 1). This helps to avoid taking "shortcuts" through the plants in curves. We then use a PID controller to steer towards the point on this line that is 1 m in front of the robot.

This algorithm is solely reactive and requires no concept of a field map. Notably, it controls the robot with only two different measures:

- the angular offset (controlling the robot orientation)
- the lateral offset to the crop rows on both sides

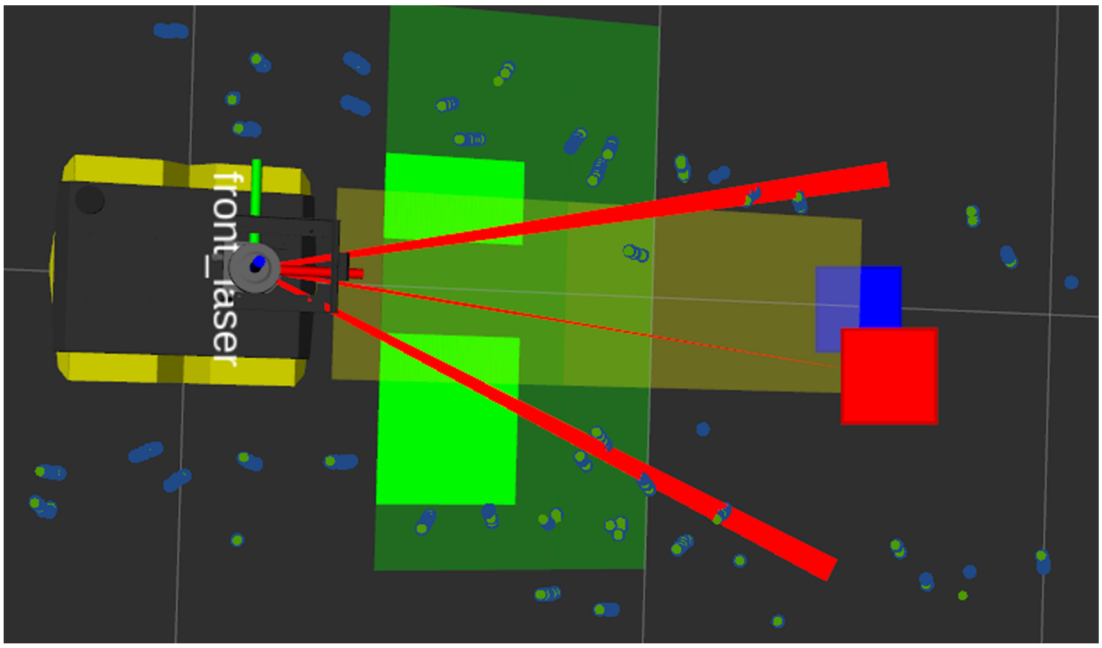


Figure 1: RViz visualization of our row_crawl node.

Our main robot “Beteigeuze”, for which this algorithm was originally designed, has two steerable axes and can therefore control both variables independently. For the simulated Jackal robot (and for our other robot “Dschubba”), which both use differential driving, we use an adapted version of the algorithm that compensates the side offset with an additional P controller to “oversteer” and correct the lateral offset using the driving direction.

One of the bigger challenges with this virtual Field Robot Event was the position of the laser scanner on the Jackal robot, where it is mounted rather high, compared to our own robots. For task 1 and 2, it was not allowed to change the robot model, and the high position of the scanner meant a lot of false positives due to leaves of the maize plants hanging into the rows.

By the time these proceedings are published, our *row_crawl* node will be available under an open-source license at [1].

2.2 Row Turning

To detect the end of the row, we count the number of LiDAR points in a wide box in front of the robot (the yellow box in Figure 1). If the number is below a certain threshold for a long enough time, the robot recognizes this as the end of the row and the *turn* node takes control via the state machine.

Our row-change algorithm heavily relies on the odometry provided by the *ekf_localization* node from the ROS package *robot_localization*. [2]

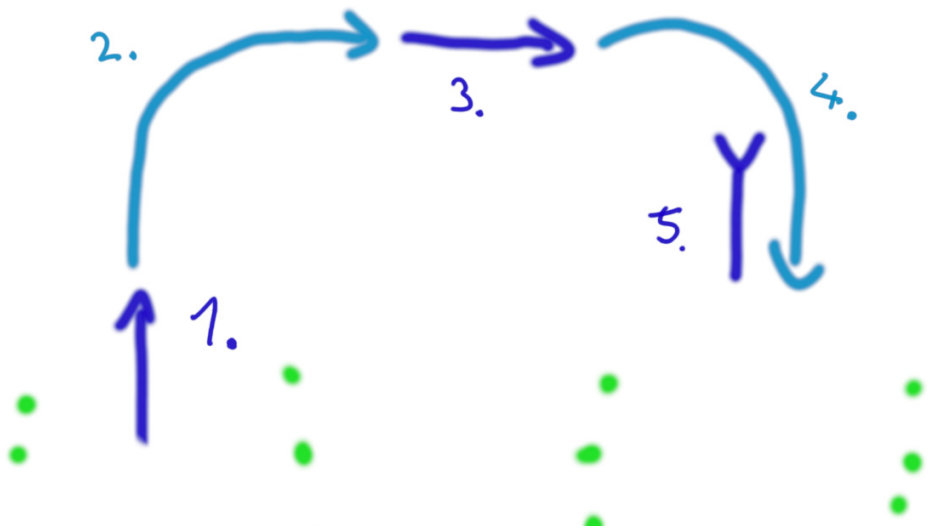


Figure 2: Sketch of the stages from our turn node.

The row change is divided into five sub-states (see Figure 2):

1. *move out*: To get a safe distance from the crops, drive a bit forward.
2. *turn out*: Perform a 90 ° turn into the desired direction.
3. *move along*: Drive in parallel to the field's edge for a distance depending on the number of rows and the average row width.
4. *turn in*: Perform a 90 ° turn in the same direction to face the field again.
5. *move in*: Aim for a certain distance perpendicular to the field edge compared to the starting point of the turn. This is required so that when the *row_crawl* node takes control, the robot is already inside the row and the plants on either side are clearly visible.

During all substates (except *turn out*/*in*), the robot compares its expected position with the position given by odometry relative to the turn's origin point. Any lateral differences are automatically corrected.

We also compare the expected row positions directly next to us with the laser scan during the *move along* substate and correct the route accordingly. However, only for long turns (number of rows greater than 4) do these corrections have any considerable effect, the odometry is good enough in most cases.

2.3 Task 1 – Basic Navigation

In task 1, we heavily optimized our *row_crawl* node on the sample map provided for this task. We found our algorithm to perform well on curved rows, and therefore aimed to drive as fast as possible – It paid off. We managed to traverse the whole field, roughly 130 m, in 2 minutes and 22 seconds with only 4 damaged plants. This performance was better than expected and good enough to land us on first place for this task.

2.4 Task 2 – Advanced Navigation

The big holes in the maize rows where a big problem for our cone finding approach. Often, the holes would be mistaken for the "right" way (see Figure 3). We tried to circumvent this issue by extending the length of the cone and ignore those with a wide

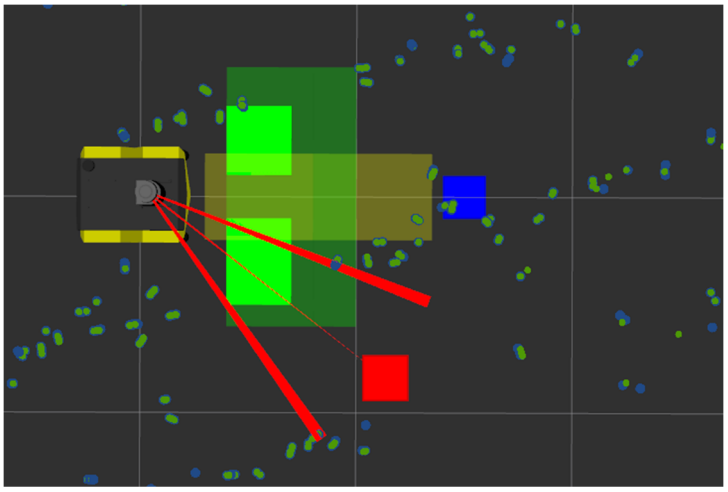


Figure 3: Failure case of the cone detection on a map generated for task 2, which means that there are up to 1 m gaps in the plant rows.

opening angle. Unfortunately, our strategy was not reliable enough and our robot failed on the first hole in the competition.

3 Applications in a virtual field

We were pleased with the organizers' decision to include realistic 3D plant models for the weed detection task. On the real field, robots will have to distinguish between multiple species of plants – all of which are green – in order to perform weeding, phenotyping and other tasks. Deep learning has been around for a few years now and seems to be the most applicable tool to tackle this challenge.

We use camera data and perform semantic segmentation on simulated camera images. Each pixel is assigned a class (background/maize/weed/litter). If a sufficient number of pixels in the current image belong to either the weed or the litter class, this is counted as a detection. As a side effect, the semantic segmentation approach also produces a visualization that shows how each part of the image is classified. This makes the model more comprehensible and reduces the "black box" effect that is often considered a downside of Machine Learning based approaches.

Our implementation is based on PyTorch, using a pretrained ResNet50. A 75 % / 25 % training/validation split was used on the dataset. For inference (i.e., running images through the detector on a robot or in a simulation), we export the model to the ONNX format and execute it on the CPU. Thus, no GPU was required for our code to run.

After the competition, we published all relevant code, training data, the trained model and comprehensive documentation, in the hopes that teams competing in future Field Robot Events can make use of this technology more easily, should the jury continue their path of demanding more realistic image recognition tasks. Please see our blog post [3] for more information and links to all provided materials.

3.1 Task 3 – Weed / object detection and mapping

While our detection worked quite well, we did not create a map and only sent signals due to time constraints. Because it was allowed to modify the robot model in task 3, we lowered the height of the lidar scanner to get better sensor data. This made our navigation in rows with holes more robust, however, our performance in the

competition was still tarnished by a lot of damaged plants. Our image recognition algorithm did however perform very well, providing a precise segmentation of maize, ground, litter and weed in each frame.

4 New innovations (real and virtual)

It has taken almost two and half years to redevelop our robot “Beteigeuze” to be more modular. It is now possible to connect plug-and-play modules to the robot without a hardware or software restart. A plugged-in module is directly powered from the robot’s power supply and can communicate with the robot or other connected modules over a CAN bus. As a middleware for this communication, the RODOS operating system is used. RODOS runs both on microcontrollers and Linux computers and uses a publish–subscribe messaging system. We bridge this to the ROS system running on the robot's main computer to achieve a unified publish-subscribe system where microcontroller firmware can seamlessly interact with Python and C++ ROS nodes through the standardized interface of ROS messages.

Built around a Nvidia Jetson Nano computer as the main controller, “Beteigeuze” has been designed with Deep-Learning-based image recognition in mind. We hope that we will be able to demonstrate its capabilities under realistic field conditions in a future non-simulated Field Robot Event.

4.1 Task 5 – Free Style

The goal of our team for the task 5 was to present the modularity of our robot “Beteigeuze” and to discuss the benefits of this extensible approach to agricultural robotics. The redesign of all electronic components around this modular concept took two years. Although no modules for usage in the field of agriculture were presented, the modularity of the robot has been shown through the successful connection of 2 plug-and-play modules.

We believe that future field robots – like today's tractors – will be a platform for a multitude of different tasks, requiring different implements for each. A modular electronic architecture is a prerequisite for any robot aiming to be such a versatile platform. We plan to present agriculturally relevant modules at a future Field Robot Event.

5 Conclusion

This virtual Field Robot Event had many welcome changes and new challenges, compared to previous events. We put great effort into the development of the virtual maize field the field generator used in this event. [4]

We are looking forward to presenting Beteigeuze Nova at a future offline FRE. We hope that Deep Learning based image recognition will play an increasing role in years to come.

6 Our Code

fre21_row_crawl: https://github.com/Kamaro-Engineering/fre21_row_crawl

fre21_object_detection: https://github.com/Kamaro-Engineering/fre21_object_detection

7 References

[1] https://kamaro-engineering.de/?p=1729 (2021-07-29)

[2] T. Moore and D. Stouch, A Generalized Extended Kalman Filter Implementation for the Robot Operating System, Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13), 2014, Springer

[3] https://kamaro-engineering.de/?p=1709 (2021-07-22)

[4] https://www.fieldrobot.com/event/index.php/2021/02/26/kamaro-engineering-joins-the-development-of-the-virtual-maize-field/ (2021-07-16)

8 Sponsors

BULLSEYE JACKAL - ROBATIC

Johan Blom [1], Thomas Frankes [1], Dorien Goudsblom [1], Aline Hazelaar [1], Frans Kemp [1], Herjan Riemens [1], Martijn Velhuizen [1], Sophie Wildeboer [1], Sam Blaauw [1,*], Rick van Essen [1,*], Thijs Ruigrok [1,*]

[1)] *Wageningen University, Farm Technology Group, Netherlands*
[*)] *Instructor and Supervisor*

1 Introduction

For all participants of team Robatic working with simulated environments like this was a completely new experience. So, the very first thing we did was teach ourselves how to work with a virtual machine. Luckily with our experience with Python we already had some knowledge that could help us familiarize ourselves with this software.

Once we figured out how to work with a virtual machine we had to make a difficult decision in regards to which robot we would use for task 3 and 4. Because normally we would have improved the software made by our predecessors. But the unique wheels of the original Bullseye can turn in place. Meaning that the navigation software that was given to us was not applicable to the Jackal. Not wanting to waste time on developing two different navigation software we chose to use the Jackal for all 4 tasks. And how we did all that is described below

2 Navigation in a virtual field

According to the task description of task 1, 2, 3 and 4 the robot must be able to drive through curved maize rows and straight maize rows with gaps of missing plants. Points are awarded for distance covered within a given time and a penalty is given for each damaged plant.

2.1 Task 1 – Basic Navigation

The standard Jackal was used as was dictated by the organizers of the competition

2.1.1 Problem description in row

According to the task description of task 1, 2, 3 and 4 the robot must be able to drive through curved maize rows and straight maize rows with gaps of missing plants. Points are awarded for distance covered within a given time and a penalty is given for each damaged plant.

Due to the obligation to use the provided standard Jackal for tasks 1 and 2, the control commands had to be changed compared to the Bullseye. The Bullseye last year had 3 inputs for driving, namely the x (m/s), y (m/s) and theta (rad/s) commands, which are respectively forward/backward, sideways and the steering angle (these commands are published with a Twist message, as *linear.x, linear.y* and *angular.z*). Because of its differential steering, the Jackal cannot move along the y-axis (perpendicular to the driving direction). Instead, it only needs 2 input commands to drive, namely x and theta.

The driving commands last year were calculated based on the data coming from the LiDAR. On the Bullseye, the LiDAR is mounted very close to the ground, so the LiDAR scans under the leaves and thus only detects the stems of the maize plants, which results in low-noise data. This is not the case with the Jackal, where the LiDAR is mounted at a height where the plants

the leaves and thus only detects the stems of the maize plants, which results in low-noise data. This is not the case with the Jackal, where the LiDAR is mounted at a height where the plants have a lot of leaves, which can be seen in Figure 1. Because the LiDAR is mounted higher at the Jackal, the result of the LiDAR data is that more data is generated with a larger variation. In addition to that, as the LiDAR is mounted in front of the Bullseye, it can perceive a situation like a curve in the row before the robot is in it, thus allowing it to pro-actively anticipate on it. On the Jackal, the LiDAR is mounted nearly in the middle of the robot, causing the robot to be located in a curve already, before it has the same data as the Bullseye would have before.

2.1.2 Requirements

Based on the rules and challenges provided by the FRE organisation, our problem description, and the results from last year we defined the following requirements:

The robot:

1. Can navigate through straight rows with a velocity of 1 m/s while damaging a maximum of 1 plant in 10 m of maize row.
2. Can navigate through rows with a *max_angle_variation*[1] of 0.15 [-] at a velocity of 1 m/s while damaging a maximum of 3 plants in 10 m of maize row.
3. Can navigate through straight rows with gaps of up to 1.5 m of missing plants on one side.
4. Can navigate through straight rows with gaps of up to 1.5 m of missing plants on both sides.

The navigation algorithm

5. Can only use the camera, LiDAR, IMU and odometry data
6. Can detect the location of the end of the maize rows
7. Can stop the robot when it is 1 meter on the headland

2.1.3 Method in row

Based on the requirements set in section 2.1.2 a new iteration of the in row navigation algorithm is developed. This navigation algorithm consists of 4 steps. In step one data from the LiDAR scanner is filtered. In step two, maize rows are detected in the filtered LiDAR data. Information from the detected rows is fed into the control algorithm which steers the robot. Additionally, the row navigation algorithm must detect the end of the row. A flow chart explaining the coherence of the in-row navigation algorithm can be seen in Figure 1.

[1] This is a parameter in the (old version) of the world generator, which is provided by the FRE organisation. The exact meaning and unit of this parameter are unknown to us, but it controls the severity of the rows curvature. Considering the world generator, it can be find on the git of the FRE organisation at https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event/tree/master/virtual_maize_field . The 'old version' is from before May 10, 2021, the 'new version' is from after that.

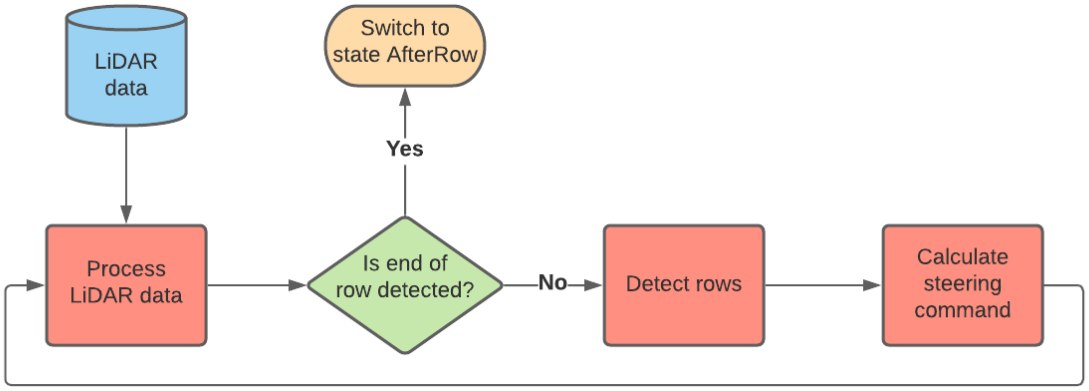


Figure 1: Global overview of the in-row navigation algorithm.

2.1.4 Problem statement headland

In order to navigate the maize field, the robot will have to transition from the navigation in the row to navigation on the headland, and back to navigation into the row. To achieve this transition, the robot has a state called TurnOut, for turning out of the row, and TurnIn, for turning into the row. Due to the differences between the Bullseye and the Jackal, these scripts had to be adapted. As the Jackal uses differential steering, it is not able to drive sideways. So, 180 degrees turn would be unnecessary. The Jackal should make a 90 degrees turn, so in the navigation on the headland the LiDAR data from the sides of the scanner can be used, as seen in Figure 2.

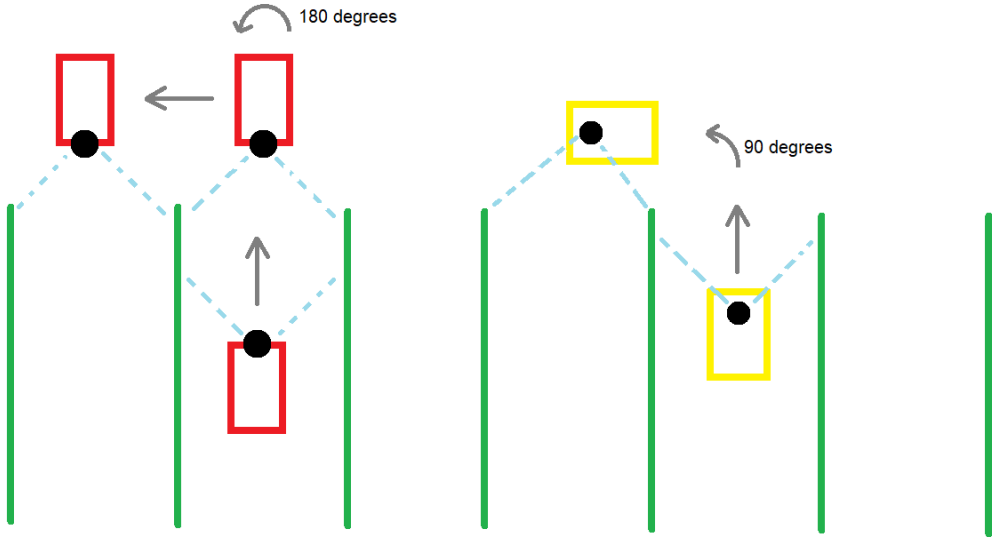


Figure 12: The different turning patterns of, on the left in red, the Bullseye and, on the right in yellow, the Jackal robots. The light blue dotted line represents the used LiDAR data.

The purpose of the TurnOut state is to transition from the InRow state to the DriveStraightHeadland state. For this transition to go smoothly, the TurnOut state should turn the robot fast and precise, so rotate the robot precisely to the desired angle. Furthermore, the robot should not knock over any maize plants and turn within reasonable distance from the entry point of the row. For turning in, the robot should always be able to make a successful turn from the headland into the row.

A successful turn is defined as a turn in which no plants are damaged, in which the InRow state is able to pick up the navigation successfully and in which the turning time is equal to or faster than the turning time with the algorithm from the Bullseye.

2.1.5 Method

2.1.5.1 Iteration 1

To fix the problem of the Bullseye code being incompatible with the Jackal, it was slightly changed. As described above in Paragraph 2.3 and visualised in Figure 12, alterations were made to the turnout state, to make sure the Jackal only turns 90 degrees when leaving the row, thus allowing it to drive over the headland. The range of the LiDAR data that was used for detecting the headland shape and counting the rows had to be altered. Instead of considering the data in front of the robot, data from either the left or right side of the robot was used, depending on the turning direction of the robot.

It was expected that this would work and give similar results as those that were obtained last year (97% success rate in entering the correct row). This appeared to be far from the truth, as due to the nature of the code, the part responsible for letting the robot drive parallel to the end of the rows, at a constant distance, could not control the Jackal. This means the Jackal would drive straight on in the direction that it had at the start of the state, which can be problematic in multiple scenarios. It can for example cause the robot to drive into the plants, or away from the field, as illustrated in Figure 3, which comes from the report of last year.

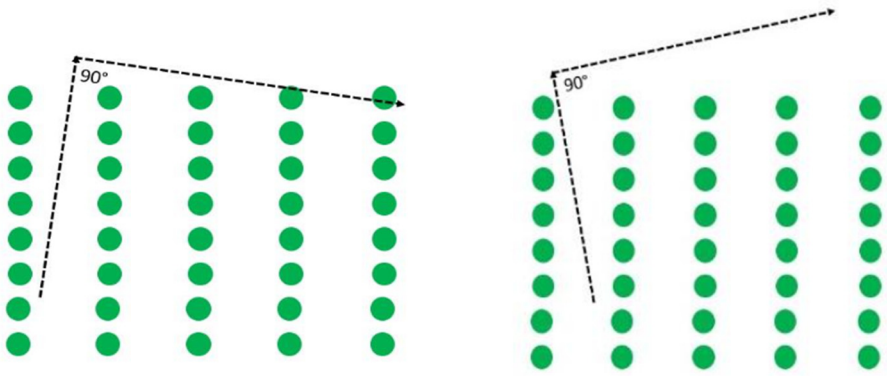
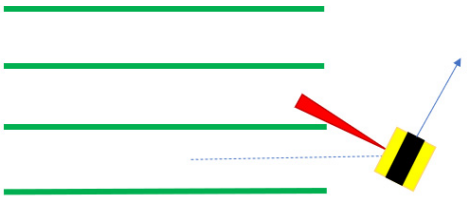
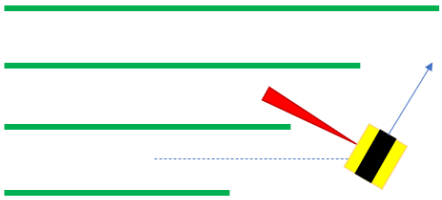


Figure 3:2 Possible problems occurring when no corrections are made after entering the headland slightly skewed. The robot can either drive into the maize or away from the field.

Besides the robot driving in another direction than desired, the situation where the robot is under an angle with respect to the field also influences the counting of the rows. In Table 14 it is illustrated how the row detection can go wrong.

Table 4: Possible problems in case of a non-parallel orientation of the robot relative to the headland. The dotted blue line represents the driven path, the blue arrow the driving direction. The red beam visualises the LiDAR range used for counting rows.

	Robot skewed	Headland skewed
Counting previous row		
Missing a row		

Upon testing, the effect on the row counting capability of the robot was the most prominent. To solve this, the beam of LiDAR data used for detecting and counting the rows was adjusted. Using the IMU, it is possible to know the angle of rotation of the robot with respect to its last driving direction (in the row). As illustrated in Figure 4, this angle alpha is used to adjust the range of used data points. This same angle is also used to correct the default distance of 0.3 meter that the robot must drive between the detection of two rows, as this distance increases if the robot is under an angle. The last parameter that was influenced and therefore adjusted was the distance the robot must drive after detecting the maize plants of the row it has to enter.

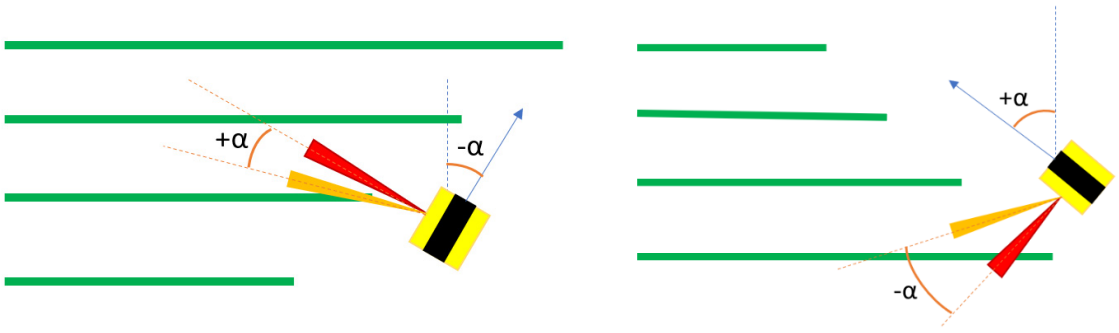


Figure 5: 3Adjusting the LiDAR range used for row counting based on the IMU orientation data.

These adjustments gave an improvement of the performance on the headland, but unfortunately the final result did not satisfy the requirements. The robot would still sometimes count a row

twice, sometimes don't count a row at all, and it also had difficulty stopping right before the middle of a row. This combined with the problem where the robot could drive into or away from the field, made clear that another approach was required.

2.1.5.2 Iteration 2

Using the knowledge from previous attempts, the approach to headland navigation was altered. The old approach separately tried to drive parallel to the headland and to count rows, both with a limited range of the complete LiDAR data. Instead, the new aim was to use as much of the available data as possible, in order to get the best understanding of the robot's environment. The intention was to detect the rows of plants and use that information to both count them, as well as to drive parallel to a line through the last plant of each row.

Two different methods were tried for detecting lines (rows) in the LiDAR data. The first method was the Hough transform, which is an algorithm that works with a voting system in a parameter space. As our data however did not really contain lines, only scattered points, this method did not give good results. The LiDAR measures the position of the leaves and stems of the maize plants in a row. This results in a pattern where the data is spread around a line, with a radius/deviation of approximately 0.3 meter. Upon research it became clear that for such distribution of points, RANSAC should be much more suitable. RANSAC stands for *random sample consensus*, which describes the method: a random sample (in our case: two points) is drawn from the data and a line is fitted through these two points. It is than calculated how well this line describes all the data and that score is saved. After repeating this process, a certain number of times, the best scoring line is the result. In our case this one line describes only one of the many visible maize rows, meaning the process must be repeated to find the others. This is done by removing the data that is within a certain distance of the detected line and then using RANSAC again to find a new line in the remaining data.

Once multiple lines are detected, they can be used as a representation of the field and headland. To assist the driving of the robot, a line is fitted through the end of each of the detected rows. This fitted line approximates the headland shape, which can be used to drive parallel to it, at a specified distance. Figure 15 shows an example of this row and headland estimation. Once the robot can drive along the rows without crashing into the field or driving away from the field,

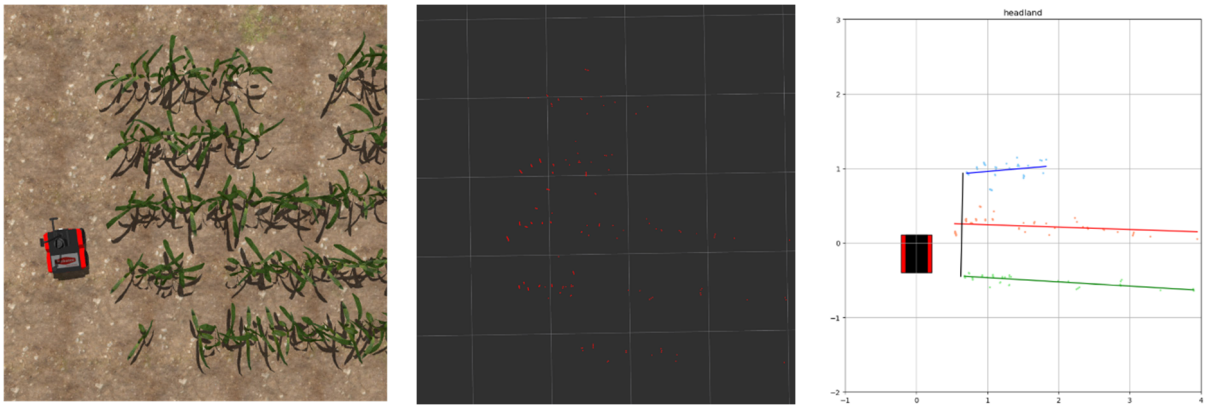


Figure 6:4 RANSAC row estimation. From left to right: simulation, RVIZ and own visualisation of the field. The blue, red and green dots are LiDAR data, the lines through those are the estimated rows and the black line is the estimated headland shape.

the only problem is counting the rows and determining when to stop. As RANSAC gives the coordinates of the last point of each rows, it is very easy to know whether the robot is standing right before a row of plants, or right between two rows of plants. Driving along the field will give an alternating pattern of being in front of a row of plants and being in front of the open part of the row where the robot can drive. These observations are used to count the rows, but to make sure no row is counted twice, use is made of the fact that the maize rows are 0.75 meter apart. The inbuild odometry of the Jackal is used to make sure the robot drives at least 0.6 meter between two row detections. This value of 0.6 is determined by trial-and-error and has to be lower than 0.75, due to most likely inaccuracy of the odometry and the processing time of the algorithm.

Due to the large spread in the data, the row pattern is not very distinct in the raw data. This fact, combined with the random nature of RANSAC, meant that sometimes wrong row estimations were given, as visible in Figure 7. Sometimes the incorrect row estimation has no influence on the performance, but in other cases it is so wrong that it can cause the robot to drive into the maize. The probability of this happening can be reduced by running more iterations of RANSAC, but that is time consuming, meaning it would reduce the rate at which the robot could steer and therefore hurt the performance. Therefore, it was decided to try to make use of information that the rows are 0.75 meter apart and virtually parallel. So, a manual filter was created that checked the RANSAC results for being in accordance with this knowledge about the field. However, it appeared to be very challenging to discard only the wrong row estimations and keep only the good ones. So, after quite extensive testing, this solution was saved for later and a simple backup plan was implemented. This backup mechanism meant that automatically a row was assumed to be passed if the robot had driven 0.8 meters without detecting a row. During testing of this system, it worked quite well and often allowed the robot to correctly detect the next row.

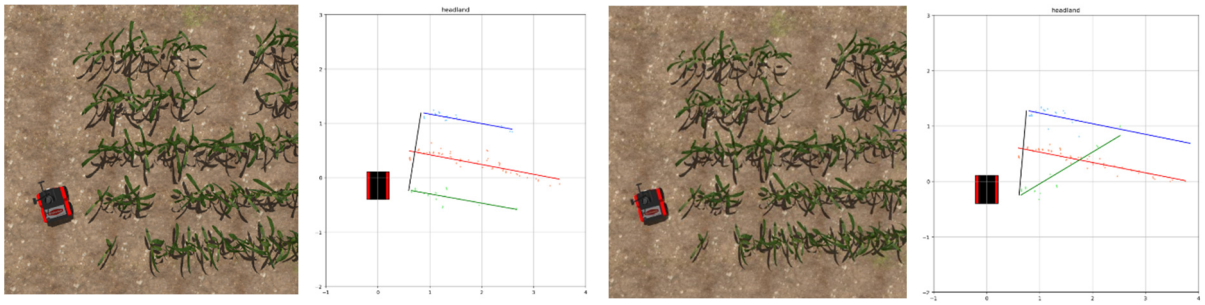


Figure 7:5 Example of correct and wrong row estimations by the RANSAC algorithm, using barely different data. Note that even in the right image the start of each row is estimated correctly, as well as the headland shape, meaning that this would not influence the robot's behaviour.

2.1.6 Result

We calculated the efficiency of the software separately for in the row and in the headland.

2.1.6.1 Curved rows

For this test, different worlds were generated per maximum angle variation, this parameter indicates the maximum curvature of the row, those rows were 10 meters long. 5 runs were done

for each angle variation, each in a different world. The unit of the parameter *max_angle_variation* is unknown to us, as we couldn't find it in the organisations documentation, but it determines how much a row is allowed to curve over a certain distance. Figure 8 shows a world per *max_angle_variation*. So, this is about what the maximum angle is and not a fixed angle, this makes testing more difficult and less representative, because a *max_angle_variation* of for example 0.3 can theoretically still yield straight rows.

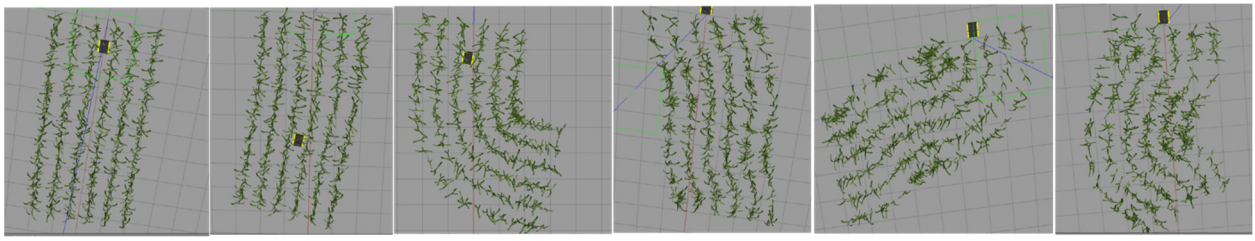


Figure 8: Per max_angle_variation tested an example of how the world could look like. From left to right a max_angle_variation of 0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30 was used.

Table 2: Test results of the different runs, under several angle variation in the generated world

run		1	2	3	4	5
Angle variation [-]	**0.00**	np	Np	np	np	np
	0.05	np	Np	np	np	np
	0.10	np	Np	np	np	np
	0.15	np	Np	np	np	np
	0.20	4	2	np	np	np
	0.25	3	2	1	np	3
	0.30	np	3	2	2	3
	0.35	2	2	3	3	2

The meaning of the codes in the table above is as follows:

Table 3: Explanation of the codes used in table 2.

code	explanation
np	no problems for this run, the robot did not knock over any plants.
1	A lot of noise when leaving the row, causing the robot to adjust too much and then entering the row goes wrong.
2	Many leaves in a bend, so that the bend is not recognized, and the robot continues to drive straight ahead[2]
3	A too sharp bend in the row where the robot cannot steer fast enough and knock over plants in the outside bend
4	Too many leaves (noise) causing the robot to make incorrect adjustments and knock over plants.

The results obtained during testing raised the question what the cause could be of the robot failing in the sharper turns. To investigate this, a visualisation tool was developed that showed all perceived data points, highlighted those that were used for the actual calculation and showed the calculated row estimations. Figure 9 shows this tool, alongside with RVIZ, the default data visualisation tool in ROS, and the view in the simulation. Using this tool, the problem is quite apparent: due to the severe curvature of the robot, plants from the outside bend are in the data range used for the inside bend. This cause the estimated line to be completely wrong and therefore the robot does not steer correctly.

This problem mainly occurred in worlds created with the old world generator, which had uneven plant densities in the bends. The problem for the most disappeared with the introduction of the new world generator and better tuning of the P-controller. Note that the robot is not exactly in the middle of the row and correctly aligned in the bottom left image in Figure 9. This means the controller already must do a poor job to reach such a situation.

[2] With the old world generator used for these tests, the plants are not at a distance of 13 to 19 cm. In the inner curve the plants are closer together and in the outer curve the plants are further apart, making sharp curves very dense with leaves. See also Figure 8.

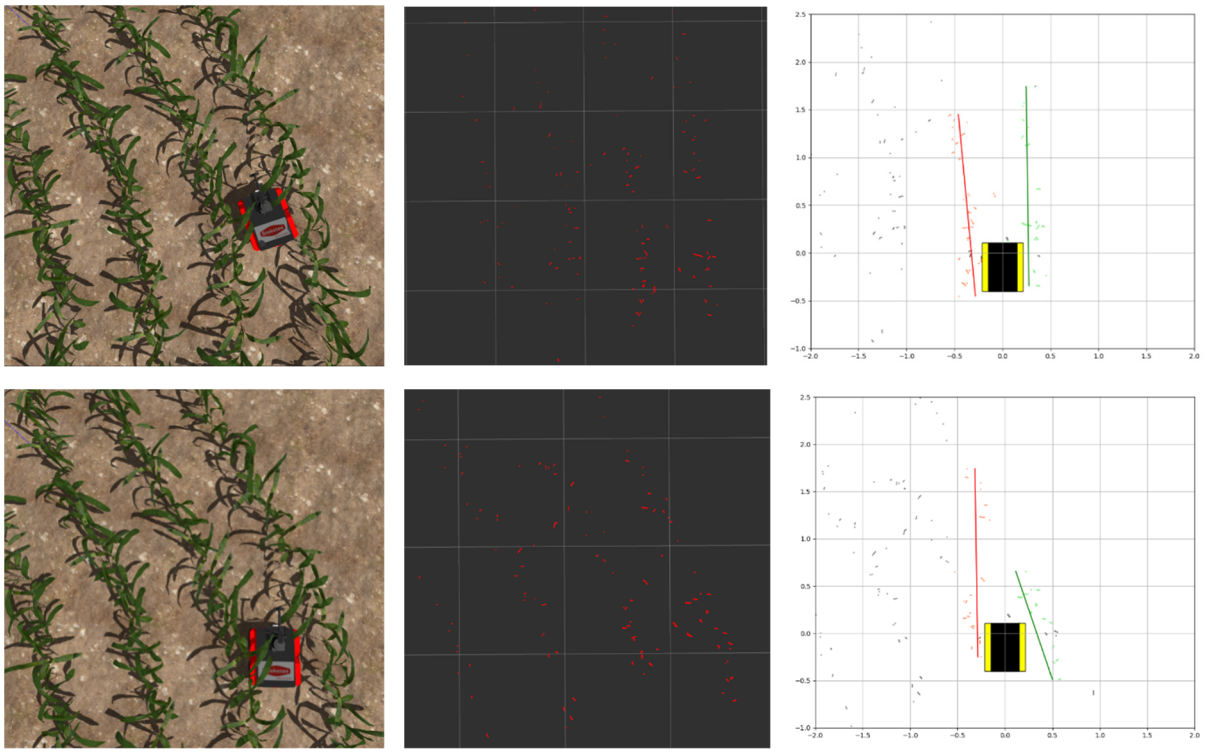


Figure 9: Comparison of row estimation results. Top shows a correctly estimated left and right row, bottom shows a wrong left row estimation. From left to right: simulation, RVIZ and own data visualiser.

2.1.6.2 Rows with gaps
Single sided

For the first test, plants were removed one by one, from one row (either right or left), to see what the influence on the performance was. So, after removing one plant, the robot had to pass that gap in the row, to see whether it would give problems. This was always tested twice and in both driving directions. The testing was done in worlds with straight rows of 10 metres, to make sure there was no effect of turning in or out on the performance around the gap.

Table 4: Test results in world with straight rows, with an increasing number of missing plants.

	run	1		2		3		4		5	
		left	right	left	right	left	right	left	right	left	right
Missing plants	1	np	np	np	np	np	np	np	np	np	np
	2	np	np	np	np	np	np	np	np	np	np
	3	np	np	np	np	np	np	np	np	np	np
	4	np	np	np	np	np	np	np	np	np	np
	5	np	np	np	np	np	np	np	np	np	np
	6	np	np	np	np	np	np	np	np	np	np
	7	np	np	np	np	np	np	np	np	np	np
	8	np	np	np	np	np	np	np	np	np	np
	9	np	np	np	np	np	np	np	np	np	np
	10	np	np	np	np	np	np	np	np	np	np
	15	np	np	np	np	np	np	np	np	np	np
	20	np	np	np	np	np	np	np	np	np	np
	all	np	np	np	np	np	np	np	np	np	np

Note: all plants missing on one side is exactly the same as driving along the last row of a field.

As visible in Table 4, removing plants in one row on one side of the robot gives no problems. This was expected, as the robot is also assumed to be capable of driving along the outer row of a field, which is the same as having all plants missing in one row. There could have been a difference, as in that case the robot could theoretically keep a larger distance from that one row, as it can't collide with any plants in the adjacent row. In this case however, it did have to stay right in the middle of the row, as after one or more missing plants, the row 'starts' again and it has to be between both rows, without hitting plants.

Table 1: Explanation for the code used in table 4.

code	explanation
np	no problems for this run, the robot did not knock over any plants

Double sided

For the second test, plants were removed on the left and right side simultaneously, causing a situation where the robot can't use the row opposite of the gap to define its position and orientation. This means the robot has to use the plants behind or further in front of the robot (as explained above the adjacent rows are not used).

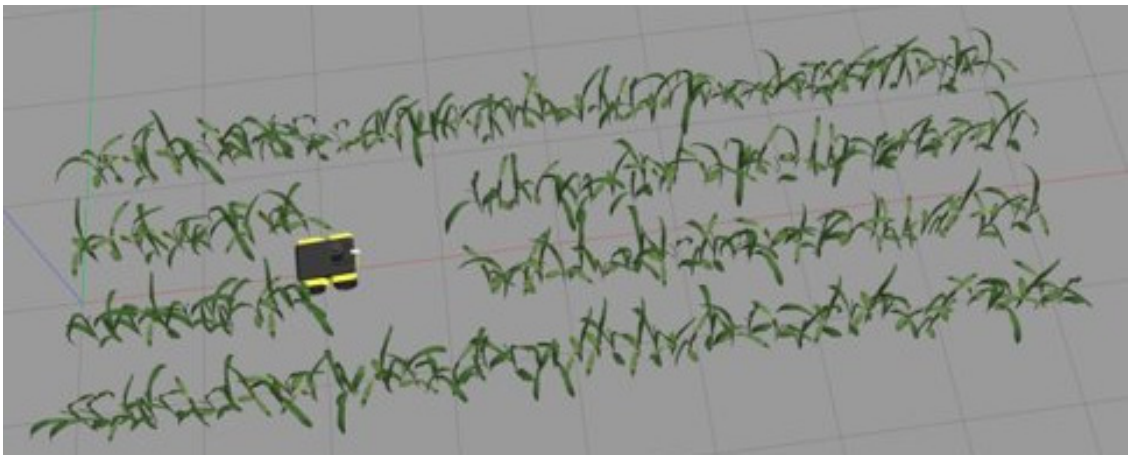


Figure 10: Example of task 2 world from the old world generator. Note the manually created two sided gap of approximately 12 missing plants.

Table 6: Test results in a world with a gap at both sides of the robot.

run		1	2	3	4	5
Missing plants	1	np	np	np	np	np
	2	np	np	np	np	np
	3	np	np	np	np	np
	4	np	np	np	np	np
	5	np	np	np	np	np
	6	np	np	np	np	np
	7	np	np	np	np	np
	8	np	np	np	np	np
	9	np	np	np	np	np
	10	np	np	np	np	np
	11	1	np	1	1	np
	12	np	np	np	np	np

Table 7: Explanation of the codes used in table 6

code	explanation
np	no problems for this run, the robot did not knock over any plants
1	robot touched a plant when driving in one of the two directions, other direction works

2.1.6.3 New world generator

Table 2: Results from navigation in the row in different, specific generated for each task, worlds

world	run	velocity (m/s)	# Rows (10 m long)	plants damaged
task 2	1	1	15	0
task 2	2	1	11	0
task 2	3	1	13	0
task 2	4	1	13	0
task 2	5	1.5	13	2
task 2	6	1.5	13	0
task 2	7	2	13	9
task 1	8	1	13	0
task 1	9	1	13	0
task 3	10	1	13	0
task 3	11	1	13	0

The in-row navigation works so well that virtually no plants are hit while driving in the row. On each run, the robot drove through approximately 13 rows of 10 meters. The planting distance was 16 cm on average, so with plants on both sides, the robot drives past about 1600 plants during a run (in practice slightly less because of gaps in the rows). During testing, plants were only hit when the velocity was increased. There are two important things to mention here. First of all, the gains are not adjusted for the higher speeds, so the robot can make relatively less adjustments. Secondly, it is important to mention that the plants that were driven over at a higher velocity were all at the beginning of the row. Our assumption is therefore that the robot at a lower velocity has more time to correct if the robot is not correctly positioned at the beginning of a new row. After all, sometimes serious adjustments must be made when the robot steers into a row, but there is less time to do so when the robot drives faster.

Task 1 Headland

Table 9: Results of testing the algorithm in task 1 worlds.

Direction of turning	# turns into correct row	# turns into wrong row	# plants damaged
Left	14	0	0
Right	13	1	3
Combined	27	1	3

Table 13 shows that 27 out of 28 turns were performed correctly in the worlds of task 1, this means a success rate of 96%.

Task 2 Headland

Normal speed

Table 10: Results of testing the algorithm in task 2 worlds.

Direction of turning	# turns into correct row	# turns into wrong row	# plants damaged
Left	27	1	1
Right	27	1	> 10
Combined	54	2	> 10

As displayed in Table 14, in worlds of task 2, at normal speed (1 m/s with a small reduction during steering) 54 out of 56 turns were performed correctly, this is a success rate of 96%.

Increased speed

Table 11: Results of testing the algorithm at 1.5 m/s in task 2 worlds.

Direction of turning	# turns into correct row	# turns into wrong row	# plants damaged
Left	28	0	5

Table 15 shows that while constantly driving 1.5 m/s (in row, on the headland the usual 0.7 m/s) in the world of task 2, all of the 28 performed turns were correct, so that is a success rate of 100%.

When driving 2.0 m/s second in the row, the robot leaves the row while driving fast and almost drives into the ditch. It seemed the detection of the headland is working less well and the robot does also not always leave the row lined up nicely. Also, when entering a new row, the robot immediately speeds up to 2 m/s, which does not allow for quick corrections. Therefore, in a very large portion of the cases a large amount of plants was hit, which made it nearly impossible to score all the damage.

Task 3

Table 12: Results of testing the algorithm in task 3 worlds.

Direction of turning	# turns into correct row	# turns into wrong row	# plants damaged
Left	12	0	0
Right	13	1	0
Combined	25	1	0

For task 3 worlds the test results are in Table 16. At normal speed, 26 turns were performed, of which 1 went wrong, this means a success rate of 96%.

Number of rows to pass

As mentioned above, theoretically there is the possibility of a cumulative error when passing multiple rows. To check the possible influence of this factor, data is recorded separately for each specific number of rows that the robot passed. As there were no obvious differences between these results for the different tasks, they have been combined for overall statistics. Figure 20 shows the respective success rates for entering the correct row, when a given number of rows had to be passed. There is a slight variation in the results, but definitely not a trend towards worse results if the robot has to pass more rows.

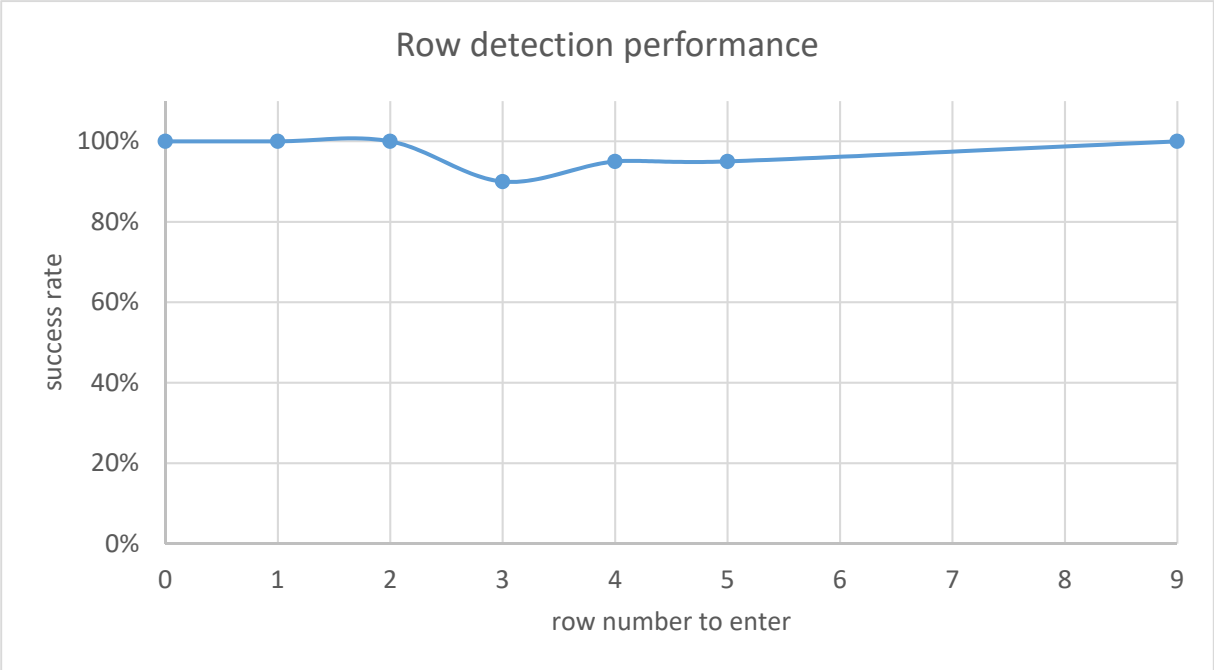


Figure 20: 6 Graph showing the success rate at different numbers of rows to pass for the robot.

Direction of turning

Table 13: Results of testing the influence of the turning direction on the algorithms performance.

Direction of turning	# turns into correct row	# turns into wrong row	% correct turns
Left	81	1	98.7
Right	53	3	94.6

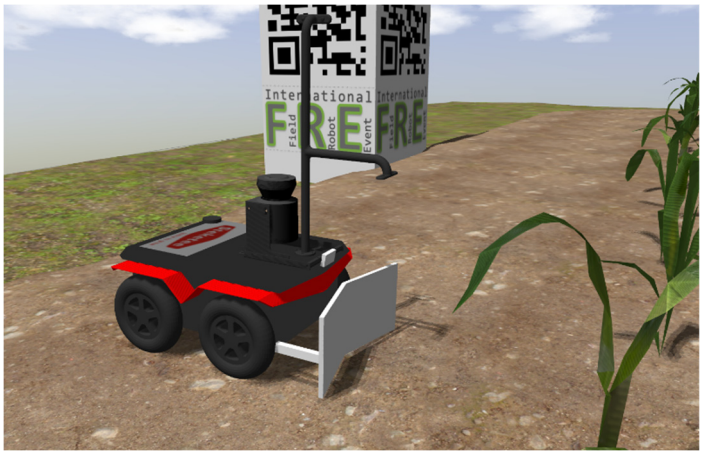
The results in Table 17 indicate a small difference between the number of times the robot enters the correct row after turning left or right. However, the difference is not alarming and therefore no extensive further research was done. From the algorithm, there seems to be no reason for a difference.

Overall

In total 138 turns were performed during testing, of those turns 134 were into the correct row, meaning that the overall success rate is 97%.

3. Applications in a virtual field

On top of the standard Jackal two cameras were added and a shovel for task 3 and 4.



3.1 Task 3 – Weed / object detection and mapping

Figure 11: The ale bottle.

Figure 12: The beer can.

Figure 13: 8The coke can.

Figure 14: The retro pepsi can

The litter detection seemed like the easier, more straight forward part of the detection because the colours of the litter are quite a bit different from the maize plants and the ground, but after making some segmentations based on colour, the conclusion came that recognizing all the litter

together with one segmentation is not possible. This is because the organisation of the event used four different types of litter and all types have different colours, they are shown below in Figures 21 up to 24. As the different types of litter have different types of colours, the segmentation of the litter was done individually per type of litter.

3.1.1.1 Ale bottle

For the ale bottle, the segmentation was made based on the HSV-image. This gave a segmentation of the glass part of the bottle, however this did not include the label on the bottle. To segment the label as well, two more masks were used, based on the BGR-image and the HSV-image. However, the mask for the label gave a lot of noise. To remove the noise, two closing and an opening filter were used. Without the noise, the label masks and the glass mask were combined. This gave again a little noise in the mask, so to get the final mask for the ale bottle, an opening and a closing filter were used.

3.1.1.2 Beer can

For the detection of the beer can multiple segmentations were used as well. The recognition was split up in recognizing the blue part of the can and the white part with the HSV-image and the red part with the BGR-image. As can be seen in the picture of the beer can, there also is a light brown part. For the detection, this part is too similar to the ground. The result of segmenting this part, based on colour, gave too much noise from the ground. To solve this, the light brown part of the beer can is left out of the segmentation because it results in too much noise of the ground. Leaving out this part did not give any other problems in recognizing the beer can.

The mask of the beer can recognize the robot, as the red colour in the beer can and the red colour of the robot are almost identical. A part of the robot is then incorrectly recognized as a beer can. This is solved during the processing of the final mask for all litter types.

3.1.1.3 Coke and retro Pepsi can

When making the segmentation for the coke can and the retro Pepsi can, the conclusion was made that the white text from the coke can is almost the same white as the retro Pepsi can. Therefore, the recognition of those two cans can be combined. The coke and Pepsi cans are recognized with two masks based on the HSV-image. One mask segments the red parts of the cans, and the other mask segments the white parts of the cans. These two masks combined give a mask that segments the coke cans and the retro Pepsi cans, but it still had a little noise, so a closing and an opening filter were used.

3.1.1.4 Total mask

The last step is the combination of the previous described masks in one mask that segments all the litter types that needed to be recognized. This mask is used to find the contours and the centroids of the objects in the image. The mask can be found in Figure 25.

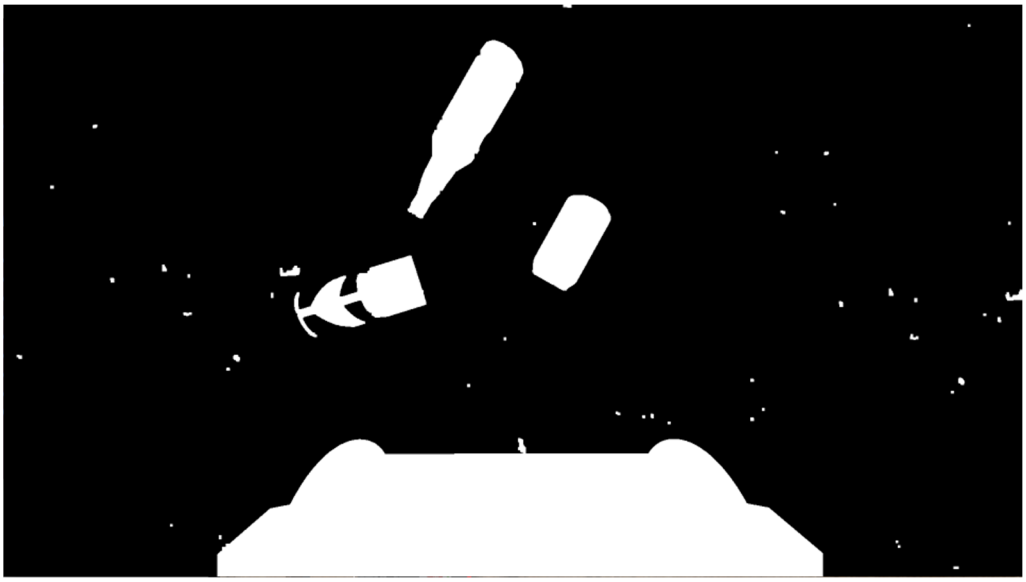


Figure 15: This mask is used to find the contours of the objects

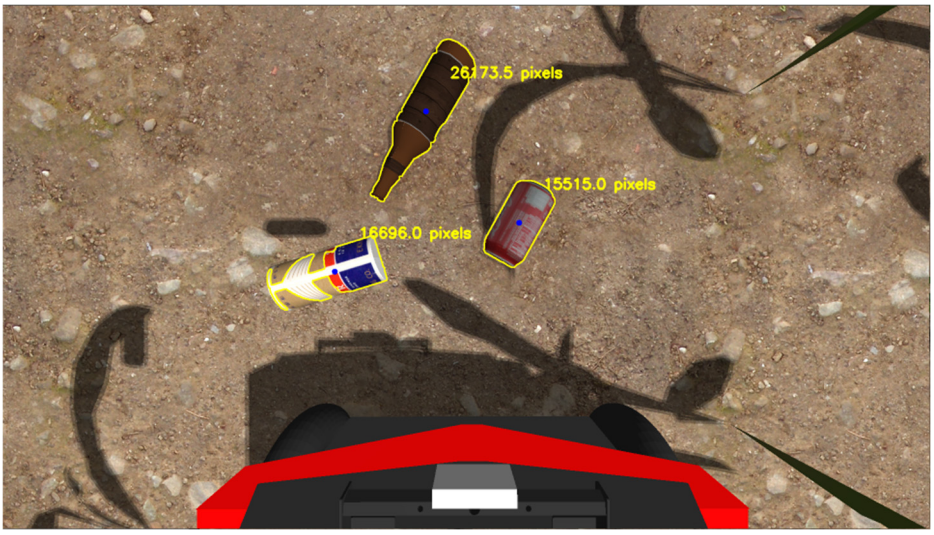


Figure 16: The final result, with the contours of the litter indicated in yellow. The blue dots are the centroids of each object.

The contours in the mask are found with the function *findcontours*. *Findcontours* is a function in the OpenCV library, the function finds all contours in a binary image. Then for all contours, the area is calculated, this can be done by another OpenCV function: *contourArea*. All contours which have an area less than 15000 pixels are filtered out. For the remaining contours, the

centroid of the contour is calculated, this is done with the 'moments' function from the OpenCV library. Based on the centroid of the contour, some more filtering needs to be done. Earlier it was stated that the robot also was recognized in the final mask. This can be easily filtered out with the centroid of the contour from the robot. The camera is fixed on one position on the robot. This gives that the robot is always in the same position in the image. So, all centroids which are in the position where the robot is in the image should be neglected. This is also done if there are centroids that are on the sides of the image, that way, when objects are in the row next to where the robot is, these objects will also be neglected. The final contours used for detecting the litter can be found in Figure 26.

When everything is filtered correctly, the only contours and centroids which are left are the contours and centroids of the objects which are correctly recognized. All the pixel coordinates of the centroids of these objects are being published to a litter topic. When there is no correctly recognized object in the image, the value, -1, will be published for the x and the y coordinate. This way the localization code can subscribe to the topic and can check if there are objects that need to be localized.

3.1.2 Weed detection

The main problem for the weed detection is that the weeds and the maize plants are both green. This makes it hard to get a good segmentation for the weeds. The objective was to detect two different types of weed. They are show below.

Figure 17: The "Unknown weed".

Figure 18: The "Nettle"

The first step to detect the weed was applying an excessive green filter, this reduces the influence of the lighting. Based on the excessive green image, a mask was made that only shows the green parts in the image, so only the maize plants and the weeds. This mask gave a little noise, so opening and closing filters were used.

Then based on the HSV-image another segmentation was done to remove most parts of the maize plants. This was possible because the colour of the most parts of the maize plants was a slighter darker shade of green than the green from the weeds. This produced a mask of the weeds with sometimes a part of a maize plant.

However, in this mask some leaves of the weeds were not connected to the rest of the weed. This is a problem because this would yield multiple contours for only one weed. This problem is solved with a closing filter. This way all leaves of the weeds are connected with each other and form one contour. The result of the segmentation is to be seen in Figure 29.

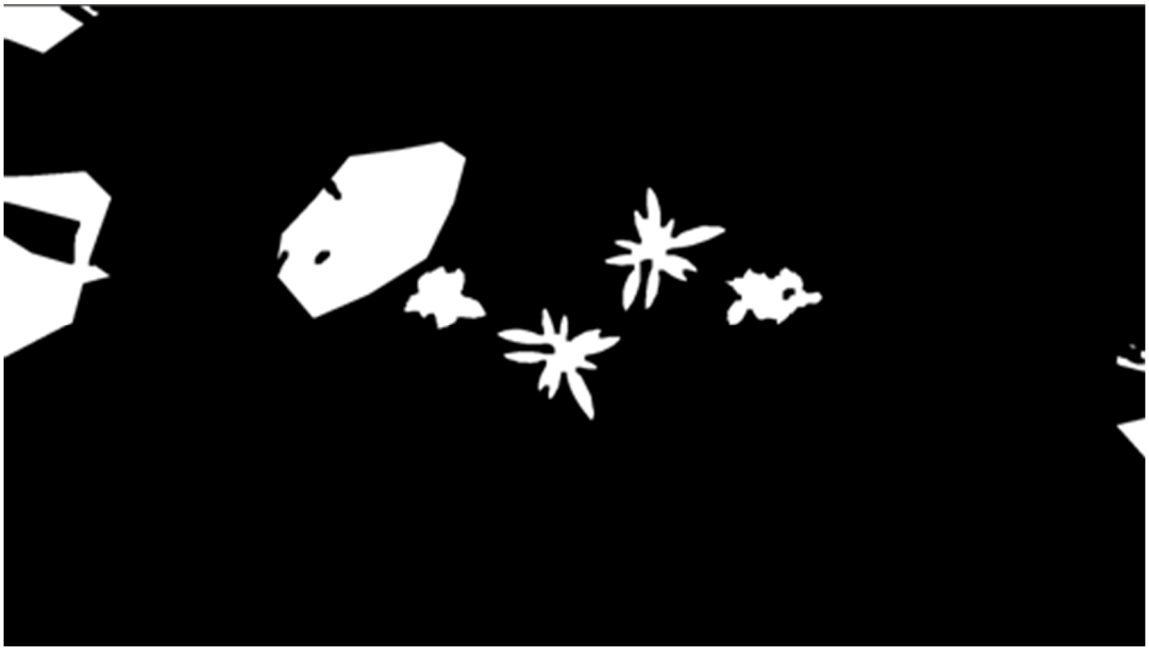


Figure 19: With this final mask the contours and the centroids of the objects in the mask are found.

Then a minimum bounding rectangle is fitted around the contour. This can be done with the OpenCV function *minAreaRect*. The sides of this rectangle are checked and when the sides are too long or too short to be from the weed, then the contour that the rectangle is based on will be neglected.

Also, the ratio between the area of the rectangle and the area of the contour will be checked. As you can see in the pictures of the weeds, both weeds have leaves that are pointing outwards. This means that the area of the bounding rectangle should be quite a bit larger than the contour itself. So, when the area of the bounding rectangle is almost the same as the area of the contour, then the contour is not from a weed. Also, the other end is taken into consideration, when the area of the bounding rectangle is very large in comparison to the area of the contour, then this contour will also not be used.

Furthermore, a convex hull was fitted around the contour as you can see in Figure 30. OpenCV has a function for this called: *convexHull*. The convex hull is used in the same principle as the bounding rectangle. The ratio between the area of the convex hull and the area of the contour is checked and when the ratio is too close to one or too close to zero, then the contour is not from a weed.

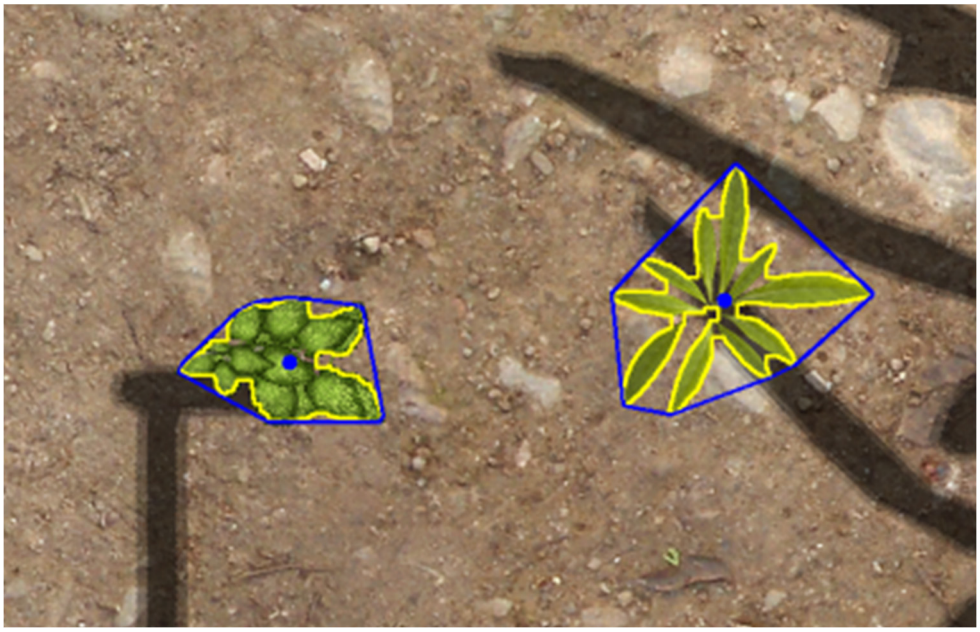


Figure 20: Weeds with the convex hull and contour drawn.

The final step to filter out the false detections for the weeds is the same as for the litter. If the centroid of a contour is on the side of the image, this contour will then not be used, and for the weeds, also when a centroid is on the top or on the bottom of the image, these centroids will also not be used. This way only weeds that are completely in the image will be recognized. The final result can be found in Figure 31.

When everything is filtered out correctly, the centroids of the correctly detected weeds will be published to the weeds topic, in the same way as is done for the litter detection.

Figure 21: The final result of the detection. The red rectangle is the minimum bounding angle, the yellow outline is the contour of the weeds, and the blue dot is the centroid of weed, and thus the location to be registered.

3.1.3 Results

On average 70% of the litter in the field was detected. The rest was not detected and there were no false detections. For the weed was 80% detected but there was also 10% of false detections.

3.2 Task 4 - Weed / object removal

3.2.1 Plan A: Robot arm

The first idea was to mount a robot arm on the Jackal. This had several advantages over other solutions. First, the cans and weed could be located throughout the field, so also between the maize plants. Because a robot arm is flexible it can pick up the cans and weed even if it is located between the plants or very close to them without damaging the plants. Also, the robot arm was thought to be able to pick up de cans as well as the weed. Therefore, only one solution should be sufficient. Because extra points were given for actually picking up the weed or cans, the arm was preferred.

It was chosen to use the Niryo Ned robot arm, for specifications see Appendix 6.3. It is a 6-axis robot arm used for education and supported by ROS. An image of the arm and its degrees of freedom can be seen in Figure 22. Because it has 6 degrees of freedom, the robot arm can move to every location in every orientation in its reachable area. The arm was also equipped with a gripper and a camera on the wrist. With this camera object could be detected, recognised, and navigated to. This seemed a very useful feature for task 4. The robot arm was at first placed on top of the Jackal, just behind the LiDAR.

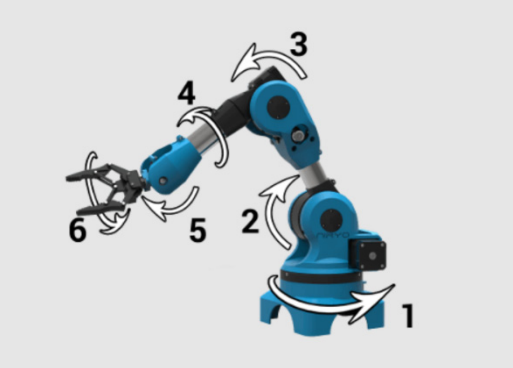


Figure 22: 9Niryo Ned robot arm with the 6 degrees of freedom visualised

Forward and inverse kinematics

The positioning of the arm can be done in two ways, named forward and backward kinematics. The robot arm consist of joints (the movable parts of the arm) and links (the rigid bodies that connects the joints). There are two types of joints: revolute and prismatic, see Figure 23. A revolute joint is for rotation and a prismatic joint for translation, the arm used for the task was revolute. For moving the robot arm, the joints have different positions for every pose of the arm. The end effector of the arm is the gripper. With forward kinematics, the orientation and position of the end effector is determined, and this is done with the displacement of the joints. Another way to set the robot arm to a certain position is with inverse kinematics. Hereby the end effector has a specified position, and the joints are set in such a way that this position is achieved (Kucuk & Bingul, 2006). For task 4, inverse kinematics has been used. In the next paragraph is explained how this is done.

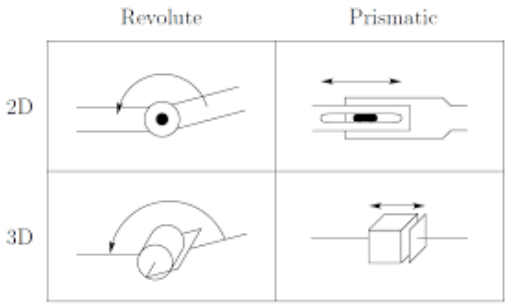


Figure 23: 10Two types of joints

Planning movements

In the program used to control the arm is MoveIt. This is a package on top of ROS. It is used for motion planning, manipulation, 3D perception, kinematics, control, and navigation. This makes it possible to implement inverse kinematics. Inverse kinematics is the planning of the movement of the arm by giving only the end coordinates instead of the rotations of each link (MoveIt, 2021). This is very useful in the case of task 4 since the end coordinates differ each time. In the most optimal scenario, the camera can detect the object and give the coordinates of the object after which the arm can move to it. By implementing this, the arm will only need 3 states: rest, move to object and release. A schematic drawing can be seen in Figure 24*: 12*. The gripper will need only 2 states: open and close. For picking up an object the sequence will be: rest – move to object – close – rest. After driving to the correct headland, just before the turn: rest – release – open – rest. Apart from 'move to object' every state is similar and does not require much programming since the poses can be made in MoveIt.

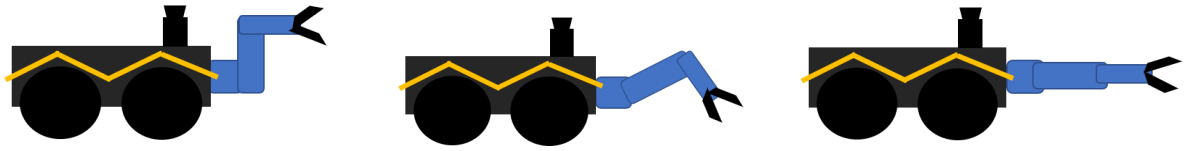


Figure 24: 12The 3 positions of the arm on the Jackal. LTR: rest, move to object (just an example depends on the objects location), release

3.2.2 Plan B: Blade

Since there was not enough time to finish the arm before the code of task four should be hand in, the decision has been made to implement a blade in front of the Jackal, see Figure 25 and Figure 26. A blade could easier be attached on the Jackal since it has no moving parts (just one fixed link). The joints, which the robot arm had, did not have to be taken into account. Furthermore, the blade did not have any topics. The problem which the arm had, was that the topics were missing, and a lot of effort had to be done to get these, could be avoided. Altogether the result was that the blade could be attached to the Jackal without having a lot of errors due to missing topics etc.

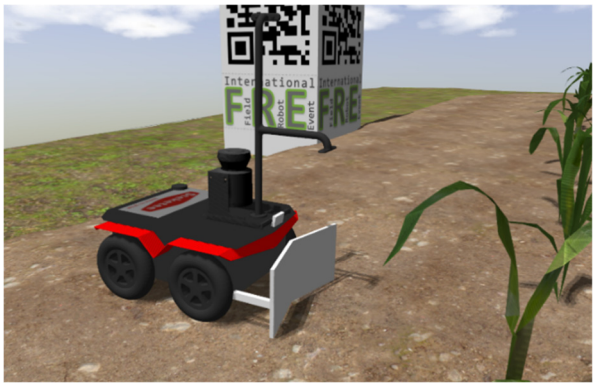


Figure 25: The Jackal with the blade on the front

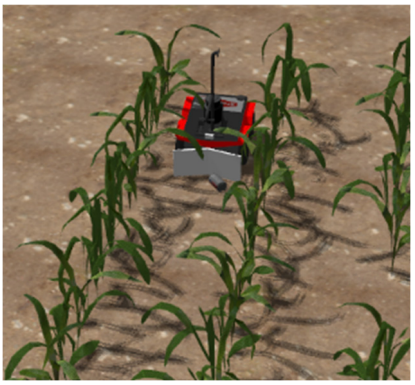


Figure 26: The Jackal with the blade and a can

3.2.3 Results

5 pieces of weed and 5 cans were placed randomly in the competition environment. The rows the robot drove were counted, the moved cans and weed, the cans and weed delivered on the headland and the number of plant damaged. There was not made a distinction between the headlands, because the route was not specified. The simulation world was made with a random world generator. This means that for a new world the location of the maize plants, cans and weed changed. Also the height profile was different. There was tested with four different maximum height differences: 0.0, 0.05, 0.10 and 0.15 m. These values indicate the difference in height between the highest and the lowest point in the field. Table 27 shows the result for height difference 0.0 m. For every world, the task was performed twice (a and b) to see if the result is consistent.

*Table 27: Results for height difference 0.00 m. * = simulation stopped because the robot got stuck*

Run	Rows driven	Cans moved	Cans delivered on headland	Weed moved	Weed delivered on headland	Plants damaged
1 a	2	0	0	0	0	20+ *
1 b	10	3	2	0	0	10
2 a	8	3	2	3	1	10
2 b	10	2	2	2	2	11
3 a	10	1	1	2	2	4
3 b	2	0	0	0	0	20+ *
4 a	10	3	3	1	1	5
4 b	3	1	1	1	1	16

*Table 28: Results for height difference 0.05 m. * = simulation stopped because the robot got stuck*

Run	Rows driven	Cans moved	Cans delivered on headland	Weed moved	Weed delivered on headland	Plants damaged
1 a	1	1	0	1	0	10 *
1 b	3	1	1	2	0	9
2 a	10	3	3	4	3	8 *
2 b	10	2	1	1	1	15

*Table 29: Results for height difference 0.10 m. * = simulation stopped because the robot got stuck*

Run	Rows driven	Cans moved	Cans delivered on headland	Weed moved	Weed delivered on headland	Plants damaged
1 a	10	2	2	1	0	20 +
1 b	2	1	1	1	0	15
2 a	10	2	2	2	0	8
2 b	10	3	3	1	1	1

*Table 30: Results for height difference 0.15 m. * = simulation stopped because the robot got stuck*

Run	Rows driven	Cans moved	Cans delivered on headland	Weed moved	Weed delivered on headland	Plants damaged
1 a	10	3	3	2	0	3
1 b	8	3	3	0	0	20 +
2 a	5	1	0	1	0	0
2 b	1	0	0	1	0	4 *
3 a	7	2	1	2	1	20 + *
3 b	6	0	0	2	1	12

In Figure 27 the percentages of the moved and delivered weed and cans are shown. The percentage of delivered objects is for the 0, 0.05 and 0.10 m are the same. For 0.15 m it is less, there half of the objects moved are delivered. Due to more hight difference, the object are easier lost. The amount of damaged plants are not taken into account.

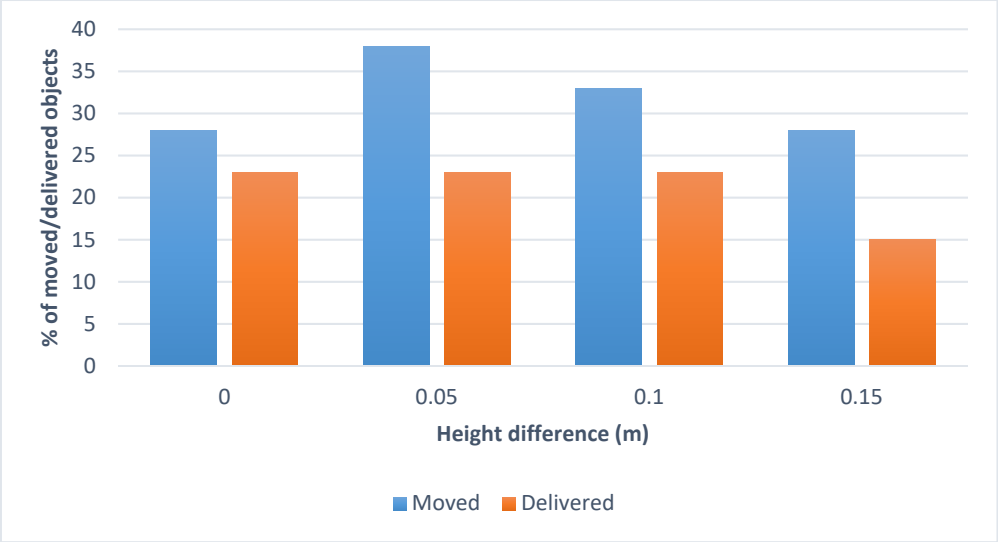


Figure 27: 15Success rate of moved and delivered objects (weed and cans combined)

The results of the two runs in the same world can have a very different outcome, for example run 3a and 3b in Table 27. The first run completed the whole field, collected a can and 2 weeds and did not damage much plants. The second run escalated quickly and did not collect anything. However, in general the two runs in the same world differ less from each other than from the other runs with the same height differences. Yet, it is better to look at both runs before drawing conclusions.

From a height difference of 0.10 m and higher, the Jackal started to bouncing across the field. It even got stuck in the ground sometimes with the blade. From the results can be seen that the number of cans delivered on the headland with a height difference between 0.0 m and 0.10 m is comparable. But with a difference of 0.15 m the results differ a lot between the different worlds. In half of the tests the Jackal did not manage to deliver a single can and got stuck with due to the blade quite often. The delivered weed on the headland is in general lower than the number of delivered cans. From a height difference of 0.10 m, the number of delivered weed is almost zero. One explanation is that the blade loses the weed easily due to the uneven ground. Secondly, the weed is sometimes taken into the row again after the Jackal turned on the headland. Since the weed is smaller than the cans, the weeds are less frequently dropped on the headland because of the shape of the current blade and the fact that the headland management is not adapted to this task. The weeds would stay on the headland if the Jackal drives straight back before turning.

The robot sometimes drives straight over a maize row. The blade than collected the maize plants and pushed them forward. When there were too many maize plants in in blade, the robot did not have enough power to continue driving and got stuck. When this happened, the simulation was stopped.

On the event the height difference was 0.20 m. This made the robot deviate its path and destroy the plants almost immediately.

4 Conclusion

At the start of this project, the requirements for the robot software were formulated in line with the tasks on the Field Robot Event. It can be stated that those requirements for the larger part were met, as team Robatic 2021 won the event. However, there are more detailed conclusions possible for the separate parts of the robot software.

The robot can navigate in the row with high speed and precision, damaging little to no maize plants. The algorithm is both able to cope with curved rows, as well as with gaps in the row, both to an extremer extend that the event required. The turn from the row to the headland is executed precisely, in a small amount of time. On the headland, the robot can turn into the correct row with a success rate of 97 %. Overall, the robot navigated very well during the event.

In the first task no plants were knocked over, but not the whole field was finished within the set time. In the second task, the whole field was finished, again without damaging any plants. It thus can be concluded that the navigation through the field performs satisfactorily, although improvement is always possible.

For task 3, a detection and localisation algorithm was developed. The combined algorithm can detect both the weeds and litter correctly, although sometimes false detections occur. The localisation algorithm is able to correctly map the objects, within a radius of 37.5 cm from the ground truth, for at least 60 % of the detected objects. Overall, the combination of algorithms for task 3 functioned acceptably and performed well on the event.

For task 4, a last-minute solution was developed, which did not perform well because of the uneven ground and a too tight gap between the blade and the surface. However, upon testing it was shown that the modified driving algorithm worked as intended. This algorithm lets the robot slow down when coming in a set range towards the objects.

Overall, the robot functioned as required for the Field Robot Event, except for task 4. The combination of a solid navigation algorithm with an effective detection and localization algorithm resulted in a third, first, first and second place for task 1, 2, 3 and 4, respectively. The consistent performance led to the overall win on the Field Robot event.

CARBONITE – TEAM CARBONITE

Jonas Mayer [1], Klara Fauser [1], Jacob Schupp [1], Lukas Locher [1,*]

[1)] *Schülerforschungszentrum Südwürttemberg (SFZ) e.V., Überlingen, Germany*

[*)] *Instructor and Supervisor*

1 Introduction

Since the Schülerforschungszentrum Südwürttemberg (SFZ) e.V. (English: *Student Research Center*) at the Überlingen site consists mainly of students in grades 7-11 and the team changes every 2-3 years, we are particularly happy when we can reuse the developments of our predecessors. This was often the case in the past with the hardware, which, even if not reused completely, provided a good basis to participate at least in Task 1 and 2 without much change. This year, because the competition took place virtually, many things were different. To participate in Task 1 and 2, this year we not only had to rewrite a good software to control our robot but also had to adapt to a new virtual environment (Jackal simulation). In addition, we had been able to rely on our good hardware for the last few years and had a winning advantage due to mechanical developments such as the Y-turn (a symmetrically built robot that allowed us to navigate backwards and forwards in the field, so we did not have to turn in the headland, only move sideways). Such a winning advantage now had to be developed via software.

Nevertheless, we were lucky. We had already used Ubuntu 18.04 together with ROS Melodic at SFZ for several years for various ROS-based robots, which was needed to run the Jackal simulation. The last Carbonite also used both, meaning we had only little problems finding our way in this environment. As a result, we were able to control the simulated robot manually via a terminal or RViz (a ROS visualization program) within a few days without much work. It also allowed us to use our old software despite the new environment. But of course, there were small changes such as the change from a custom ROS message to the cmd_vel ROS message to control the chassis as well as some bigger parts of the software which had to be replaced, just as the previously mentioned Y-turn, which unfortunately was no longer possible in the simulation because the given robot model only had a forward-facing laser scanner and so we could not navigate both sides.

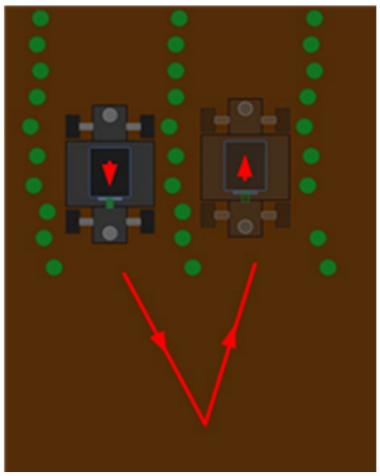


Fig. 1: Demonstration of Y-Turn with old Carbonite

Additionally, the software was still set to control hardware such as a siren. On top of that, the various control algorithms were still configured and optimized for the old robot. Luckily we were able to benefit from our modular software structure of the last Field Robot Events through ROS, Qt, and C++.

Since the last Field Robot event, we have been using a modular software concept managed by ROS-Actions. ROS-Actions are divided into action servers and action clients. In our case, most processes are represented by an action server. A started action server can accept a goal and tries to reach this goal. Meanwhile, it can constantly send feedback e.g., if there are delays in reaching the goal. It also sends a result as soon as it has either reached the goal or had to cancel it, which then for example contains an error code. The action client, which is implemented in various ways, partly within other action servers, can send a goal to an action server and receive the feedback or the result from the action server. Thus, we have a standardized, stable, easily extensible, modular, and powerful interface for most processes on the robot. In addition, an action server can be requested by an action client to terminate its process. This allows us to stop the robot at any time.

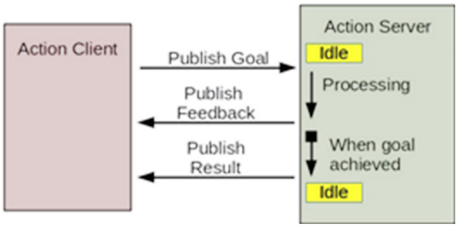


Fig. 2: Sketch of ROS Actions

For several years now, we have also been using a graphical user interface to set parameters for algorithms used during runtime. The individual program parts can access the settings via ROS services and adapt accordingly during runtime. This user interface helped us a lot in optimizing and adapting the algorithms to the new robot and new environment. Since this years' testing and development was only possible in the "home office" and not everyone had a gamepad to control the robot at home as we would usually do, we simulated such a gamepad with the most necessary control inputs and buttons in our GUI, that everyone was able to start the robot at home and control it manually.

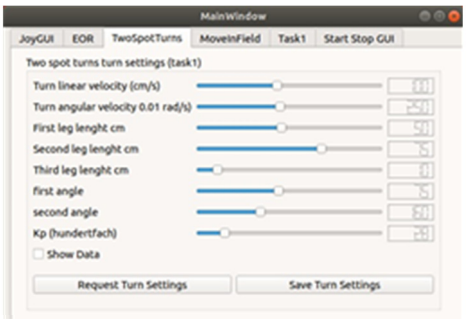


Fig. 3: Qt GUI for Turn settings.

2 Navigation in a virtual field

2.1 Task 1 – Basic Navigation

In task 1, the task was to drive through the field as fast and error-free as possible. Since our software strategy of the last time proved itself and we have even reached the first place for this task, we decided to use as much of the old software as possible. Our concept was more or less the same as last time. It looks like this:

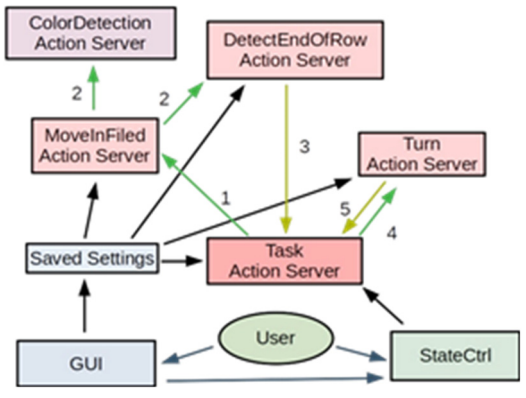


Fig. 4: Sketch of Team Carbonites software structure

Generally, the complete behavior of the robot is managed by a task-specific action server. When the software is started, the action servers for driving are also started, but they wait until they receive a goal. In addition to the action servers, a StateCtrl ROS node is started that preprocesses the user input on a gamepad and provides it to the

task action server. The user can also save settings via the GUI, which in turn are read by the action servers during execution. If the StateCtrl node receives the start signal to start the run, either from a real gamepad connected to the PC or from the virtual gamepad simulated in the GUI, it sends a goal to the task-specific action server. The action server then sends a goal to the MoveInField action server to start the navigation in the field. Since there is another action server for image recognition for Task 3, a goal is also sent there when the MoveInField action server is started. Because action servers only react to a goal, if they were started e.g., by a launch file, there are task-specific launch files, which start the ColorDetection action server with Task 3 but do not start it at all with Task 1. The sent goal is then simply sent "into the void". After a certain distance has been driven in the row (this can also be specified in the GUI), the MoveInField action server sends a goal to the DetectEndOfRow action server, which searches for the end in the row while the MoveInField action server takes over the navigation in the row at the same time. If it has found this end, it sends a done-result and the MoveInField action server terminates. This result is also sent to the task-specific action server and sends a goal to the Turn Action server with an additional parameter, depending on the setting and on which side of the field the robot is standing, to start a turn to the left or the right. The Turn Action server also sends a result after the completion of the turns, which the task-specific Action server receives and thus restarts the cycle (See Fig. 3 for clarification).

Our approach to navigate in the field was based on the same idea as in the last Field Robot Event. To stay between the corn plants, we first convert the laser scanner data from the published format ("sensor_msgs/LaserScan") to a PointCloud format ("pcl_ros/point_cloud") so that we can apply algorithms to the data more easily. This PointCloud is now buffered by a "circular buffer" to filter out momentary extremes like driving over a rock or hill. To find out where in the row the robot is and in which direction it has to steer to stay in the row, first the PointCloud is filtered to a small square (adjustable by settings in the GUI) in front of the robot to save computing power. Then the PointCloud is reduced to the y-dimension, which indicates the displacement of a point to the left and right of the robot. Now a mathematical function is used (see black modeling in Fig. 5 or red markers in Fig. 6) which, starting from the far left, evaluates points in the standardized row distances better than those lying between the rows from the starting point. The sum of all points calculated by this function gives a score, which indicates how well the current displacement of the function and its defined high points match the distribution of the points. If you start to calculate this score for different displacements of the function from left to right, you can calculate from the displacement with the highest score, the direction to the left or to the right at which the function best matches the distribution of the points. A parameter that can be changed in the GUI multiplied by this determined direction gives the actual new direction of the robot.

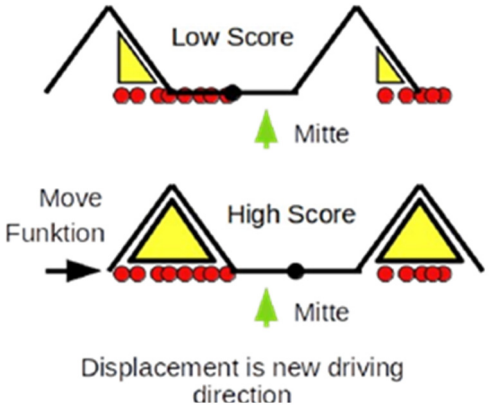


Fig. 5: Illustration of how MoveInField navigation works

To detect the end of the row the space in front of the robot is divided into several stripes. When the number of PointCloud points on the strip exceeds a certain threshold, it is considered occupied. If the number of occupied stripes falls below a threshold, the end of

the row is detected. The advantage of dividing the space into strips is that irregularities such as the pillar in task 3 have little effect on the detection.

This year we have also added that if the number of "occupied" stripes falls below a threshold that is higher than the "DetectEndOfRow Threshold" but still below the normal value, the speed will be lowered. This is especially important so that the end of the row is detected safely and in time at high speeds. It also has the advantage that in a terrain with few plants, the robot also travels more slowly and can thus steer more sensitively.

Since we did not have a hardware advantage this year e.g., the mentioned Y-turn, we tried to increase the speed of our robot in Task 1 and 2 with a new algorithm. We used Hough Transformations to interpolate a straight line left and right from the surrounding PointCloud points (See yellow lines in Fig. 6). When these interpolated straight lines matched in angle and this angle also roughly matched the direction of travel, we gradually increased the speed depending on how well the angles matched. Thus, in straight and well-defined terrain, we could drive faster and in curvy terrain or terrain with few plants, we drove slower and more accurately.

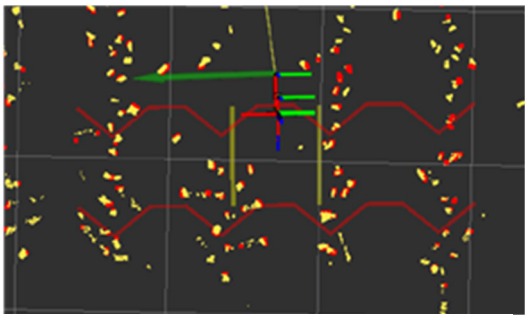


Fig. 6: Markers in RViz while navigation in field

The effort to improve speed and to trust the older base of the code paid off. Even though we knocked down 2 plants entering the row after the turn, we were able to pass the entire field 24 seconds before the time expired and secured 2nd place for Task 1.

2.2 Task 2 – Advanced Navigation

Task 2 is similar to Task 1 in terms that it is about navigating the field but with a much more challenging environment. Furthermore, rows must be skipped during the turn and passed without hitting plants.

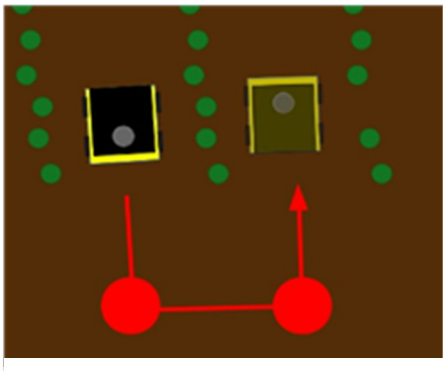


Fig. 7: Demonstration of new Turn with Jackal

This year, as already mentioned, we could no longer perform the old Y-turn in Task 1 and 2, so we developed a new turn that also works with the simulated robot. In the new turn, after the end of the row has been detected, the robot drives out of the row, turns 90° on the spot, covers the average width of a cornrow in distance, turns 90° again, and drives into the new row.

The same turn can also be used for Task 2 since only the distance the robot has to cover after the first 90° turn has to be increased by n (number of rows to be skipped) times the average corn row width.

In the past, we often had problems with the end of the row being recognized late and the robot starting the turn crooked due to the lack of orientation in the headland. In addition, until now we had not corrected the 90° turn and thus had a relatively large deviation from the real orientation to the target orientation. To compensate for these inaccuracies, we had the idea to write the YawOrientationValues determined by the IMU into a "circular buffer" during the navigation in the field, in order to determine the average Yaw on the

last few meters before the turn. This way the exact YawValue can be calculated, which should be reached before and after the first 90° turn. Also, the angle can be controlled during the turn, by steering slightly while driving the tracks, based on the difference between the target and the measured value.

This change was also very worth the effort and worked outstandingly well together with the old software. We did not make a single mistake in the competition. Neither did we skip a row too much or too little, nor did we knock over any plants. Unfortunately, in order to make the detection of the end of the row safer and more stable, we turned down the speed, leading to not having quite enough time to complete the run within the 3 minutes. Nevertheless, we secured 2nd place again.

3 Applications in a virtual field

3.1 Task 3 – Weed / object detection and mapping

In Task 3, in addition to the general navigation in the field, another aspect was added. Objects lying on the ground had to be recognized and classified and theoretically additionally noted in a map relative to 2 pillars, each masked with a QR code. However, since we considered the development of a good method for classifying the objects lying on the ground to be a huge challenge, we decided to concentrate only on the classification for the time being.

We also decided not to develop a special robot for Task 3, because it would only have been more work and we did not necessarily expect better results. Furthermore, since we used the Jackal demo robot, we could simply continue to use the software from Task 1 and 2 and did not have to start optimizing again.

Our first approach to detect the objects was to use existing neural networks and algorithms. We particularly liked the Yolo-v4 algorithm. It uses a neural network to identify time-efficient objects relatively reliably. After a little testing, we were able to train a Yolo-v4 artificial intelligence through darknet (framework for artificial intelligence) with about 120 images of the 6 objects from the simulation. The recognition of the objects also worked relatively reliably with only a few false detections, but unfortunately, the processing time of each image was a bit too high. Processing the images at a resolution of 255 x 255 pixels worked only with 2 FPS on a mediocre laptop. We suspect this is mainly because docker, as we used it, does not allow GPU blading. While performance probably would have been better on the competition machine, it was also important for us to be able to test our software and image processing with acceptable performance.

So, we looked for another method and came up with the simple idea of basing recognition on colors. We looked at the different objects and defined color spaces that are only found in this one object and do not occur "naturally" in the environment. We were able to recognize the objects quite well this way. However, there were still isolated pixels that happened to be in one of the color spaces we defined for the objects. To filter these false detections, we added a minimum number of pixels that had to be detected for an object to count as an object. We also filtered out false detections by setting a maximum and minimum size. Additionally, we compared if the area per detected pixel is too large. That

way, even more, reliable results were achieved than with the Yolo-v4 algorithm. Although, it must be admitted that this method probably only works that well in the simulation since the lighting conditions do not change and so the colors remain quite similar. The new method is also much better in terms of processing time. With the new method, it is 7 FPS on said mediocre laptop. Unfortunately, we had great difficulty recognizing the nettle weed, since the corn plants have similar colors.

Since in our tests the simulated camera published images at different speeds on different PCs, independent of the real-time factor in Gazebo, we decided to have the speed of the robot dependent on the publication rate of the images. This ensures that the route traveled is fully scanned for objects.

Our tactics worked well. Even if we did not recognize all objects, we at least recognized a large part of them. Unfortunately, the robot also knocked over some plants and once turned into the wrong row after one of the turns. Nevertheless, we managed to be placed again, this time 3rd place.

3.2 Task 4 - Weed / object removal

In Task 4 the additional task, besides navigating the field and recognizing the objects placed on the ground, was to somehow transport the items within the field to the edges of the field. Here, in order to get the full score, a distinction would have to be made, as one type of object should be brought to one side and the other type of object to the other side.

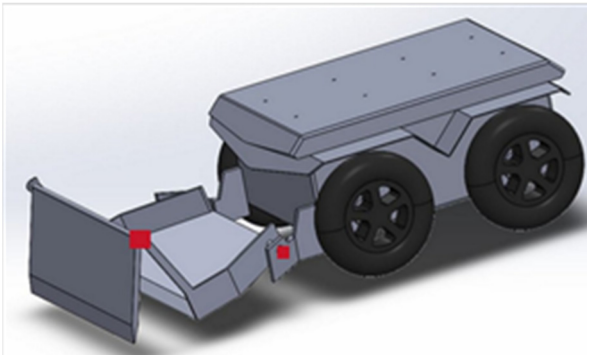


Fig. 8: CAD of the Task 4 robot model

However, we decided to only try to get points for picking up and transporting the objects, mainly due to time constraints. Our idea was based on only modifying the Jackal demo robot to take over the software unchanged from Task 1, 2, and 3. So we constructed a first model of a Jackal with a shovel attached to the front and transferred this to the Gazebo simulation. The shovel should be controllable via ROS. For this purpose, two virtual "motors" were placed at the red positions in Fig. 8, in order to be able to lift the complete construction and to be able to execute turns, for example, without being stopped by unevenness. The second motor should rotate continuously while the construct is on the ground and thus push objects onto the bearing surface. At the end of the row, the robot could lift the complete shovel construct again with the first motor, so that the storage surface is slanted and the collected objects are deposited between the robot and the bucket construct in the headland.

Unfortunately, we had problems shortly before the competition, because the simulation could no longer be started together with the robot. Therefore, we were unfortunately not able to participate properly in Task 4 in the competition and present our idea. Nevertheless, we at least got a point for the attempt and got 3rd place.

4 New innovations (real and virtual)

4.1 Task 5 – Free Style

This year, we challenged us to develop a prototype that is able to drive in steeper and just more challenging environments such as vineyards. Therefore, we have built a robot with a special hinge, an articulated joint, in the middle.

The prototype does not lose any or at least minimal traction due to this special hinge in the middle. It allows the robot to bend itself a bit up and downwards and mainly sideward to maximize the contact between the tires and the ground.

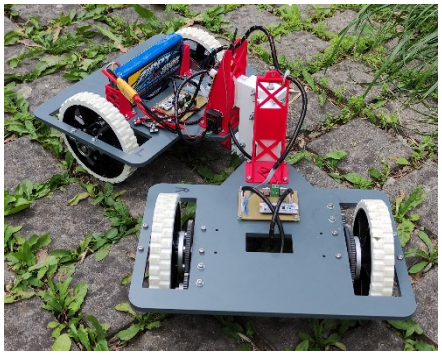


Fig. 9: Our robot the "Knicklenker"

Robots like this already exist, with the joint being used to enable smaller turning radii, but mostly the turn is created by expressing a force on the hinge, meaning that the hinge causes the turn. Our prototype as well has an extremely small turning radius for his size, but the hinge just follows along. We turn by having each wheel equipped with a motor and its speed calculated and controlled individually by micro controllers. Making a turn by giving the outer wheels a higher velocity than the inner wheels, allows the turns to be more optimized and concludes in less slippage for the tires.

Another feature of this robot is that it controls its path by itself. If the rotary encoder, located at the hinge, detects a different angle than it should be at, for example for a turn or even if it should be just going straight forward and it is slightly off, it reacts by itself to change the velocity of the wheels to come back to the positioning, which it should actually be at. This is useful in many different cases to stay in on the right path.

5 Conclusion

Of course, this event was very unusual for all parties and had both advantages and disadvantages. To only concentrate on developing the software and to optimizing it in a virtual environment was certainly an experience that will help us a lot in the future. We have further optimized our already good software and have learned how to deal with a virtual environment. In the future, we will try to test and optimize our robot in the simulation, even if the event takes place in real life because this is often easier and faster than testing the software on the real robot.

Overall, we won 3rd place, which we think was mainly due to the good image processing, the stable basis for navigation in general, and the method for acceleration in rows.

6 References

Figures 1 to 9 created by Jonas Mayer

Other issues

The Team Carbonite sponsored by: Micro Macro Mint, Wilhelm Stemmer Stiftung, Sick AG, Volksbank Überlingen

CERES II – CERES TEAM

Andy Dehn [1], Marc Philipp Funcke [1], Maximilian Grote [1], Christine Hoffmann [1], Jochen Korn [1,*], Matthias Nießing [1,*], Natalie Peracha [1], Jannis Wagner [1]

[1)] *FH-Münster – University of Applied Sciences, Department of Mechanical Engineering, Germany*
[*)] *Instructor and Supervisor*

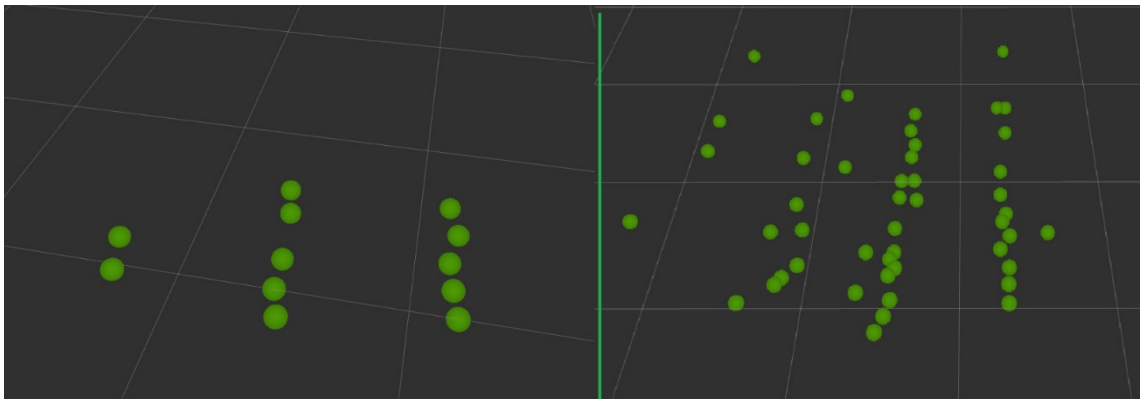
1 Introduction

The Robot Operating System (ROS) is a framework for writing robot software [1]. The team has been using ROS since the beginning of the project. This means the team members already knew the functionality of the robot operating system. However the transfer from the real world into the gazebo environment brought numerous challenges. For tasks 1 and 2, the team had to use the given Clearpath Jackal robot and its sensors. This led to probably the biggest challenge: The usage of the laser scanner for plant detection. Since the field robot event 2019, the team has decided to completely switch to depth cameras for plant recognition. Thus, there was less or no experience with the usage of the sensor data from the laser scanner and it was quiet a big challenge to rewrite or adjust the algorithms accordingly. For task 3 and 4, the team was able to use the own robot model, the CERES II, with its given sensors. This gave the possibility to use the well developed detection algorithms for depth cameras. However the integration of the own robot model into the simulation environment brought some new challenges.

2 Navigation in a virtual field

The Clearpath Jackal robot was equipped with a color camera and a laser scanner as sensors for environment sensing [4]. As it is for similar to the depth cameras, the team has decided to just use the laser scanner for navigating in the virtual field. The robot utilizes differential drive for its movement. Because the CERES II uses the same concept for steering, most of the controllers could be reused with the Jackal robot after adjusting the parameters.

2.1 Task 1 – Basic Navigation



Picture 1: Plant position results from LIDAR (right) and depth cameras (left)

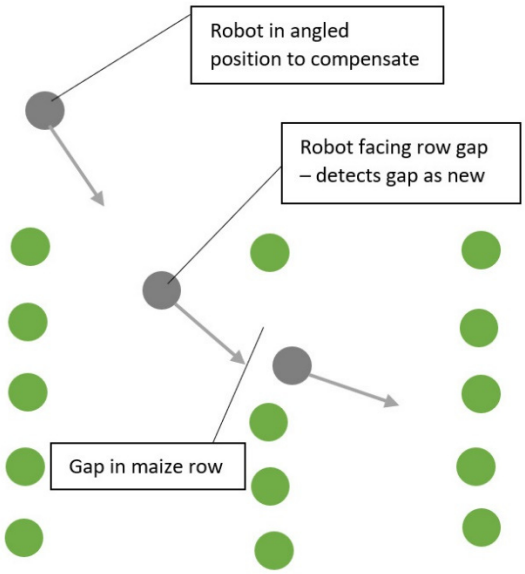
To navigate through the rows, the robot detects individual plant positions and tries to drive through the middle of them. Because the lidar detection returned also many error detections (e.g. leaves in the middle of the rows), it was mandatory to average the plant positions to get a good robot behaviour. However this approach can lead to mistakes, which can be seen in task 2. In picture 1 the different results from the plant detection via laser scanner and depth camera can be seen. When the robot reaches the end of the row, a separate node detects the row end and changes the state machine of the robot into the turn state. In this state given movements will be made to drive into the next row.

The strength of the control-algorithms and robot speed can be tuned via a pid controller. Because the team often tested the robot in simulation worlds with a rough terrain texture, it decided to use lower speed and higher control settings. As it turned out, the maize field in the competition would have allowed for higher speeds and lower control settings. So in the competition run, the robot performed safely and accurate, but in comparison to other teams, covered significantly lower distances. The team is still happy with these results, as safety was one of the main priorities. However for task 2, the settings were changed to a more risky, higher velocity approach.

2.2 Task 2 – Advanced Navigation

In comparison to task 1 the biggest challenge of the second task is the navigation in the turns. Despite the parameter changes, the row drive remained the same. However the turning manoeuvre now contains longer parallel driving in the headlands to fulfill the given driving pattern. The turns were done by the same node as in task 1, however the node was given the driving pattern as additional information. The robot relied on odometry data for the drive next to the rows.

This simplified approach can lead to problems as it can be seen at the mistake the robot did in the competition. When fulfilling a long parallel drive in the headland, the robot drove a bit too far to enter the next row due to unexpected sideways movement when turning on the spot (which can happen when using differential four-wheel-drive). As a result the robot had to enter row with an angle. Unfortunately this lead to the robot directly pointing into a gap between the plants in one of the rows. The averaging row drive then interpreted this gap as the middle of the row and drove the robot straight into it, which lead to the robot damaging the plants. This scenario can be seen in picture 2. Up to this point, the team was quite happy with the performance of the robot in the second task. With the usage of the row driving algorithms with depth cameras plant detection, the angled row entry probably could have been compensated. However the headland drive is definitely a spot for improvement, e.g. the plants next to the robot could be used for improved navigation.



Picture 2: Driving mistake that occured during the event

3 Applications in a virtual field

For task 3 and 4 the team used a gazebo model of their real robot CERES II. This allowed for usage of the same code and sensing algorithms which have been developed for the real world robot. The robot is equipped with two depth cameras in the front to get a bigger field of view. The depth cameras are Intel Realsense D435. They offer a depth image with a range up to 10 meters [3]. Furthermore an inertial measurement unit in combination with wheel encoders is used to get an estimation of the robot position. These sensors were represented in the gazebo model. The parameters for noise and other sensor errors were chosen to represent the real world application as best as possible [2].

The CERES II is equipped with four Wheel Hub Motors and is utilizing differential drive for steering. Here as well the gazebo parameters have been chosen to represent the real robot behaviour as best as possible[5]. The same has been done with masses and inertia of the robot body. For the visual representation of CERES II in gazebo and rviz, CAD data was used to create a 3D mesh of the robot. In addition camera picture were used to create a texture for the model. The real world CERES II and its digital gazebo representation can be seen in picture 3.

Picture 3: Real world ceres (right) and its digital gazebo representation (left)

3.1 Task 3 – Weed / object detection and mapping

In comparison to the software for the jackal, a slightly changed row drive algorithm was used. This was due to the change from lidar to depth cam plant detection and thus allowed to use the well developed systems for the real robot CERES II. For the driving pattern the robot used the standard behaviour from task 1.

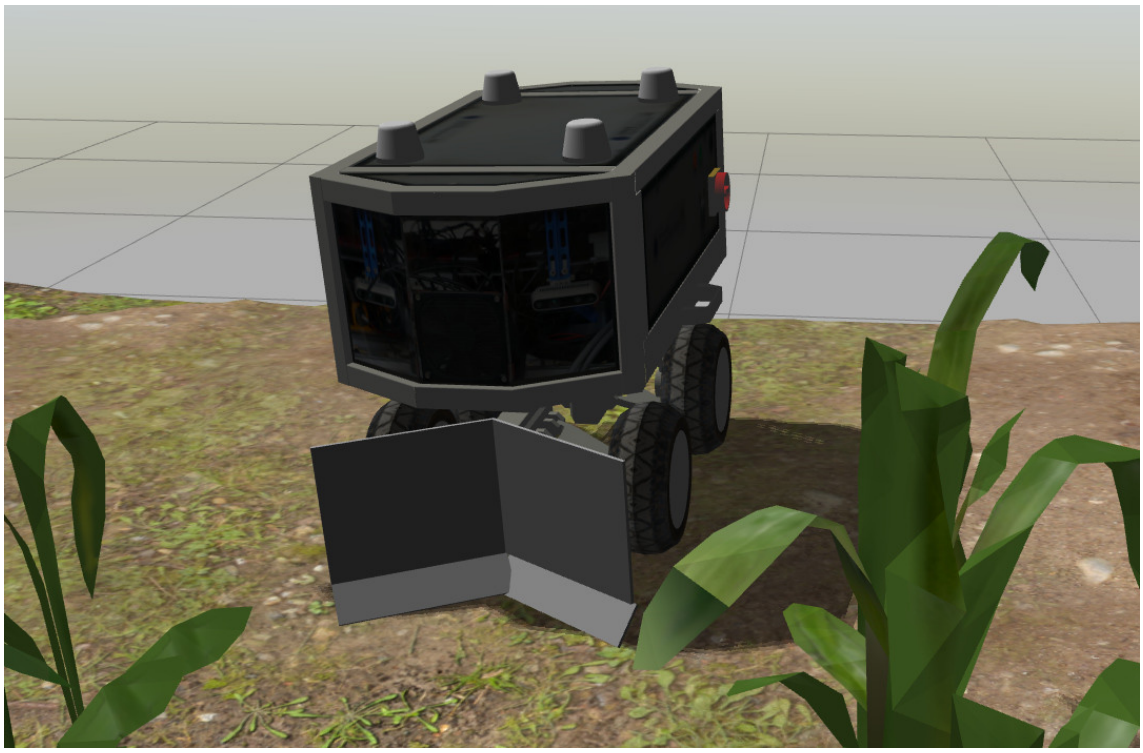
The object detection was realized with colour image recognition. The trash cans and bottles were distinguishable from the other field objects due to their colours. So with simple HSV-filters it was possible to detect the trash candidates in the field. To verify these candidates as trash and to avoid multiple detection of the same object, the depth image was used. With the position of the trash objects from the colour image, the real world coordinates could be gathered from the depth information. The coordinates are saved, so in theory, a map could be possibly generated. But the error from the odometry were too big, so just after one row, the coordinate drift was way too high to get an usable map, so the team decided against submitting the map in the task. The team was quite surprised to see that the odometry in the simulations was worse than the odometry of the real robot. However up until the event, no possibilities of improvement were found.

Due to time limitations the team decided to not recognize weeds in the fields. It turned out, that detecting weeds in the field is more complicated than detecting waste, because the weeds have the same colour as the maize plants. This is a topic the team wants to develop further in the future.

The team was quiet happy with the performance of the robot in the contest. One of the most important aspects was the safe and accurate driving algorithms to avoid plant damage. Within its limitations the recognition worked and detected the possible objects. In hindsight the message declaring the detected objects should have been delayed until the robot reaches the object itself. In the past, the message was published as soon as the object was sensed, which in some cases lead to too early detections. In the future the team plans to expand the recognition to detect weeds and to improve the mapping capabilities of the robot.

3.2 Task 4 - Weed / object removal

To move the objects within the maize field, the team decided to mount a shield at the front of the robot. With this approach, the robot tried to push the objects through the rows to the headland. That means only objects in the middle of the row can be removed whereas objects in between the plants are ignored. The ideal solution to pick up all objects would have been an robotic arm. Because of the limited time before the event and the faster speed to remove an object, the simpler approach of the shield was chosen. The shield was attached to the front of the robot and could be raised and lowered via a position controller. As there is no possibility to model a soft object in gazebo another approach has been used to compensate the uneven floor. Picture 4 shows two flaps attached to the bottom of the shield. They are freely rotatable and meant to lower onto the ground. By this approach the team tried to avoid, objects slipping under the shield.



Picture 4: CERES with attached shield

At the beginning of the task, a separate program evaluates the given map to determine in which rows the objects are located. It then generates a driving pattern, which tries to push the objects into the desired headlands. If there are objects of different types in one row, they all were pushed into the same headland as no object separation was possible with the shield. The driving pattern was generated similar to task 2 which meant the turning program could be reused. For navigation within the rows, the standard row drive algorithm was used.

During the competition unfortunately, a problem occurred which led to bugged behaviour of the shield. Probably it was caused by an error of the shield controller, which lead to the shield bouncing on and off the ground. This means objects could not be pushed through the whole row. The error might have been caused by poor simulation performance or different soil resolutions. After the competitive run, the team was happy to have the possibility to show the shield working fine on their local

computer. In this scenario, the task could be fulfilled within the limitations of the shield quite well.

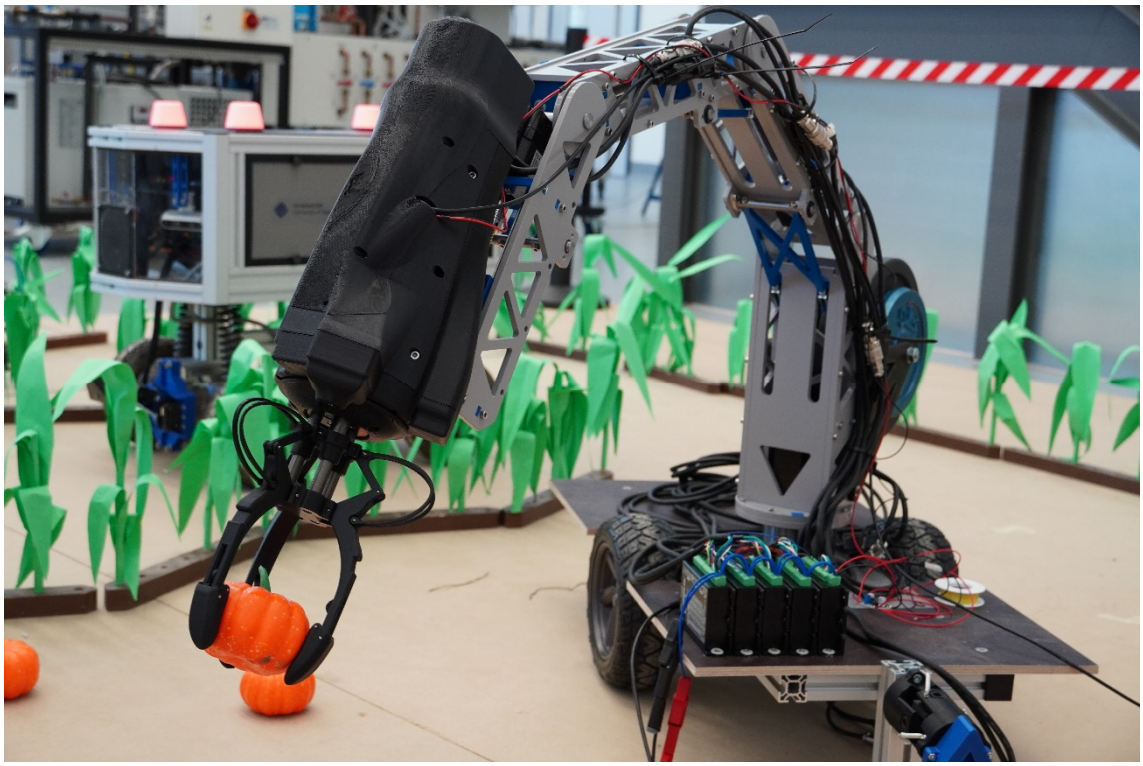
4 New innovations (real and virtual)

4.1 Task 5 – Free Style

The idea of building an robotic arm results out of the problem that the robot was not able to interact with his environment. So it wasn't even able to compete on task 3 or 4 at the FRE. To get more flexibility for future tasks the team developed an arm with the following requirements:

- Operation range up to 1 meters
- Able to use multiple tools
- Able to dismount the arm from the robot
- High precision for lifting objects

To capture these points the project was split in the three following parts. Picture 5 shows the complete system.



Picture 5: Robotic arm on a trailer

4.2 Trailer

Because of its operation range the arm was expected to get very big. This could lead to problems while navigation through the fields. By using a trailer combined with the automatic hitch from the robot, it is able to connect and disconnect the trailer on its own. Because of its size the trailer also has to carry the computational unit and batteries for powering the arm.

4.3 Robotic arm

The arm is based on an excavator arm. Instead of using a hydraulic power unit, it runs with stepper motors which powers threaded rods. The arm is split in four segments. Each of these segments includes one motor and one threaded rod to lift the next one. The most important advantage of using this combination, is the fact that the arm stays in his current position, even if a blackout would occur. Another advantage is the fact that the arm runs with less power intensive motors which leads to cheaper components.

4.4 Robotic hand

Since the robot hand should be flexible in its applications, the team had to design a decoupling system for the tool head that would allow to attach other components for future tasks. The motors and sensors were to be reused for each tool, so the interchangeable part of the tool is completely mechanical. The motion to rotate the tool is transferred via friction. To operate the tool, there is a rod that exerts a force that is converted by the hand into a closing movement. Opening the fingers is done by several 3D-printed springs. A force resistor measures how hard the object is gripped. This allows the hand to grip a lot of different objects without damaging them.

5 Conclusion

As the previous sections described, the switch from the real robot to a simulation model broad a lot of challenges. But thanks to the great work of the organizers with the development of the virtual environment, a really powerful simulation was built up. This means that after overcoming this challenges, the team was able to use the simulation environment, especially with their own robot model, really successfully. Despite some drawbacks, the virtual fields seem to be a good representation of the real world. The team will probably use the simulation environment and robot model, that were built up leading to the field robot event, in the future for the further development of the robot.

Concerning the performance in the event, the team is very happy, especially with the tasks 3 and 4, which were done with the own robot model. It was really nice that the event could take place despite the Covid-19 pandemic even though only in the virtual world.

Finally, the CERES Team wants to thank the Department of Mechanical Engineering at the FH Münster – University of Applied Sciences for sponsoring this project. Without their consulting and financial aid, the team would not have been able to participate in the contest.

6 References

[1] Morgan Quigley, Brian Gerkey & William D. Smart (2015). Programming Robots with ROS: A Practical Introduction to the Robot Operating System, page 3
[2] Tränkler H.-R., Reindl L. M. (2018). Sensortechnik: Handbuch für Praxis und Wissenschaft: Springer Vieweg
[3] Intel Corporation (2020). Intel RealSense 400 Series Product Family, URL: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf (last visited: 29.07.2021), page 14

[4] Clearpath Robotics. Jackal_description Package, URL: https://www.clearpathrobotics.com/assets/guides/kinetic/jackal/description.html (last visitied: 29.07.2021)

[5] ROS.org. Using a URDF in Gazebo, URL: http://wiki.ros.org/urdf/Tutorials/Using%20a%20URDF%20in%20Gazebo (last visited: 29.07.21)

COLLECTHOHR - AGCHOH ROBOTICS

Daniel Mayer[1], Johannes Bringsken[1], Christoph Brüning[1], Philipp Tegethoff[1], Sebastian Biederwolf[1], Lars Krämer[1], David Reiser[1*], Jonas Boysen[1*]

[1)] *University of Hohenheim, Institute of Agricultural Engineering, Technology in crop production, Garbenstr. 9, 70599 Stuttgart, Germany*

[*)] *Instructor and Supervisor*

1 Introduction

For the Field Robot Event 2021 we have a completely new team. We first met in the winter semester 2020/2021. Only a few members had experience in programming. It helped a lot that the virtual model of the machine was provided. We use the provided model for task one and two. For the task three and four we programmed add-ons for the base model. At this point, special thanks to David Reiser and Jonas Boysen for the great support before and during the event.

2 Navigation in a virtual field

2.1 Task 1 – Basic Navigation

The first task this year was to have the robot navigate through curved rows of maize, turn around at the end and drive through the next row. The winner was the one who covered the longest distance within three minutes without damaging the plants. In addition to plant parts protruding into the rows, the uneven ground also posed a challenge. Because of this, we decided to use a combination of laser scanner and IMU data for navigation.

The basis for the navigation was data from the LIDAR sensor, which was transformed into a point cloud.

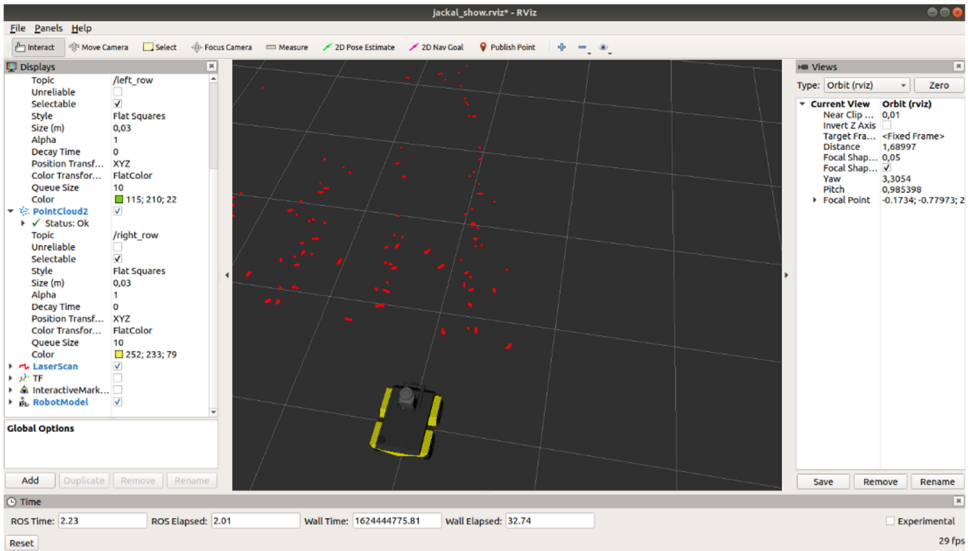


Figure 1: Scan points of the LIDAR sensor

This point cloud was read in and the relevant areas to the left and right of the robot were separated out by applying a "BoxFilter". The field of view was limited to 100 cm to the front. To the side, the viewing area was restricted 15-70 cm to the left and right of the vehicle respectively. To minimize interference from out-of-row scan points, outliers were filtered out using the "Radius Outliers" filter.

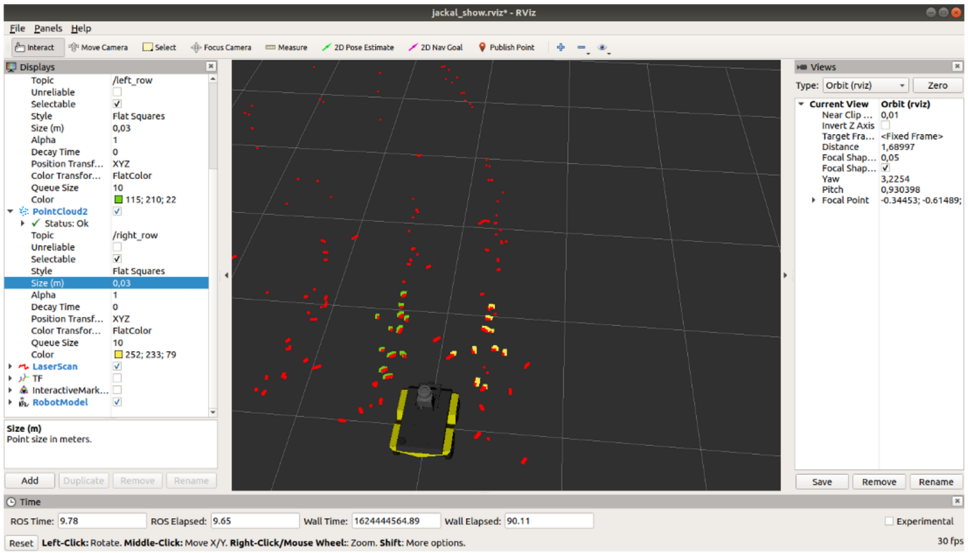


Figure 2: Scan points with box filter, distinguished in left (green) and right (yellow)

From these two new point clouds, a Ransac line was created to the left and right of the robot. The mean value of the last 15 lines was calculated from the Ransac lines.

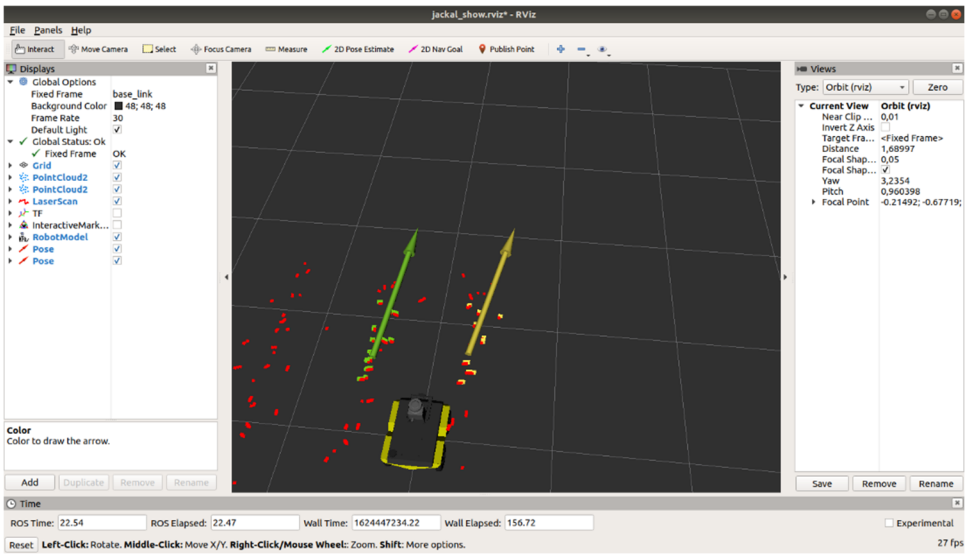


Figure 3: Generated Ransac lines from point cloud of the box filter

Using this mean value, the center point, so-called "row_point", between the maize rows is now calculated. This is generated at a distance of 70 cm in front of the robot.

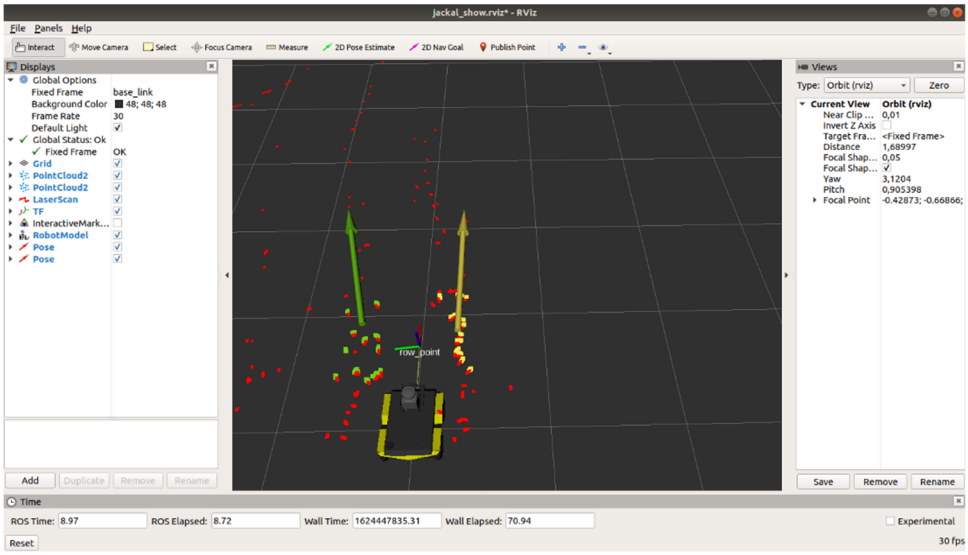


Figure 4: Calculated center point (row_point) between the ransac lines

This "row-point" now acts as a target for the robot, it now follows the point by its travel movements. If the robot comes to the end of the row, no more points are detected, now the "headland_mode" is initiated.

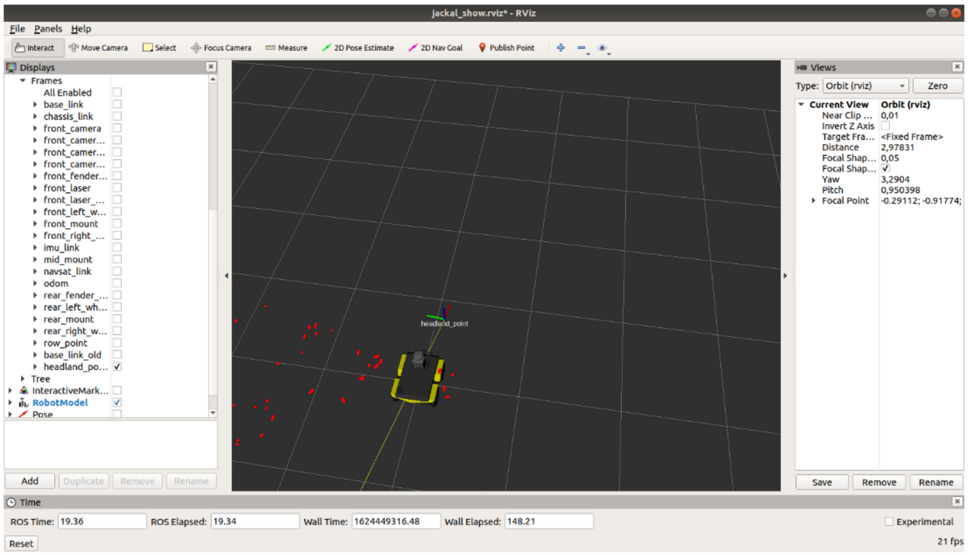


Figure 5: Detecting the end of the row and creating the first headland_point

In "headland_mode" the robot travels along preset waypoints. If it reaches a waypoint, then the next one is preset. The waypoints are stored in a .txt document and contain rotation as well as distances in x and y direction. The first waypoint is set slightly forward.

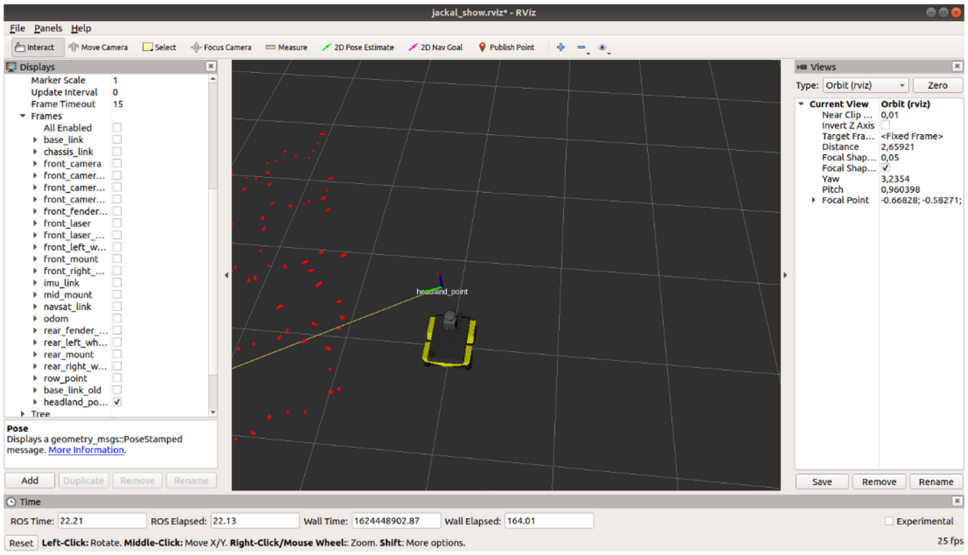


Figure 6: Moving to the second waypoint in Headland_mode

The second waypoint is located 75cm to the right or left of the robot.

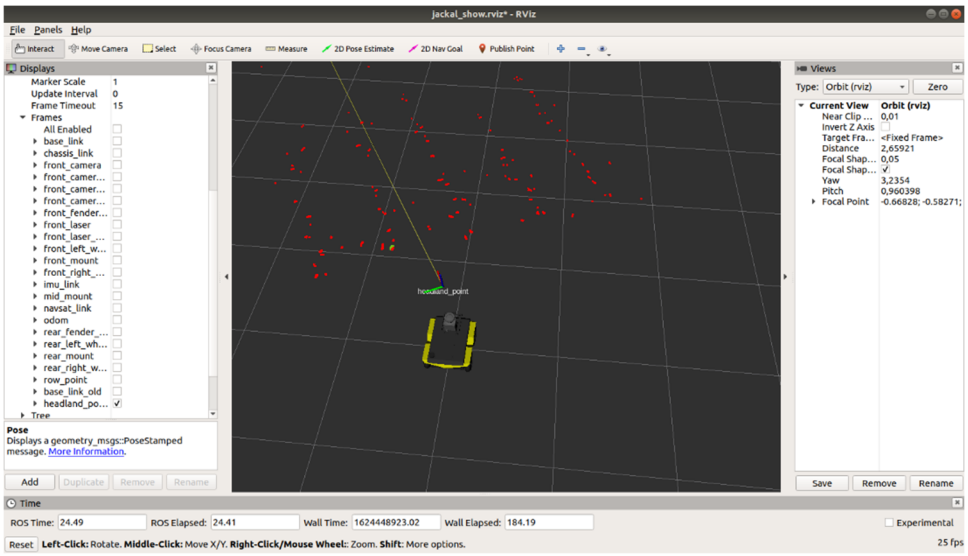


Figure 7: Moving to the third waypoint and then recognizing the rows again.

The last waypoint is again behind the robot in the direction of the maize row. When the robot has passed all waypoints, it should ideally be aligned with the next row. Once it has reached the last waypoint, the robot switches back to "row_navigation" and navigates through the rows of maize again.

A mode controller changes the waypoints after each turn, so that the robot turns once to the right and once to the left.

Discussion: The navigation worked well in most of the cases, occasionally there were problems in recognizing the ends of the rows. The robot sometimes drove diagonally out of the row, as leaves hanging out of the row were incorrectly detected as a curve in the plant row. As a result, there were also problems when driving into the new row. The detection of the row ends would have to be done more precisely, here the laser

data could contain further useful information, it would have to be filtered in such a way that no irritations are caused by leaves sticking out of the row.

2.2 Task 2 – Advanced Navigation

Task two of the event involved navigating through the rows under more realistic field conditions. In doing so, the robot had to bridge missing sections in the rows and more pronounced unevenness in the ground. The field had to be navigated according to a predefined sequence. This sequence had to be taken from a text document.

As a basis, we took the row recognition from Task 1. Because of the misses in the rows, the settings of the box filter had to be adjusted. The visibility was increased to 150 cm. However, the basic detection of the maize rows and the navigation is the same as in Task 1. To keep the order of the row traversal, the given text file was read into a new "mode_controller". This controller processes the given driving pattern. The individual waypoints for the driving pattern are again stored in text files and are called. The waypoints are specified again in the "headland_mode" and the robot moves along them.

The problem of row end detection is exactly the same as in Task 1. The robot should drive straight out at the end of the row. If a leaf is detected at the end of the row, it can lead to a curve being incorrectly detected and the robot trying to follow it. This leads to the problem that the robot stands at an angle at the end of the row.

3 Applications in a virtual field

3.1 Task 3 – Weed / object detection and mapping

In general, our strategy to detect the virtual objects was implemented by using an RGB-camera. It has been placed in front of the jackal, looking straight to the ground. The first challenge was to find a position for the camera that was able to include the complete row width with the area between the maize plants without a second detection of the same object in the next row. So we tested a few positions and with a resolution of 640 x 480 and the distance 75 cm to the base link of the jackal fitted well for the detection tasks.

The main challenges regarding the object detection while driving over the field were the similarity of the weed colours and contours compared with the maize plants and the similarity of the cans and bottles to the ground colours. For this reason, we decided to divide the tasks “detection of weeds” and “detection of waste” in two single programs.

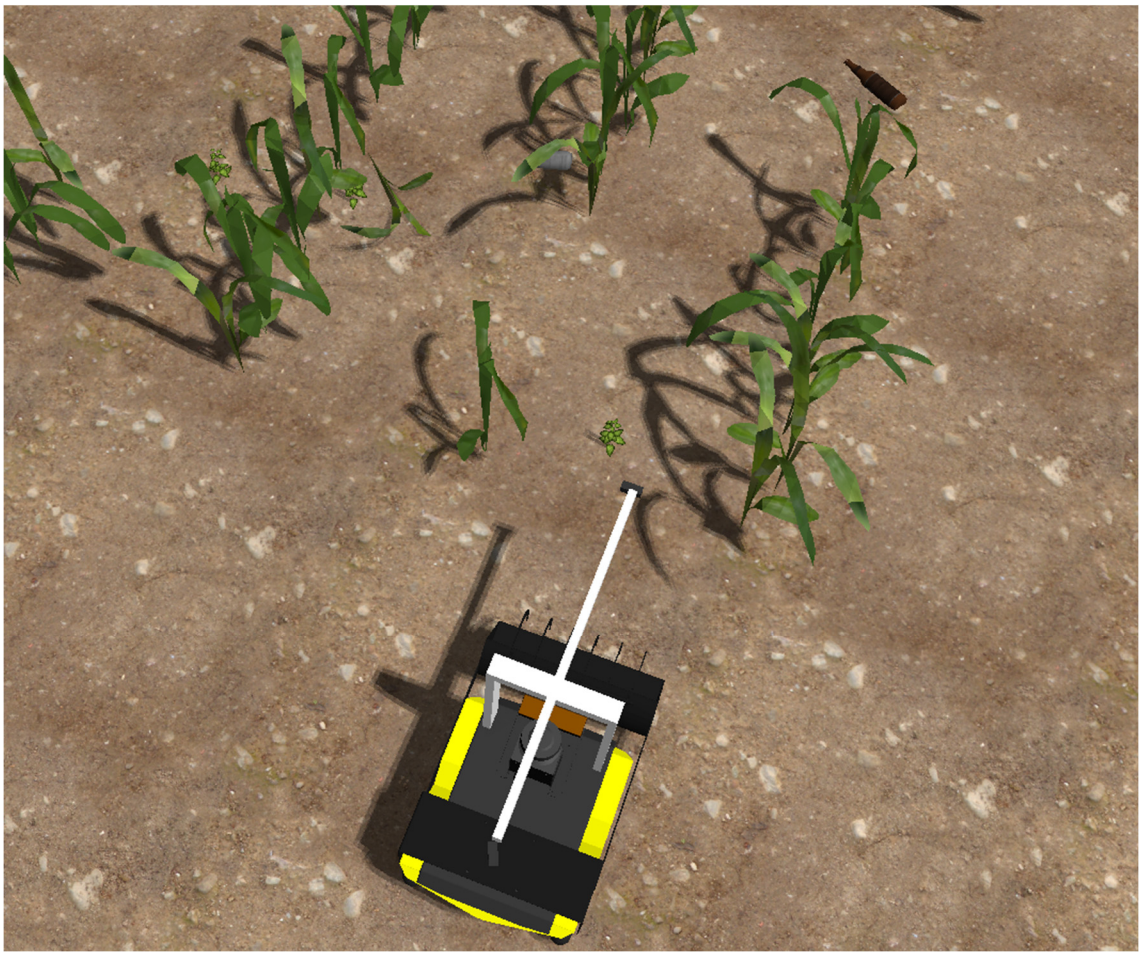


Figure 8: Jackal with RGB Camera in the front (Black Box)

3.2 Task 3 - Weed Detection

Because of the similar colours of the maize plants and the weeds, we were not able to extract the contours of the weeds only by using a colour filter. Whenever we tried to get only the weeds by using small ranges of RGB-filter values the detection on the go was not stable enough because of the changing floor and light conditions.

So we used an alternative way with an Excess-Green filter on the raw image to get the weeds as well as the maize plants and a contour detection what worked much better. To avoid the wrong declaration of maize plants as weeds, the next step was to find a method to find more differences between the maize plants and weeds.

With the given camera position, it was a good approach to exclude all contours that begin outside the camera resolution edges and are shown on the left and right side of the camera image. This had the effect that all maize leaves standing in the rows are excluded safely.

This was already a big success but there were some mistakes left when driving over the field. Sometimes the maize leaves and ground points were still detected as weeds so we added circles around every found contour. With these circles it was possible to define the dimensions of the weeds and extract them safely from the maize. The result of our weed detection node is shown in figure 9.

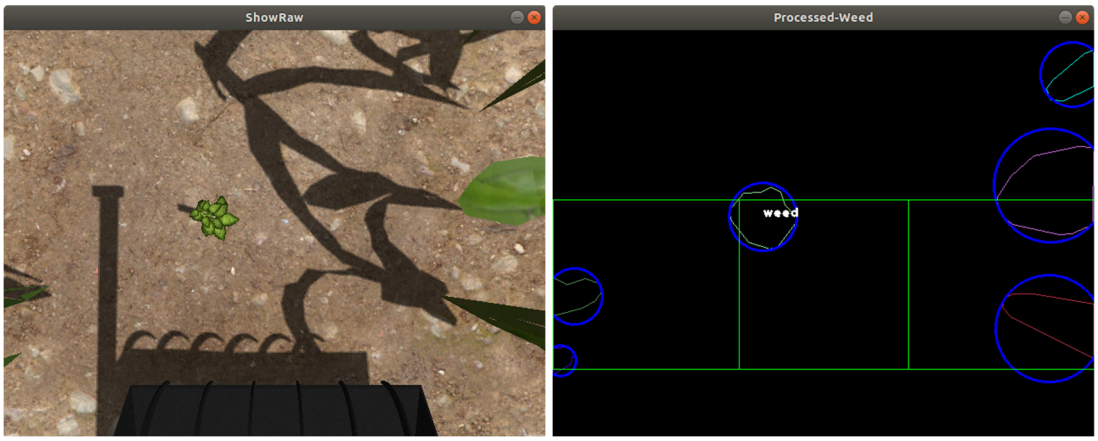


Figure 9: Raw image and Processed Image for Weeds

3.3 Task 3 - Waste Detection

To detect the cans as well as the bottles we used similar approaches as in the weed detection. To get their contours it was easier than extracting the weeds from the maize plants because of the colour difference between the bottles and cans on the one hand and the ground on the other hand. All green and light-brown (and others) except the can and bottle colours were filtered so only their contours appeared in the process-image.

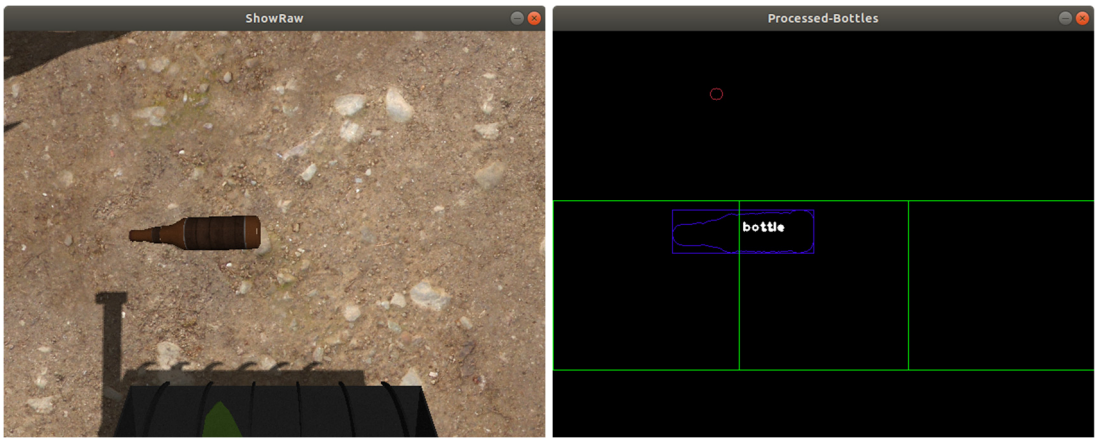


Figure 10: Raw Image and Processed Image for Bottles

To become safer in detection while driving and to avoid wrong detection of ground structures as “waste” we also added bounding contours around the detected waste structures. In comparison to the weeds the bottles and cans have more in common with rectangles than circles so we chose this as the favourite bounding contour. With the dimensions of the rectangles that were automatically fitted and turned to the contour in the image we were able to detect bottles and cans safely.

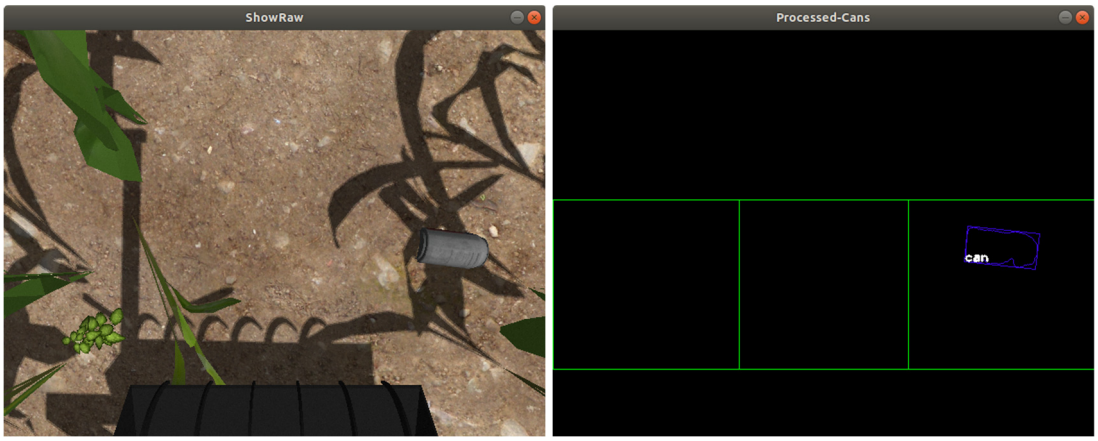


Figure 11: Raw Image and Processed Image for Cans

3.4 Task 3 - Getting the reference coordinate system

The next step was to reference the found objects to the pillars in the field map. Fortunately, the jackal was able to reference his own position to its start by using the odometry with the encoders. The big task was to find a way to reference its start point to the pillar coordinates.

The two pillars differed in a QR-Code with a text-output “Pillar A” or “Pillar B” after scanning. To identify if the jackal starts at A or B we implemented a second camera in the back of the robot also looking backwards in the direction of the closer pillar. The second camera was necessary because the first one is looking directly on the ground and the pillars can not be displayed. The second camera has also the advantage not to turn in the beginning what saves time for detection.

With the implementation of the package *zbar_ros* the second camera scanned the close pillar before the row navigation and object detection started. The Process is shown in figure 12.

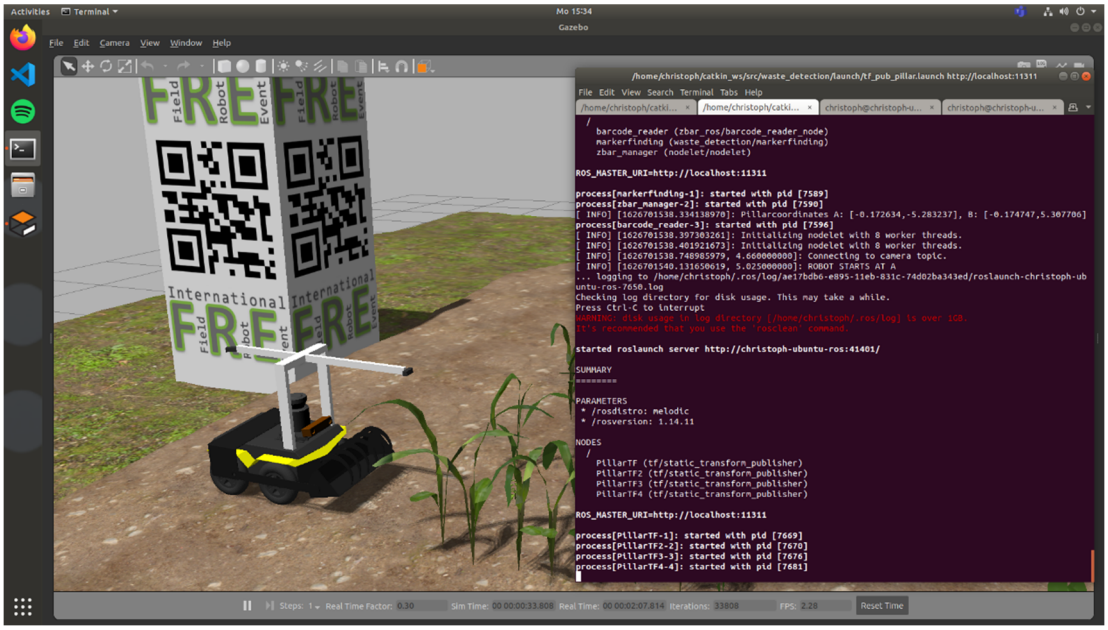


Figure 12: Scanning the closer pillar QR-Code

To get also the position of the odometry to the first pillar we used the data of the LiDAR Sensor. In the scan the pillar is shown with two visible edges.

The ground dimensions of the pillar are square so with the longer edge (one edge is always completely visible for the LiDAR) we were able to find the main coordinate system of the first pillar. The transformations are shown in the figure 13.

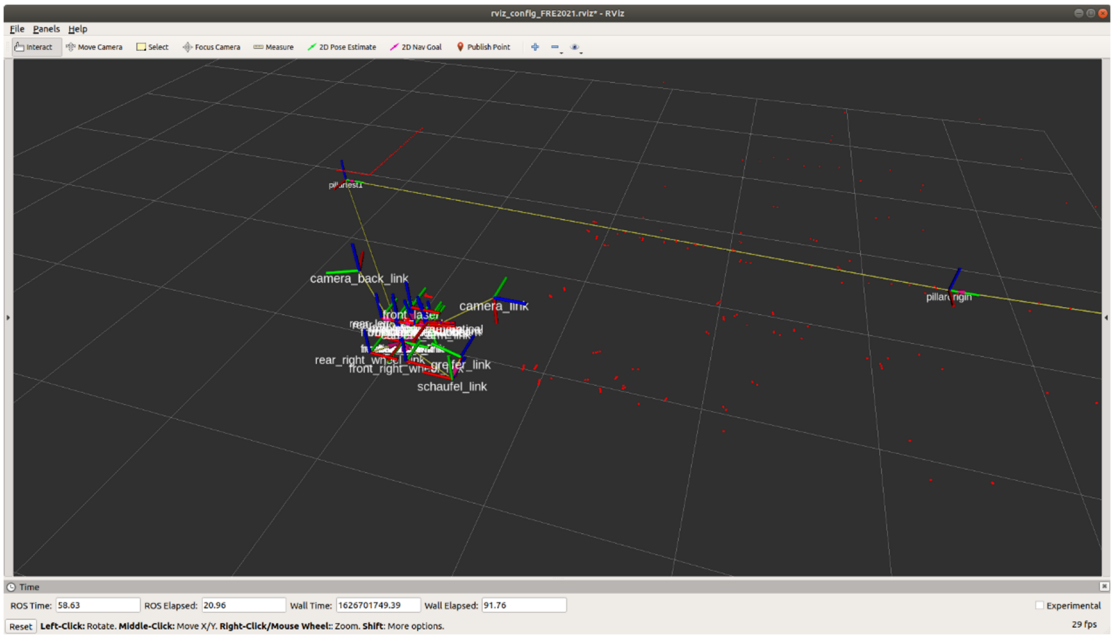


Figure 13: Transformations to the first Pillar from odometry point

The robot position to the main coordinate system of the field what we called “pillarorigin” the last step was to read the coordinate values of the pillars from the given *.csv*-file.

With the positions of pillar A and pillar B from the file data we did a static transformation to the main coordinate system. (Figure 14) From the robot over the first and second pillar the reference to the field main coordinate system has been placed. Now we were able to reference found objects to this.

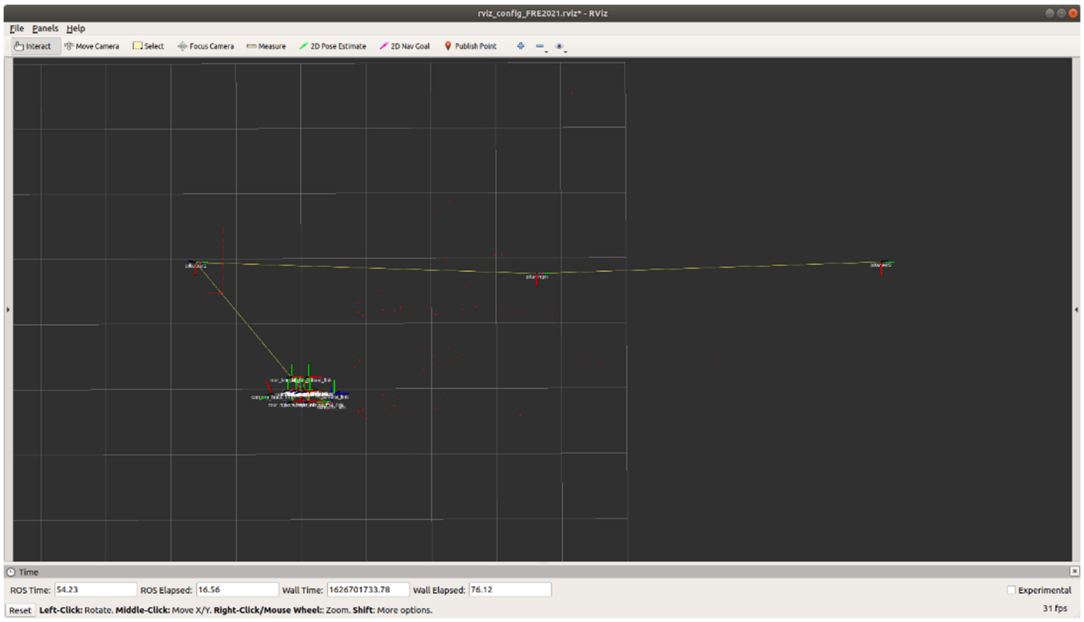


Figure 14: Transformations to the main coordinate system and second pillar

To get more accurate positioning instead of the actual base position of the jackal we used the X and Y Coordinates of the detection camera to define the object positions.

All in all, the performance of the object detection worked well. The jackal was able to find the positions of cans and bottles safely, also most weeds were detected. One weed that stood very close to the crop was not found because the contours of maize plant and weed melted to one in the image processing. The referencing of found objects also worked and the csv file has been added well with the found positions.

4 Task 4 - Weed / object removal

4.1 Task 4 - Task

The fourth task of the Field Robot Event 2021 was to remove objects from the given corn field. The objects were beer cans and weeds. A total of 10 objects were placed in the cornfield. The robot had to navigate autonomously through the rows. So the navigation of the robot in task 4 is based on the solutions of the first subtasks. The coordinate and the type of the object were provided as information. If the object was picked up this should be brought to a suitable headland. Weeds should be placed on a different headland than the other object type. The respective groups had 5 minutes to clean the cornfield. The task was rated on two points. If the object was picked up and this was taken to the correct headland. Points were deducted for incorrect headlands. Furthermore, points were deducted for each damaged plant. Another piece of information was important for the processing procedure. This says that objects were allowed to be pushed out of the cornfield, but this does not bring points!

4.2 Task 4 - Concept

The setting of the tasks placed the defaults, admission and placing of objects. This had to be implemented with the robot, Jackal Free. For this, a model was picked out the robot, in order to compile possible principles for the solution. Through brainstorming in the group, promising ideas emerged. One possibility was to pick up an object with a gripper arm and dump it on the headland. There were several considerations to realize this movement. Should the object be picked up like the mechanism of the front loader or via a clamshell bucket. Another option was to pick up the objects similar to a forklift, place them above the robot during travel and lower them again at the destination. With the help of the existing model, we looked at the installation space available to us for the implementation. Another criterion was that the robot should hardly be wider in order to keep navigation as simple as possible, be able to rotate 360 degrees if necessary, and not damage any plants.

A possible pick-up by a clamshell bucket in the center below the robot was ruled out, since the installation space did not allow this. The drive, electrical system, etc. of the robot are located there. Therefore, the option of a front loader and forklift-like pickup was considered.

4.3 Task 4 - Design/ elaboration phase

The two possible implementations were designed in the next step, with the criterion of the narrowest possible design.

The Solid Works design program was used for this purpose. The downloaded Jackal free model was inserted via a step file. The model provided possible fixation points for a structure. Furthermore, the mounting position of the laser, which was important for navigation, had to be taken into account. It turned out that a forklift-like implementation would be difficult. Here, the components hindered the function of the LIDAR scanner.

Thus, the front loader design was pursued further.

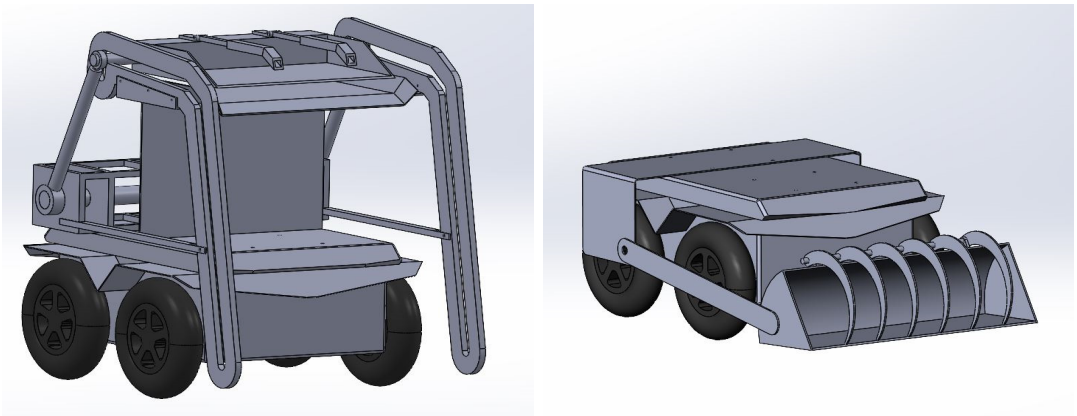


Figure 15: Design concepts

The construction should consist of as few components as possible, since these had to be implemented in GAZEBO in the further process. The assembly was thus created

from 4 components. A baseplate, which serves for the attachment on the robot. Two lateral connectors, which connect the baseplate with the shovel. The bucket which serves to pick up the object. In addition, gripping tines, which facilitates a pick-up and fixes the object during travel. During the design, it was important to place the drawing origins in the rotation points of the movement in view of the further use of the components in the GAZEBO environment. In GAZEBO, the components are rotated exactly around this point after later programming. This is an important point to be considered!

5 New innovations (real and virtual)

5.1 Task 5 – Free Style

We didn't participate in task 5.

6 Conclusion

All in all it was a really good thing to participate in the virtual world of the FRE. Although a contest on a real field would have been a great experience we are happy to be part of the virtual event in 2021.

Other issues

The Team want to thank the following companies for their support in components:

Sick AG, Mädler GmbH and Nanotec GmbH & Co. KG

Parts of the code used in previous Field Robot Events can be downloaded here:

https://github.com/hohenheimdr

FARMBEAST - BIOSYSTEMS ENGINEERING

Miha Kajbič[1], Urban Kenda[1], Gregor Popič[1], Domen Toš[1], Rajko Bernik[3], Miran Lakota[2*], Jurij Rakun[2*]

[1)] *Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia*
[2)] *Faculty of Agriculture and Life Sciences, University of Maribor, Slovenia*
[3)] *Biotechnical Faculty, University of Ljubljana, Slovenia*
[*)] *Instructor and Supervisor*

1 Introduction

The current pandemic situation impacted the way students from the FarmBeast team (FarmBeast, 2021) prepared for the usual activities involving the competition at the international Field Robot Event. The typical approach is to mobilize the students from the existing group at the beginning of the winter semester that continues the work they started in the previous years. In addition to this, they also act as student mentors for the recruits from different faculties from the University of Maribor. For the newly shaped team, a series of workshops is prepared each year to help the new students start the work and improve the mentor students' knowledge for the new event. These workshops also include courses where they get familiar with hardware parts of the FarmBeast robot and software components, including the ROS meta operating system (ROS, 2021).

However, due to the pandemic situation, the FRE 2021 took a bit of a different course. Keeping the physical contacts to a minimum, most of the work during 2021 was done online. These also presented one of the biggest challenges, where the work on the actual robot progressed very slowly, depending on the current limitations. On the other hand, work in ROS started before the actual rules and guidelines (Field Robot Event, 2021) were published by preparing the group via online meetings and tutorials. When the rules were finally shaped and published, each team member got their responsibilities and started to work.

The second challenge represented the simulated environment. The Stage (Stage simulator, 2021) was used in previous years, so the Gazebo environment (Gazebo, 2021) was entirely new for the team members. Due to good instructions by the organizers (Field Robot Event, 2021), the work progressed with some minor stepbacks where the environment was upgraded and improved, which meant that the corresponding code might not work anymore and had to be improved.

The final challenge was the docker containers (Docker, 2021) introduced quite close to the actual competition. This raised another obstacle for the teams where we successfully submitted the code for the 1st and 2nd tasks. Our new model changes for the 3rd and 4th tasks worked well while testing the code, but unfortunately, we were unsuccessful in submitting it to the event due to technical issues.

Regardless of all the challenges introduced by the situation around us, we could compete one more time at the international Field Robot Event and do great by being 3rd in the Advance navigation and 2nd in the Freestyle tasks. The following subsections describe how different algorithms were developed for the Field Robot Event work and improve

the operation of our robot. Hopefully, we will have the chance to test an improved version of this code in person at the Field Robot Event 2022 against other robots, not just in simulation.

2 Navigation in a virtual field

To actuate the robot in a virtual field (Field Robot Event Github repository, 2021), we must send commands regarding the desired velocities. The velocity, however, consists of two vectors that control angular and linear velocity, each consisting of the x,y, and z-axis, as shown in Fig. 1.

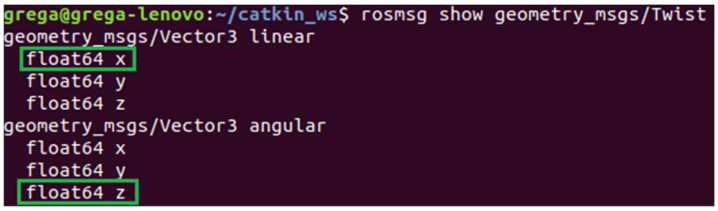


Figure 1: ROS message format for sending velocities.

Because we are driving in the X-Y plane, we were changing only the x component of the linear velocity and the z component of the angular velocity. We did that by publishing the /jackal_velocity_controller/cmd_vel topic at the end of each iteration of the algorithm.

2.1 Task 1 – Basic Navigation

We started by creating an algorithm to navigate the robot between simulated rows of corn plants in the first task. This should not be that hard in theory, but we quickly came across some problems because the environment, depicted in Fig. 2, was built to behave realistically.

Figure 2: An example of a simulated field environment.

The shape of the corn plants caused the first problem we encountered. The leaves were hanging in all directions, and they caused interruptions in the measurements of the optimal maze wall. Therefore, we added a part responsible for taking all the measures from one side, adding them together, and dividing them by the number of measurements,

to get an average number. We did this for both sides, and because the robot was the (0,0) point of the coordinate system, the left side had negative values, and the right side had positive values. When adding both values together, they produce 0 when the robot is driving on an optimal path. The algorithm is shown in Fig. 3.

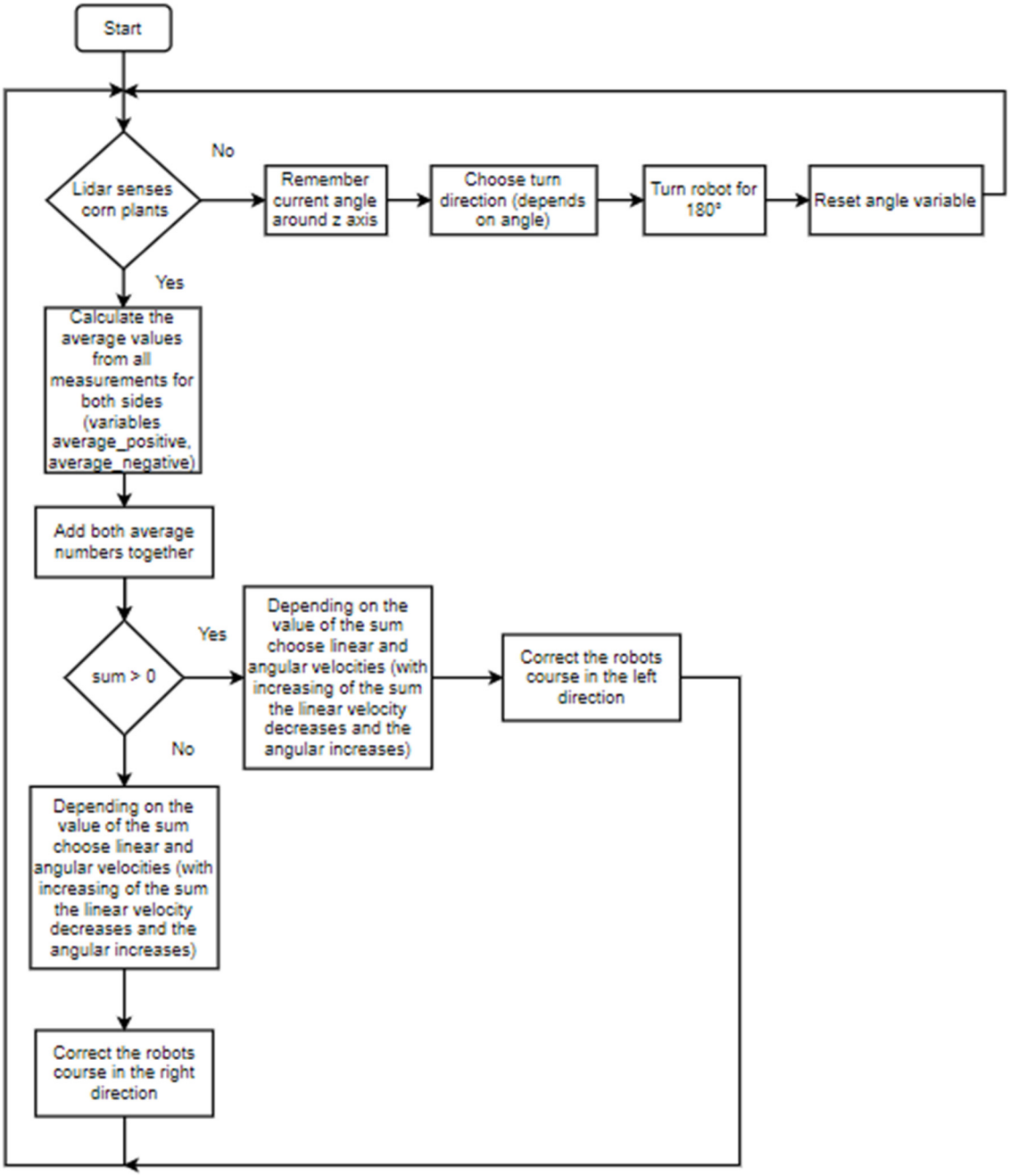


Figure 3: Flow chart of the algorithm that adjusts the linear/angular velocities for optimal driving conditions.

This quite effectively solved the problem caused by the corn leaves, but another problem was the uneven ground, which sometimes caused the robot's wheels to slip. We solved this problem by making four zones with different angular and linear velocities, depending on the distance between the robot and the corn plants. The closer to the corn plant the robot is, the lower the linear and higher the angular velocity is.

This solved the problems of the row navigation, but we also had to take a turn at the end of the row. We first tried doing this with the help of LiDAR readings but soon found out that it was not very reliable. After some thought, we decided that using IMU data would come in handy, as we could get the exact rotation of the robot around the Z-axis. At the end of the row, we, therefore, put the angle into a variable and then simply wrote the program so that it would turn until it completes a 180° turn and, after that, continues to drive between the rows.

During testing the code, the solution worked well, but at the main event, there was a problem that we did not take into consideration. The simulation started just a little above ground, and the first measurement of the sensors produced a result that would indicate that we would need to start turning. That is why our robot drove at the start in the third row instead of the first. Other than that, our robot did not drive over any corn plants, which means that our program worked very well except for the part that we missed to test. The problem was not hard to fix, and it was something that we will undoubtedly pay more attention to in the future.

2.2 Task 2 – Advanced Navigation

In the second task, we needed to upgrade the program used in the first task to drive through rows that have corn missing and turn into rows, decided by a given sequence. The algorithm is summarized in Fig. 4.

The missing corn did not cause us too many problems, as we only needed to make the area that we are taking our data from big enough to still produce enough values to maneuver the robot, even when 1 m of corn might be missing. We already restricted the area of our LiDAR in task 1 to be roughly equal to two triangles that are mirrored over each other, as shown by Fig. 5.

For the second part that had to be added, we first needed to read from the file to learn what the sequence is in the first place. That did not pose much of a challenge, and we completed this step rather quickly. The part of driving into a random row did take a little longer to work out.

We knew that we would have to rewrite a portion of the turning part of the program, and we soon figured out that using odometry would be our best approach. We added an algorithm (Alberto Ezquerro, 2021) that produces an exact distance traveled by the robot in meters (as the odometry topic only gave us how much was traveled on each of the three axes). When we came to the end of the row, we turned for 90° and then drove straight ahead as was specified by the provided file with the pattern, and then turned another 90° back into the row. We also had to compensate for the slipping of the robot, so we added a tolerance that would prevent it from driving too far or not enough.

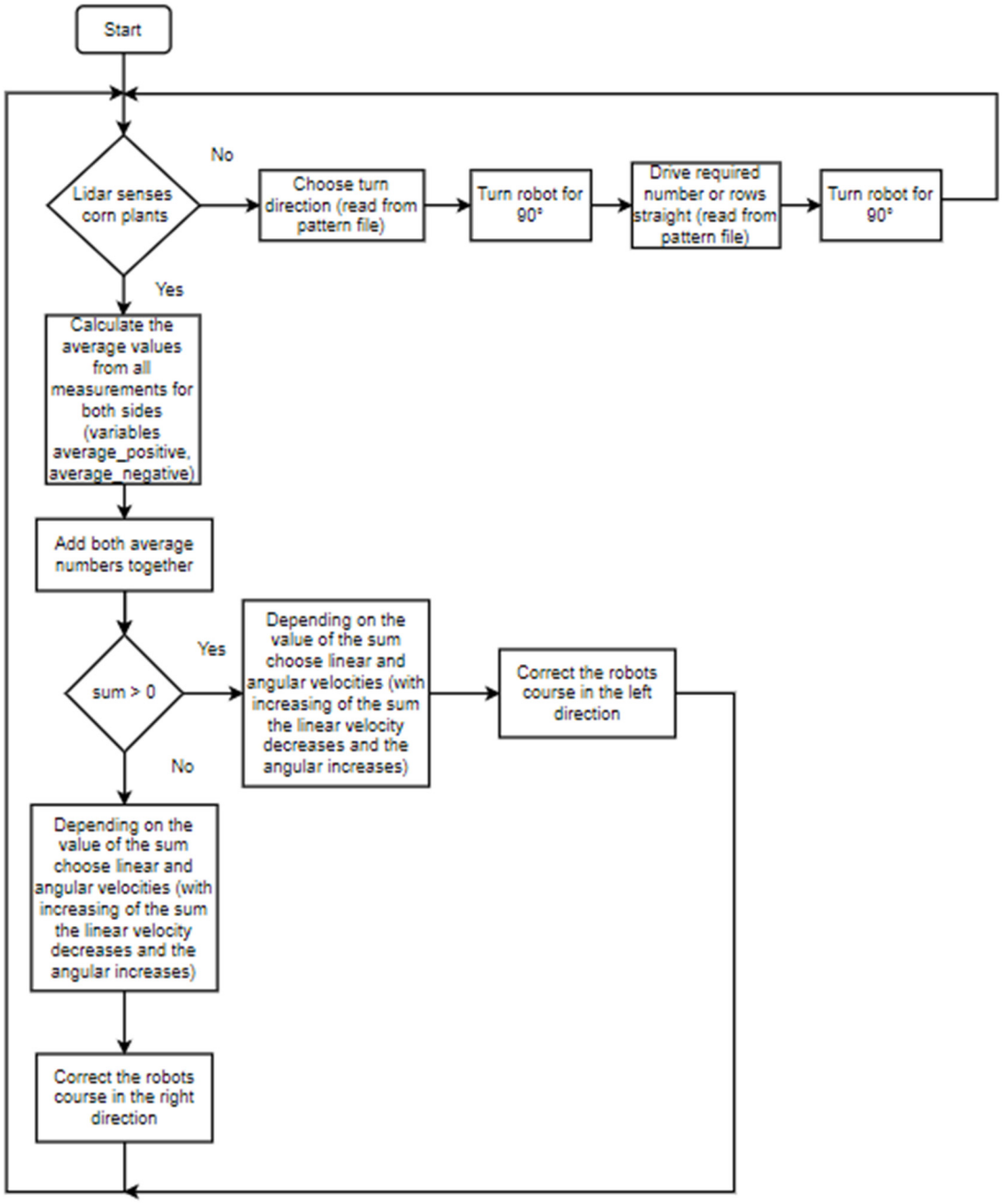


Figure 4: Similar driving algorithm, based on the one from Fig. 3, but with changes to the LiDAR data window and the row entering part.

This program would have the same starting problem and the same problem as in task 1 if we could not fix it quickly. But to our luck, we still had enough time to fix it and resend the program. Our program performed as we had hoped it would during the run, and we were happy with the result, as we achieved third place in this task.

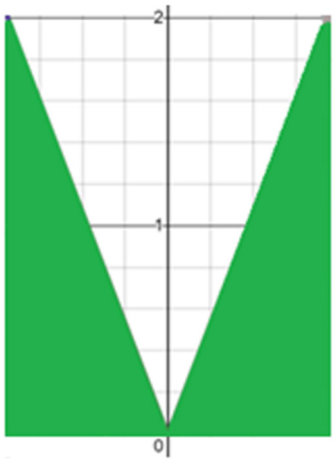


Figure 5: Observation area in meters, based on which the readings from the LiDAR sensors are selected/filtered.

3 Applications in a virtual field

The initial attempt was based on the upgraded Jackal robot model that included a camera faced straight down in front of the robot (Fig. 6), so the field of view captured a square of soil between the crop rows. Due to problems we had to add our custom robot in a docker container, we switched back to the default Jackal robot, which included a camera looking horizontally between the crop rows.

Figure 6: An upgraded Jackal robot model that was prepared for the 3rd and 4th tasks.

3.1 Task 3 – Weed/object detection and mapping

For detecting weeds and objects, we used color thresholding in HSV format (Gonzales and Woods, 2017). The thresholds for detecting weed and litter were determined and set based on empirical tests, where we tried to focus on the object's color that could not be mistaken with the ground. For detecting weeds, we focus on the green with high

saturation, for litter we were looking for gray, red and blue color which was on the stickers of bottles or cans.

Due to the non-optimal position of the robot's camera, caused by the problems of importing the right model in the docker container, the detection was not optimal when we competed with the other teams. The robot kept detecting litter, even if there wasn't any present, producing many false positives. These were produced due to color shades in the sky, which were grey and blue, similar to the colors we use to detect litter (grey cans and blue stickers on brown bottles).

We tried to map detections with the odometry, but we decided to disable the mapping due to the main problem with false positives and uncertain odometry.

3.2 Task 4 - Weed/object removal

Because of unsuccessful attempts to add our model in docker containers and lack of time, we were unable to compete in task 4.

4 New innovations (real and virtual)

4.1 Task 5 – Free Style

The main idea of our new accessories is spraying on the field. We developed two spraying systems: the back attachment and a new tool for the already made front attachment. The main purpose behind both is to reduce spraying by spraying more precisely, causing less soil pollution but achieving the same protective effect.

The back attachment, shown in Fig. 7, represents a spraying system for spraying with fertilizer. It includes a compressor for providing compressed air and another container for storing fertilizer. Compressed air pushes fertilizer through 4 nozzles which can be activated separately. Each nozzle can also be moved in depth to achieve the right spraying distance. Nozzles can be used in two positions: in the vertical mode, they could spray on tall plants from the side, and in the horizontal position, they can spray at smaller plants from the top. The system also includes its own LiDAR sensor, used when the system is in vertical mode, where it detects missing plants in the row. With the use of LiDAR, we reduce unnecessary spraying because it only sprays where the plants are, so we turn off the nozzles if there are holes in rows (missing plants).

For the front attachment, we developed a new tool, depicted in Fig. 8 for spraying with two different herbicides. This tool is supported by a new algorithm, shown in Fig. 9, that we developed for separating plants/weed detection based on narrow and wide leaf sort. The detection algorithm sorts weeds with the use of morphology operation erosion (OpenCV, 2021), where it counts the iterations of erosions needed to remove 10% of the first detected area (U. Kenda et al., 2020). After the recognition, the new tool positions the nozzle to the right location and activates the right nozzle to spray the right herbicide to eliminate detected weed.

Figure 7: Back attachment used to spray fertilizer by positioning the nozzles horizontally (to spray four plant rows at once) or vertically to spray two rows of bigger plants.

Figure 8: A new tool that can be used when mechanical or thermal weed elimination fails. The new tool can spray two different herbicides; to eliminate narrow leaf weeds from wide leaf plants, or remove wide leaf weeds from narrow-leaf plants.

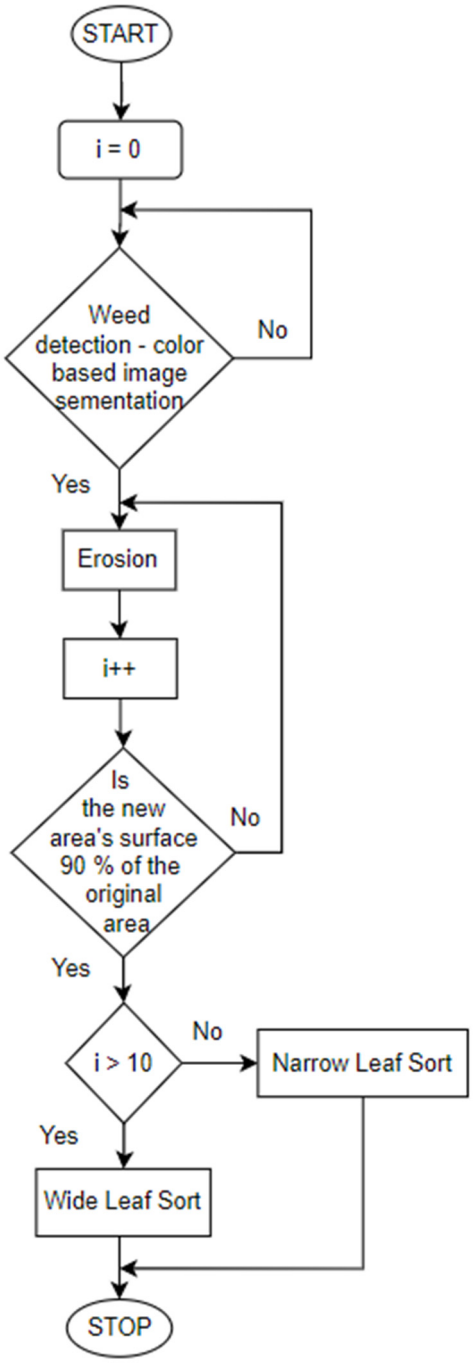


Figure 9: Flow chart of the weed type detecting algorithm.

5 Conclusion

In our opinion, the Field Robot Event 2021 presented a milestone in all Field Robot Event competitions that were organized so far. It was the first event to be held entirely online, which introduced new challenges for the teams and made some changes in the event's organization that could become a part of future events.

There were two distinct advantages introduced this year. The first is the obligatory use of ROS, even for the teams that did not use ROS before. As ROS represents a kind of standard in the robotic's world, this is a step forward for all the teams. The second advantage is the evaluation of the robot's software components that, for the first time,

used the same platform, the same background, and the same conditions. This made it possible to evaluate the algorithms of different teams without the effects of different hardware setups.

Considering the current situation, we did great by achieving one second and one-third place. We know we have even more potential and could have done better if we could solve the technical challenges introduced close to the event. In any case, we had a wonderful experience by joining the Field Robot Event 2021 and will continue to improve our robot also by testing its performance in the virtual world to be ready for the next Field Robot Event.

6 References

FarmBeast website, http://farmbeast.um.si/wp/, accessed on: 28.7.2021.

ROS, https://www.ros.org, accessed on: 28.7.2021.

Field Robot Event, https://www.fieldrobot.com/event/, accessed on: 28.7.2021.

Stage simulator, http://wiki.ros.org/stage, accessed on: 28.7.2021.

Gazebo, http://gazebosim.org/tutorials?tut=ros_overview, access on: 28.7.2021.

Docker, https://www.docker.com/resources/what-container, accessed on: 28.7.2021.

Field Robot Event Github repository, https://github.com/FieldRobotEvent/Virtual_Field_Robot_Event, accessed on: 26.7.2021.

Alberto Ezquerro, The Construct, How to know if robot has moved one meter using Odometry, https://www.theconstructsim.com/ros-qa-195-how-to-know-if-robot-has-moved-one-meter-using-odometry/?fbclid=IwAR32FwDyTGttgcaClav_G5nqREwGcy4W8RCbf49VHaUzHnw6Kz9WaoiNXco, accessed on: 26.7.2021.

Rafael Gonzales, Richard Woods, Digital Image processing, 4th edition, Pearson, 2017.

OpenCV. Morphological operations. https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html, accessed on: 25.7.2021.

Kenda U., Uran S., Bratina B., Rakun J., Belšak A. (2020). Selektivno zaznavanje in odstranjevanje plevela. https://dk.um.si/IzpisGradiva.php?id=77279&lang=eng, accessed on: 25.7.2021.

FIELD BALANCER - CAMPER ROBOTICS

Simon Sure[1], Fabian Paul[1], Erwin Kose[2*]

[1)] *Gymnasuium Thomaeum Kempen, Germany*
[2)] *Independent, Hamburg, Germany*
[*)] *Instructor and Supervisor*

1 Introduction

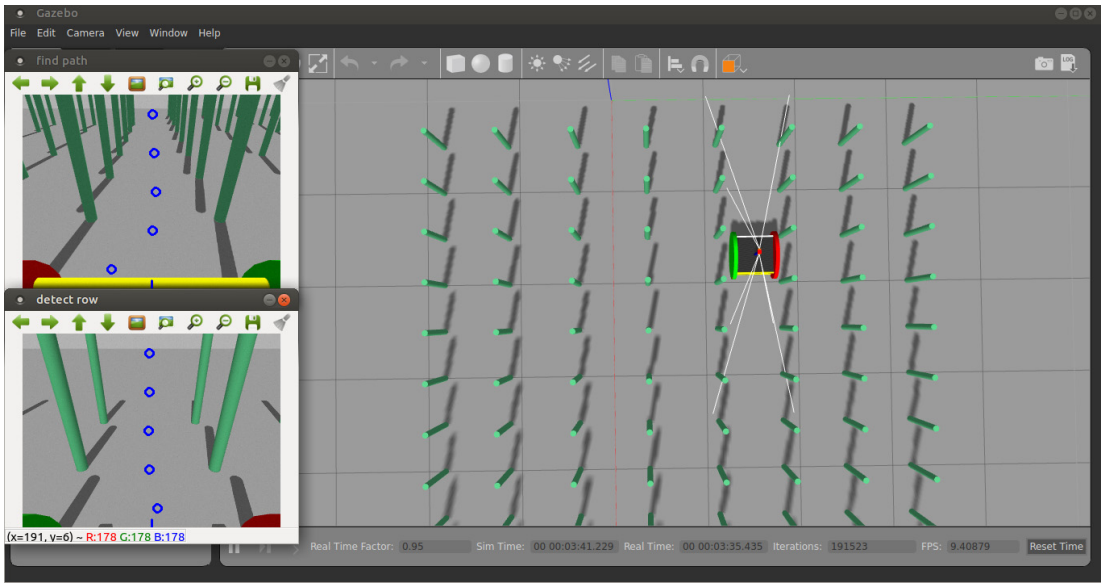
Since we are still a new and very small team of only three members, we had limited ourselves to the tasks navigation and free style.

From the very beginning, we relied on cameras instead of laser scanners for navigation. The cameras we used had a slightly wider aperture angle than the ones we were to use during the competition. The problem was that the camera could not see the plants directly in front of the robot. This detail, which was crucial for us, was not clear to us from the beginning and communicated by the event organizers only a few days before the competition. This made our strategy and our programmed code inapplicable for both tasks 1 and 2.

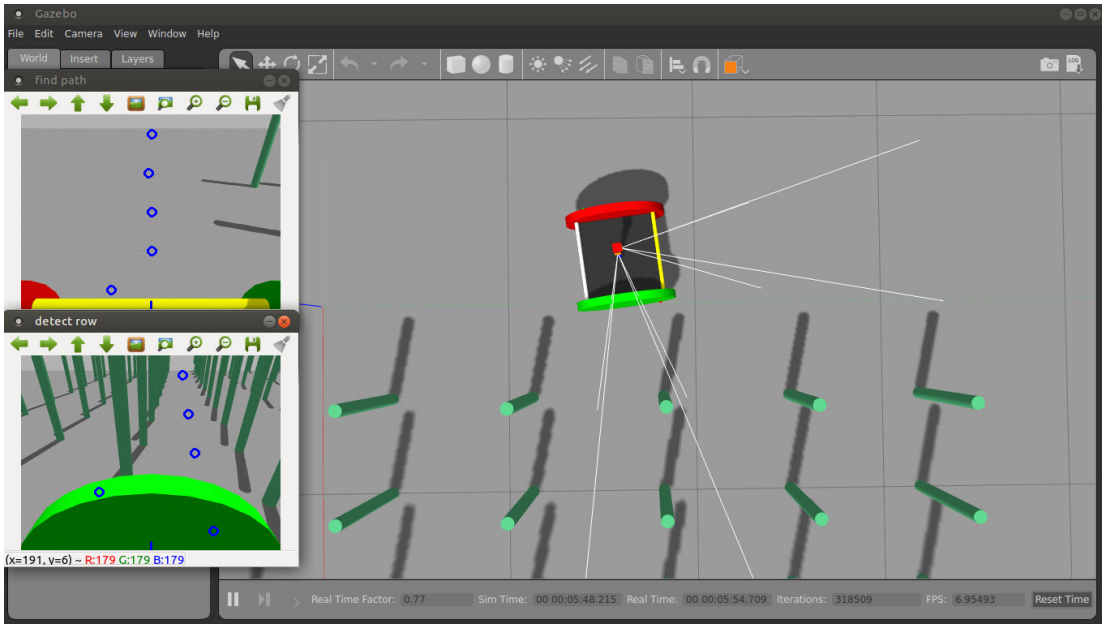
2 Navigation in a virtual field

2.1 Task 1 – Basic Navigation

We used two cameras for navigation. One of the cameras was turnable. While driving through the rows, this camera, in addition to the forward-looking camera, was able to check whether our vehicle was driving in the middle between the rows of crop plants.



Outside the rows, during the headland turn, we turned the camera to the side, which ensured safe entry within the 75 cm row distance.



We have developed a very simple method for safely moving between the rows. We used some standard functions of OpenCV to distinguish the soil surface from the crop plants. By comparing different horizontal image sectors and excluding abrupt path changes, we achieved the exclusion of atypical phenomena such as overhanging leaves or gaps in the plant rows. With a carefully adjusted PI controller we followed the unobstructed path and achieved a decent result.

2.2 Task 2 – Advanced Navigation

Gaps within the crop plant rows are no problem with our navigation method. However, it is different for curved rows, because we could not always clearly determine whether there was a curve in front of us or only leaves covering the path. Therefore, our machine was often getting a little bit too close to the plants.

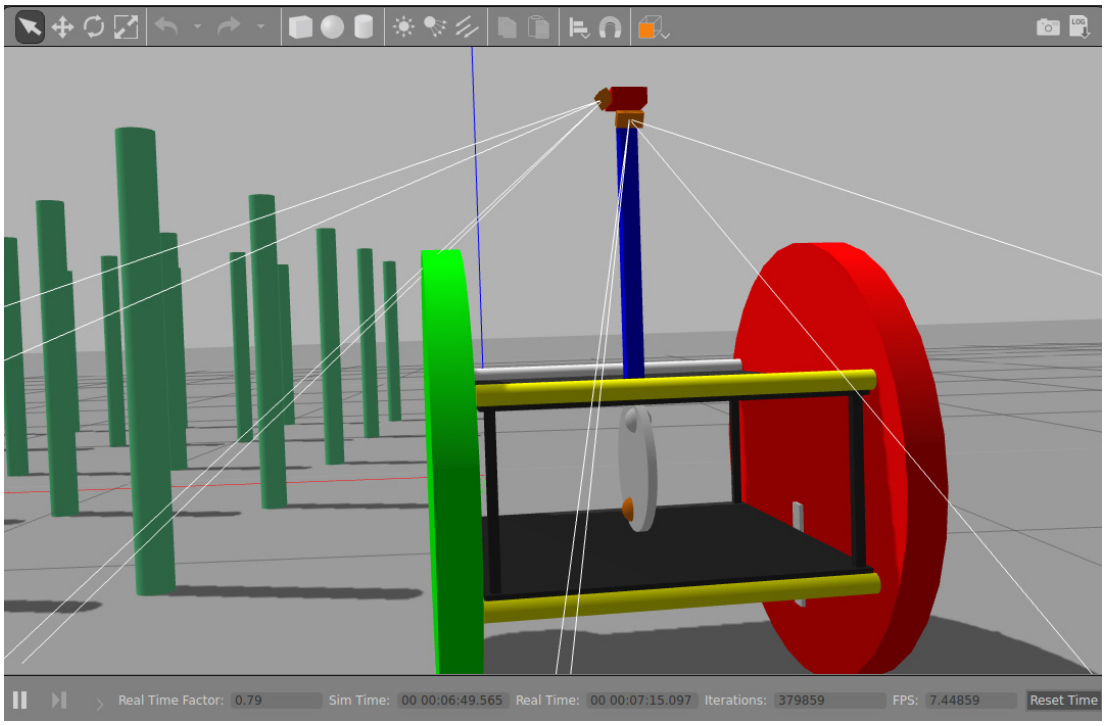
3 Applications in a virtual field

As already mentioned, we had to skip task 3 and 4 and we concentrated on the first two tasks and task 5.

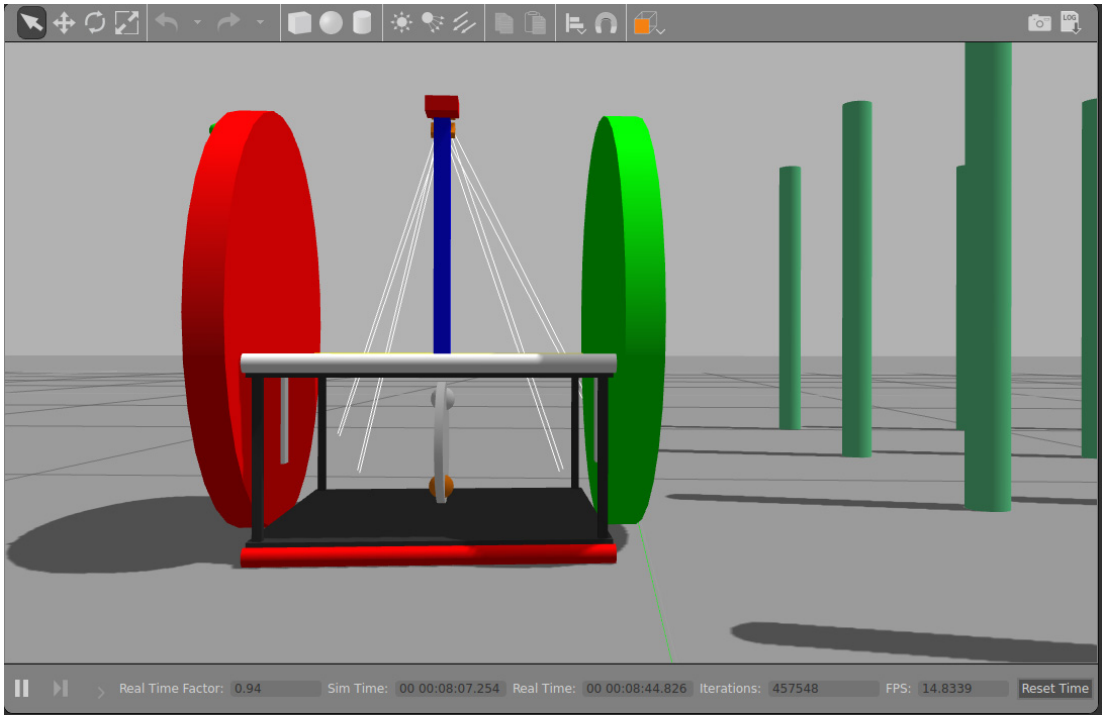
4 New innovations (real and virtual)

4.1 Task 5 – Free Style

We simulated a self-balancing two-wheeled vehicle. The vehicle is not balanced via the drive wheels, but with a reaction wheel. This has the advantage that it does not move back and forth during balancing.



In order to get into the rest position or a safe working position, the robot housing is moved down by 150 mm by means of linear actuators and thus is fixed on the base plate.



5 Conclusion

Our experience for navigating with a simple determination of the free path is promising. To improve it, we will develop a machine learning model for the next competition.

HELIOS EVO - FREDT

Enrico Schleef [1], Christopher Prange [2], Steffen Lohmann [3], Marc Schermus [3], Christian Schaub [4], Tobias Lamping [4], David Bernzen [4], Jan Schattenberg [3,*]

[1)] *Information and System Technology, TU Braunschweig, Germany*
[2)] *Electrical Engineering, TU Braunschweig, Germany*
[3)] *Mechanical Engineering, TU Braunschweig, Germany*
[4)] *Automotive Engineering, TU Braunschweig, Germany*
[*)] *Instructor and Supervisor*

1 Introduction

The Field Robot Event Design Team (FREDT) was founded in 2005 at the Institute of Mobile Machines and Commercial Vehicles at the TU Braunschweig. Since 2006 our team has participated in the annual Field Robot Event with this year's event being the first which was held in a virtual environment due to the ongoing pandemic. This situation, however, affected us significantly since the university's rules did not allow team meetings in presence during the preparation. Thus, we had to rely on virtual meetings using conference tools which required a reorganisation of our work routines and work flows.

2 Navigation in a virtual field

This event included several changes compared to the previous ones. In the first and second task a provided robot or robot model had to be used. The organizer settled on the commercial robot "Jackal" provided by the Clearpath company. It features a skid-steered one with all-wheel drive. For navigation, the robot is equipped with a LIDAR sensor and a camera which are mounted in a central front position on top of its body shell.

2.1 Task 1 – Basic Navigation

In the first task, the robot had to cover as many rows or travelled distance as possible in a limited time, but gets penalty points when damaging a crop plant. This leads to the trade-off between driving fast and driving safe. Due to the robot using a complete different drive train and steering setup as well as a different sensor configuration and mounting position, we were not able to use anything of our current code and simulation environment. Therefore, a lot of work went in to the driving dynamics.

Since the given map was very detailed, the processing power of our computer was too slow to achieve an adequate real time factor during the simulation. The institute allowed us to use one of their computers. While this resulted in an increased performance, it did not solve the problem substantially.

During the competition, the code worked well and the speed was rather conservative so the robot drove a distance of 116 m without causing any plant damage. This resulted in a fourth ranking place.

2.2 Task 2 – Advanced Navigation

In the second task, we were faced with the same problems as during the basic navigation.

The robot finished with a driven distance of 35 m. Due to the crop plant damages and the penalty point the total distance was reduced to 26.2 m. That meant we missed the third ranking place by one point. Overall critical was, that we mainly lost time during the headland turns.

3 Applications in a virtual field

We did not participate in task three and four because we had substantial problems with the docker environment. While we were able to create a docker container, it did not start on the organizer's computers.

4 New innovations (real and virtual)

4.1 Task 5 – Free Style

We did not participate in task five.

5 Conclusion

Our team achieved rank seven in the overall ranking which is acceptable considering that we did not participate in task three and four. It was an interesting experience and in the current situation the virtual online version was the only option to have a realtime event. However, from our point of view it lacked the excitement and broad variety of different approaches and solutions for the different task compared with the event in presence generates.

Other issues

The sponsors of the team were the "TU Braunschweig" and the "Freundes- und Förderkreis des IMN".

MAIZE RUNNER - DTU

Victor Haavik [1]

[1)] *Department of Electrical Engineering, Denmark*

1 Introduction

As part of the course 31388 Advance Autonomous Robots, we had the opportunity to be part of the competition Field Robot Event (FRE). The competition was held by University of Hohenheim, were the purpose was to program a robot to be able to navigate around a field, while doing different kind of task.

Due to COVID-19, the competition were held online, thus changing from being an physical attendance to becoming online competition. This resulted in all the work and development would take place using the Robot Operation System (ROS), where the environment is developed and offered by the host university.

Team

All the team members who were involved in the development of the project, were all Autonomous Systems student, which meant we had the same foundation, and prerequisites for carrying out the project.

Even though all the group members are studying the same master's degree, we all come from different universities and different studies, and thus the overall background is stronger and more diverse. Furthermore, everyone in the group has been able to contribute with different challenges that have arisen along the way.

- Victor Haavik, Design Innovation, DTU
- Frederik Zilstorff. Manufacturing and Operations Engineering. AAU.
- Stefan Larsen. Robotics. SDU
- Isabella Schøning. Physics. Hamilton College

Robot Models

For the FRE two robots were utilized, the Jackal robot and the Maize Runner from DTU.

Jackal

To perform the two first task, the FRE committee had by default given us the Jackal robot, which is developed by Clearpath robotics, thus meaning that the system is fully integrated into ROS. By being a standard component of ROS, a lot of data and instructions on how to use the system is out there and thus making the process easier. The Jackal robot’s full description is as follows:

- W x L x H (cm): 43 x 50.8 x 25
- Weight (kg): 17 kg
- Turning Radius: 0 cm
- Max Spreed: 2 m/s
- Motor Power: 500 W
- Sensor: LMS111 Sensor
- Wheels: Fixed Wheels
- Steering: Skid Steering

Maize Runner

The Maize Runner, which is the robot DTU has used for the previous FRE competitions, is not a standard robot used in the industry. Therefore, its not a integrated part of ROS, and will be needed to be manually develop into Gazebo and the ROS Environment. The benefits of developing and using our own robot, is the degree of freedom which occurs in relation to the location of sensors and the mechanical solutions. The Maize Runner full description is as follows:

- W x L x H (cm): 33 x 55 x 44
- Weight (kg): 30 kg
- Turning Radius: 50 cm
- Max Spreed: 2 m/s
- Motor Power: 200 W
- Sensor: LMS111 Sensor
- Steering: Ackermann Steering

Environment - ROS

Through the iterative process of developing and coding. The environment which were given by the FRE host, changed drastically over time. Where there were notable changes in that environment, for each of updates.

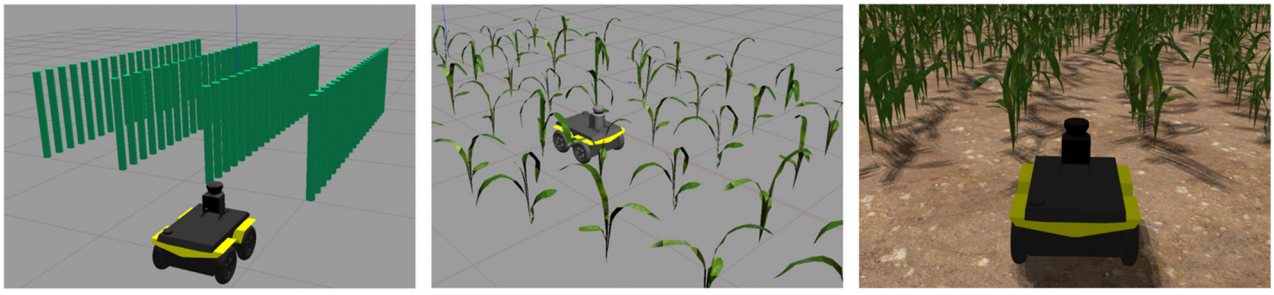


Figure 1: Left: First version. Middle: Change of maize plants. Right: With ground and light conditions

Changed ground and moving corn The first version of the fields, was relatively simplified, thus meaning, when working and understanding how the LiDAR sensor would work in the ROS environment, would change over time due to how much the physical environment would change. Since every release of the FRE environment was so drastically different, in the degree of physical

details, the outcome from the LiDAR sensor would change. In the beginning the outcome were steady and align, because the corn were shaped as robust columns with no leaf, to later on have several leafs, and end i end have a bumpy road as seen in figure 1.

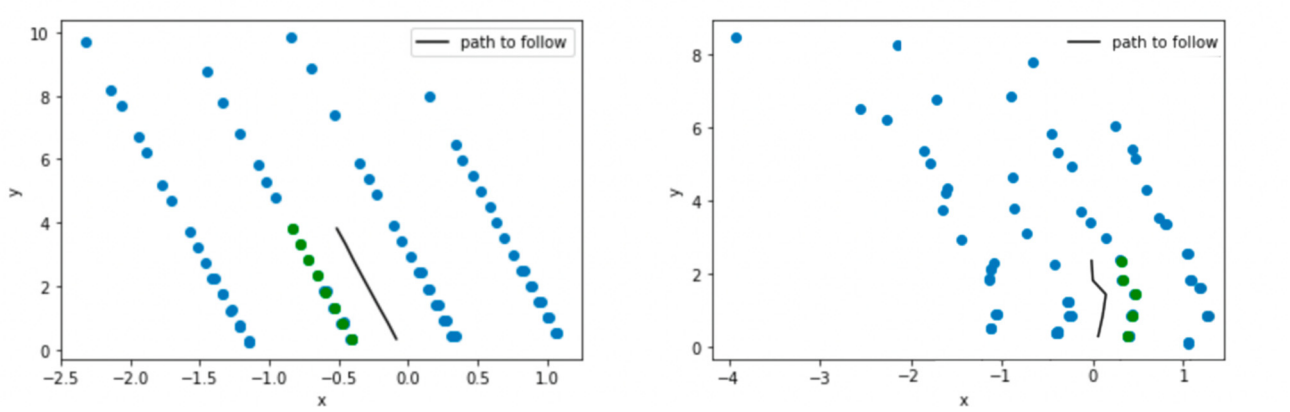


Figure 2: LiDAR data

Due to the changes, a lot of knowlegde of how the LiDAR scanner worked, would be irrelevant later in process, due to the amount of noise captured by the sensor. The viewpoint from the Jackal robot can be seen in figure 2

Development of Maize runner

Since there is limited details on how to implement, develop and calibrate an new system into ROS, a lot work went into understanding the different step necessary to transfer the DTU Maize runner into ROS. Since each group member, only had limited knowledge with ROS, and had only worked with existing integrated robots. Therefor a lot of challenges emerged in this process, due to the lack of knowledge.

CAD model

The first step in the process were to measure up the existing Maize Runner, then it became possible to recreate it in the CAD program Fusion. This was an iterative process, because over time the model would be simplified, to have the same amount of components as the Jackal robot and minimizing the redundant parts, like screws, small metal parts and hubs. The intention of creating the Maize Runner in CAD, were to import the parts into Gazebo and create the robot that way, instead of manually drawing the robot parts in the URDF file.

URDF

Most of the manuals and instructions on how to create different URDFs files, which is stands for *Unified Robotic Description Format*, is unfortunately relatively deficient, and thus a major challenge to create a functional robot. The purpose of the URDF file is to specify the kinematic and dynamic properties, of the different robot joint and the positions of the components (CAD parts)

relative to each other. The firsts versions of the URDF files, which where created, were written manually and thus a lot of erros and defects came along.

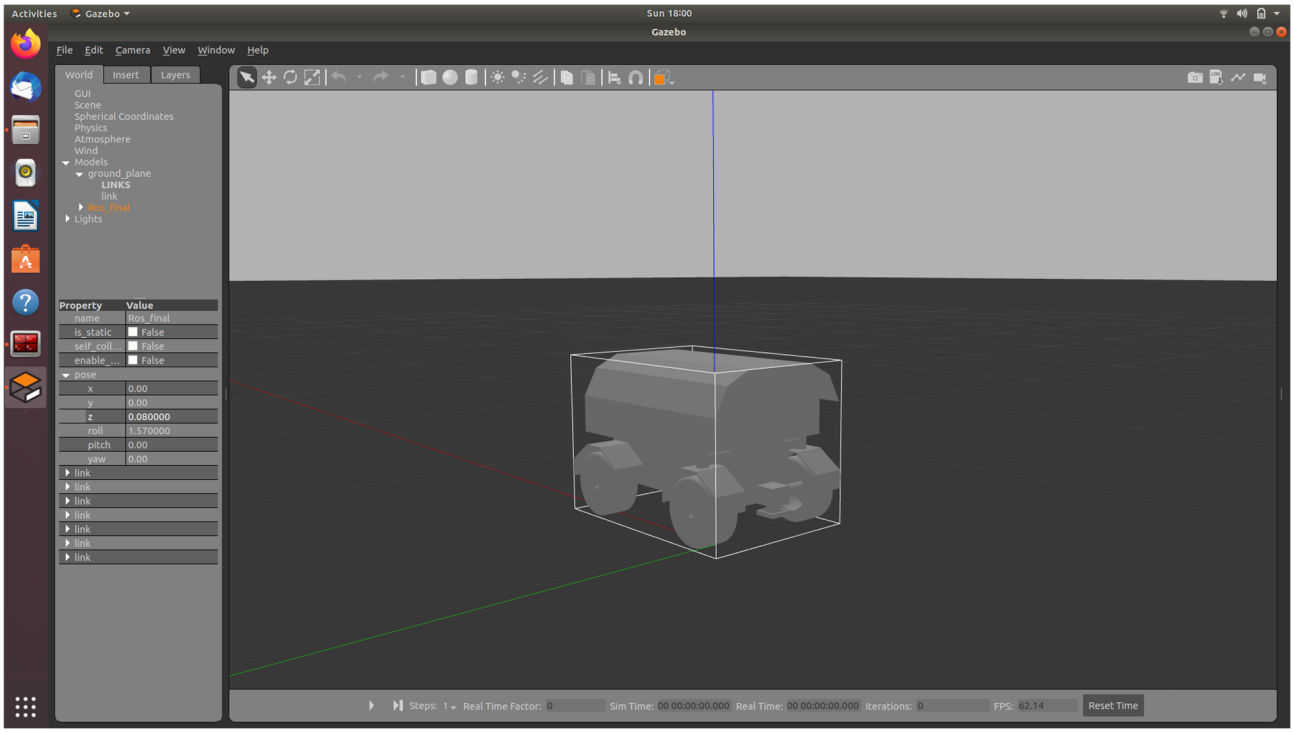


Figure 3: Maize Runner

Instead of manually creating our own URDF file for the maize runner, an plugin for the CAD program were used, that would manually export into an URDF file. This sctipt would also create all the joins and links which were determent in the CAD program. The output as seen in figure 3, and thus a lot of work is still needed to calibrate the robot to be useful in the environment, in relation to use different sensors and motor control.

Deselection of Maize Runner

In the end it was decided not to use the Maize Runner, because of the many problems which constantly occurs when setting up the controller and make it work inside the Gazebo environment. Thus all the energy would be transferred in only using the Jackal robot for the tasks.

Process

To get an better understating on how the system works, a lot of resources went into the exploring how the Jackal robot works in the ROS environment. Since all of the group members had limited experience with ROS, it became important that every team member would play around in ROS, by driving around and learning the different features of the Jackal robot, as well as learn how to work

with the data from the LiDAR scanner can be used. We chose to work in Python as it was the language we all had previous work experience with.

The experience with ROS we had in group, were based on the robot arm Jaco and the turtlebot 3, and therefore we had no prior experience with a small 4-wheel vehicle. Thus before any coding started, we decided to manually drive around with Jackal robot. By getting a feeling on how the Jackal robot would moved around with its fixed wheels, skid steering, and its turning radius compared to its size. In addition the manual movement, it also help to understand how the leafs on the corn, would interact with the robot when moving around the field.

Some of the most imported knowlegde, that came out of this process, where the effect of the sensors placement on the Jackal robot. Since the LiDAR scanner is placed on the top of the Jackal, the outcome would not be as stable as we hoped to, because, all the leaf were individual tracked. Instead of the root of the corn, and thus giving an incorrect picture of the maize.

Working with Data

To develop an algorithm that could process all the data, a lot of recording with the LiDAR scanner were conducted at random places all over the field. One of the remarkable things, that where discovered in this process, where the amount of data point that the Jackal could collected every second, meaning we had a surplus of data and thus we needed to limited the field of view. A path generator could now be develop, from all the data collected from before. This algorithm would find the center between two rows, and thus a way to navigate trough the rows. The first path generator were made trough Jupiter-notebook and not trough ROS, as seen in figure 4.

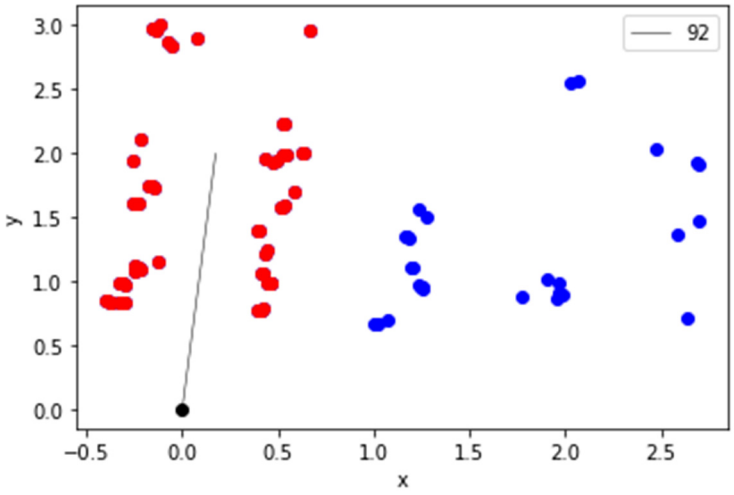


Figure 4: Generated path in Jupiter-notebook

By creating the first version of the algorithm outside of the ROS environment, a lot of time was saved, because its were more effective to run the different codes and solutions without setting up the Gazebo environment for each run.

2 Navigation in a virtual field

Task 1

The first task were made using the knowledge from prior and therefore the development of an functional code was influenced by the work that had been done before. The code were made up by four different functions, were one of the functions is the main functions. The three sup-functions is as follows:

- def Scan_Callback(msg): Processes the Lidar data
- def Func(data): Works with the data
- def Controller_Callback(erros): Error estimation

All of these functions main purpose were to process the data, that would later be use in the codes main function ”*while not rospy.is shutdown()”.*

Scan_callback(msg)

This is the first function which collect all the data point from the Laser scanner, and arranges then into a coordinate system with the XY coordinate of each located topic that are detected. Because the sick range is to wide, we narrow down its field of view, and thus having a more precise output. The output from the function is an global array that contains the coordinates/ viewpoint of the Jackal robot that would be used in the next function func(data).

func(data)

With the data form scan callback(msg) its now possible to detects the closest point from the LiDAR scan, extrapolates nearest points associated in the same row as the initial closest point, and outputs a path to follow which is shifted 0.375m from the row along the x-axis.

Controller_Callback(Erros)

To make sure that the robot wouldn’t make any hard turns, this functions were implemented to make sure its movement were under control. Its made out of an simple P-controller, therefore forces the robot to take a slower turn, and thus not making an radical turn.

Turning

In the beginning of the process, a lot of work were put into making a algorithm, that could determined what radius the robot should rotate when taking a turn and how it should drive into the rows again. This solution was quickly scrapped, because if some noise that was detected, resolution in the Jackal robot would change course. Therefore, every turn was eventually hardcoded to save time and resources.

Demo

To prepare us for the FRE competition, the code was tested on a small version of the ROS environment. This was done, to save time as well as to be able to run a lot of tests without our computer burning down. Our success rate for completing the task was relatively high, which was reflected during competition when we ended up coming in 7th place, out of 14 for task 1.

Task 2

Since task 1 and task 2 were so similar a lot of the the same code were reused, all the functions stayed untouched beside the main code. Since we needed to follow an predetermined route. The chances to the main code, were hard-coded, to determent which direction the Jackal robot should follow. Since the input either where ”R” or ”L” and the length of the turn, ”1” or ”2”. A simple if else function were hard-coded in the main function, and thus expanding the existing turning code from Task 1.

When we had to investigate how robust the code was, the testing were done like in Task 1, in a smaller version of the ROS environment. The parameters we could change in the code were mainly how hard the Jackal robot would turn, before it should depend on the LiDAR system again. During the competition we ended up getting in 9th place, because the Jackal robot took a turn to hard, an not being able to correct its path into the corn field again.

Limitations

Even though we wanted to work together as a group in the ROS environment, we only had the opportunity to borrow one computer from DTU, which we got handed out way too late, with the necessary technical specs to probably run the code, thus, the overall efficiency decreased. Since our own computers were not powerful enough to run the final implementation of the virtual environment. This was not a problem at the beginning of the project, but gradually it became more impossible to work with Gazebo, for each and every update of the environment that was released, since our computer could not handle the simulation. Even though we all had relatively powerful computers, they still didn’t live up to the requirement to have their own GPU, which was necessary during the development.

Conclusion of FRE

Although we didn’t compete in the remaining tasks, we managed to succeeds in completing the first two tasks, which taken into account is a small victory in relation to all the challenges that arose along the way. If we should do this again, then the FRE competition should have started earlier, to fit the academic calendar of DTU. Since all the exams of DTU took place 2 weeks prior the competition, while each group members had different schedules. Another element that has been a huge barrier have been the lack of computing power, none of the group members have had problems running heavy programs or simulations on our computers in our studies, before now. If FRE are going to held online again, then DTU should have more resources available for the participants in order to succes. From an educational point of view, the learning outcome was general high, because every group member contribute to the project, and have been force to learn new aspects of robot development.

TAFR – TAFR ROBOTICS

Urban Bobek [1], Janez Cimerman [1]

[1)] *Faculty of Electrical Engineering, University of Ljubljana, Slovenia*

1 Introduction

With the competition moving to a virtual environment a couple of challenges appeared. In the past we used the Gazebo simulator to experiment with different navigation algorithms, but never to the extent that was needed for the competition. Nevertheless, having some experience with Gazebo made setting up the simulation environment straight forward, especially due to the comprehensive instructions provided by the organisers.

We believe that the charm of this competition is seeing all the different robots that the teams make to solve the navigation challenges, but for this year we welcomed the organisers decision to have the teams compete using the same robot for the first two tasks. Having the same robot driving in the same environment meant that the only variable in the team's performance was their navigation algorithm.

2 Navigation in a virtual field

Because the competition was held in a simulation environment and we did not need to worry about the robot hardware, we decided to try a new approach in the navigation tasks. After some experiments we found out that a SLAM algorithm developed by Eurico Pedrosa (University of Aveiro) [1] worked surprisingly well in the simulation environment and that it was not too computationally expensive.

The SLAM algorithm computes a map of the environment and the location of the robot within it. In a structured environment the algorithm can provide localization with an accuracy of up to 5 cm. This number increases when the environment becomes unconstrained like the one in the competition. Nevertheless, when testing the algorithm on the provided simulation we decided that it was accurate enough.

The map that the SLAM algorithm creates is available as a ROS topic, where the information about the map origin is also provided. Using this information, we can calculate where on the map the robot is located. To plan where the robot should drive, we convert the map into an image on which we then apply image processing algorithms that we will explain in the following sections.

2.1 Task 1 – Basic Navigation

Solving the challenges of task 1 is crucial because they are the basis of all other tasks.

2.1.1 Navigation algorithm overview

In our approach to solve in-row navigation we used a combination of LIDAR data and the map that we get from the SLAM algorithm. For processing the image of the map, we used the OpenCV library. The laser scan from the LIDAR sensor is filtered

to omit scans that are further then 5m away from the robot before it is sent to the SLAM algorithm.

The map image is a binary image where most of the map is represented as white and obstacles are represented as black pixels. When a robot is in the middle of a row, in a perfect scenario we would not have any obstacles visible in the middle of the row and many on both sides. Because the leaves of the maize plants can reach in the rows and get picked up by the laser scanner, we get an image that looks similar to the right side of Figure 1.

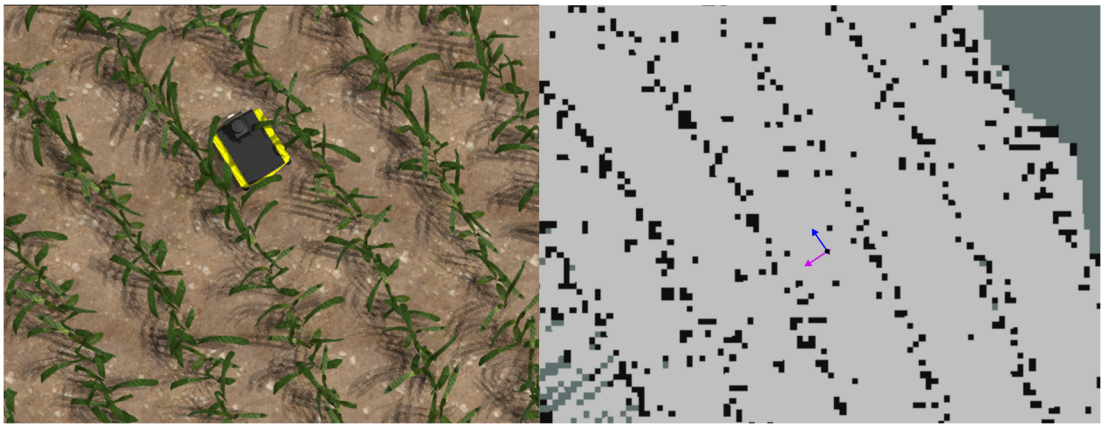


Figure 1: View from the Gazebo simulation (left) and the map and robot position from the SLAM algorithm (right)

Using the information from the map topic we can calculate where on the image the robot is located. Around this location we then create a rectangle of length *l* and width *w* that is oriented in the direction of the robot. We also create two rectangles that are moved perpendicularly to the robot for 0.35m to each side, which corresponds to half of the row width. We crop the part of the map image that is inside these rectangles and use them to evaluate the driving direction. You can see how the rectangles are placed in Figure 2. On the images cropped from the rectangles we calculate a score by summing the number of black pixel that we weight by their distance from the robot location.

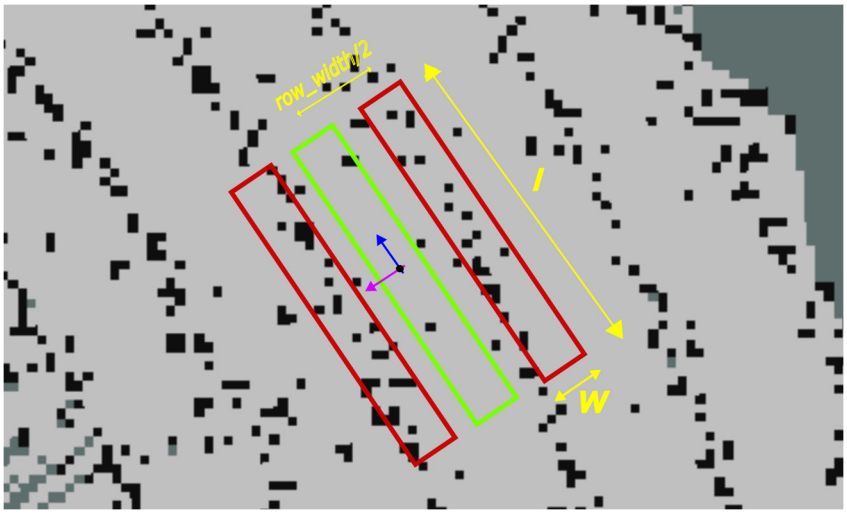


Figure 2: Centre and side rectangles that we position around the robot position on the map image

Our logic to determine the best direction for the robot to drive is as follows. The score that we get from the middle rectangle should be as low as possible because there should not be any plants and the value that we get from the side rectangles should be high. We incrementally rotate all three rectangles for a small angle around the robot origin up to 30° in both directions. For each small rotation we calculate a score, and we choose the angle at which the score is the smallest for the middle rectangle and the highest for the side rectangles. We denote these angles as ρ_{mid}, $\rho_L$and ρ_R for the middle, left and right rectangle respectively. The scores for the left and right rectangles we denote as L_{max} and R_{max}. Using these scores, we calculate weights as:

$$w_L = \frac{L_{max}}{L_{max} + R_{max}},$$

$$w_R = \frac{R_{max}}{L_{max} + R_{max}}$$

that we use to calculate a drive angle that is determined only by the side rectangles:

$$\rho_{side} = w_L\rho_L + w_R\rho_R.$$

The final drive angle is calculated by also considering the angle from the middle rectangle:

$$\rho = \rho_{side} + \rho_{mid} * |w_L - w_R|.$$

Calculating the drive angle like this considers the information from all three rectangles and emphasises the angles from the rectangles with better scores. The rotation of rectangles can be seen in Figure 3.

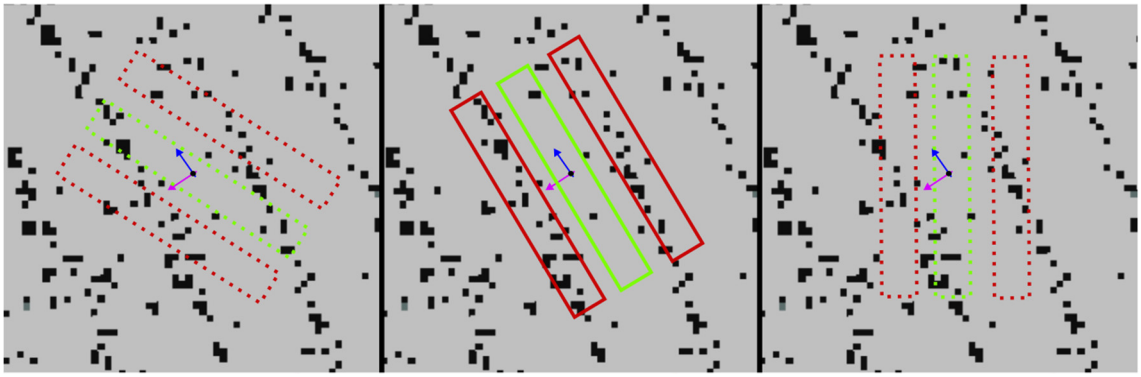


Figure 3: Three different rotations of the rectangles. At each rotation a score is calculated from an image cropped from each rectangle that is then used to determine the drive angle

Using this angle, we determine the position where the robot should drive next. We do this by calculating a velocity *v* and calculating a point on the map where the robot should drive to. In this point the previously described algorithm is repeated and a new angle is calculated. We repeat this process 5-7 times until we get a path where the robot should drive. This path is then published on a topic.

A different ROS node handles driving the robot. It is subscribed to the path topic and controls the robot velocity using a simple P controller. When the robot is aligned

with the desired driving angle it drives towards a point on the path. If it is not, it only rotates around its axis of rotation. Because of the simplicity of our controller, the robot drives in a jittery manner.

2.1.2 Results

When testing our navigation algorithm in the simulation environment, where the ground was flat, the robot could navigate the field with up to 2 m/s linear velocity. When testing it on the final version of the environment however, we noticed that the localisation of the SLAM algorithm began to be less accurate resulting in the robot consistently driving off the rows. Reducing the linear velocity to around 0.6 m/s improved the robot performance so that it could consistently drive within the rows.

In the final version of the simulation environment the ground was very uneven resulting in the robot tilting and loosing grip. Often the robot would be standing on only 3 or even 2 wheels which made turning the robot in the spot difficult and inaccurate. This was also the main reason the localisation was worse in the environment with uneven ground. Because our drive controller was strongly dependent on turning in the spot and because our algorithm was computationally expensive, our performance in in-row navigation was not as successful as we had hope for. In the end we were able to drive through approximately 4 rows in 3 minutes resulting in a 9th place.

2.2 Task 2 – Advanced Navigation

In the previous years we took a very straight forward approach to solving task 2. Once we detected that the robot reached the end of row, we simply turned 90° in the appropriate direction and started to drive along the maize field for a given amount of distance. The distance was calculated as the number of rows we had to change times the row width. We monitored the driven distance using odometry and once we travelled this distance we stopped and turned another 90° so that the robot was turned towards the field. Once we entered the field we changed back to the in-row navigation.

Because this year we were using a SLAM algorithm we thought we could use the generated map to plan the path when changing rows. When the robot is driving in the field we are constantly checking for the end of the row. We do that by checking the amount of lidar detections in front of the robot. When the end of the row is reached, we calculate the optimal entry point for the robot. Say that the robot must turn *n* rows to the left. On the map we construct a circle around the position where the robot detected the end of the row with a radius of *n*row_width*. The point where the robot should enter the field is somewhere on the left side of this circle but with the opposite orientation. We choose several different points that are spaced around the point that is on the circle and 90° to the left from the exit angle of the robot. On each point we perform the algorithm described in the previous section and we choose the pose that yields the smallest score. This idea is presented in Figure 4.

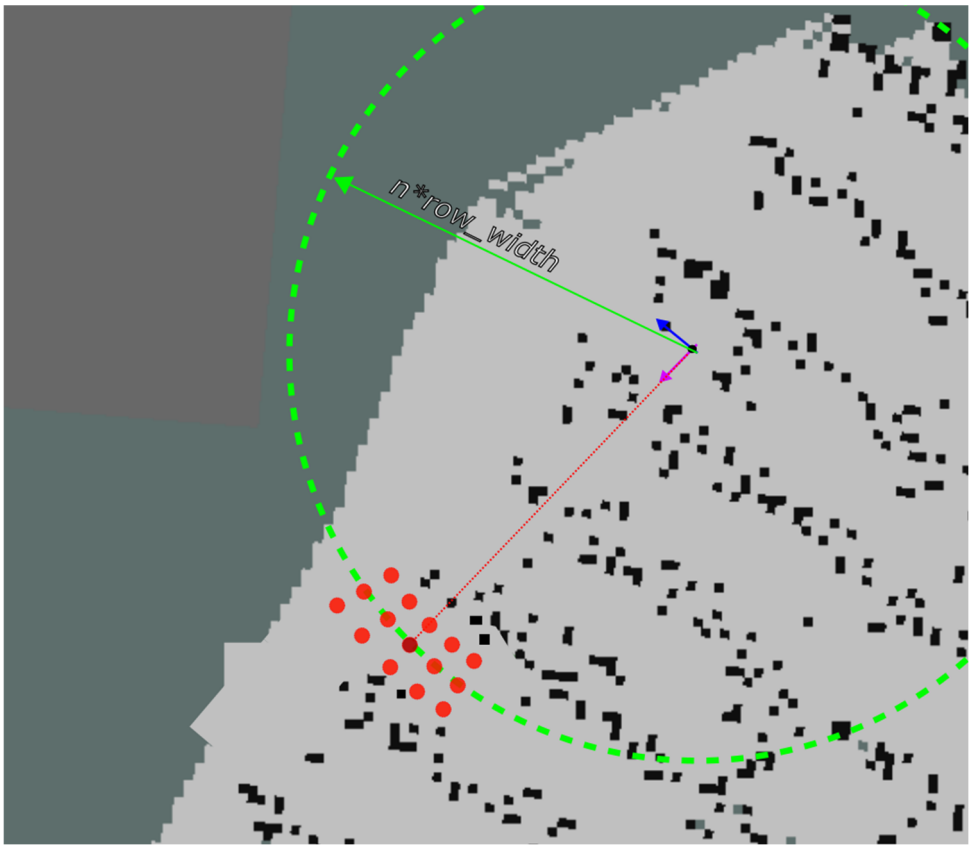


Figure 4: Planing the entry point for task 2. In this case the robot has to turn 3 rows to the left. The described algorithm is repeated on all red dots and the one with the best score is chosen as the entry point.

We than drive the robot towards this pose, but there is still a chance, that this point is position inside the row meaning that we would hit the plants. To solve this, we use the data from the laser scanner to drive away from any detections that are near the robot.

Because of this approach, there is a possibility that the robot would never reach the desired pose. That is why we stop the robot once it drives more than the *n*row_width* distance and turn it towards the field. After this we switch to the in-row navigation algorithm from the previous section which can also handle row entry.

So far in this section we have not mentioned how we handle missing plants in the rows that are present in task 2. That is because the algorithm already handles this situation in the way that the weights of the side rectangles are calculated. If there are plants missing on one side of the robot that side will yield a verry low score meaning it will be mostly ignored when determining the drive angle.

2.2.1 Results

The shortcomings of our approach persisted in task 2 resulting in our robot moving with very slow velocities. However, our algorithm proved to be robust against missing plants. In our test the previously described method seemed to work robustly even in edge cases such as when the robot had to drive in a row that was not yet mapped. In the end it was the slow movement of our robot that prevented us from being competitive with the best teams and resulted in ranking 5th in this task.

3 Applications in a virtual field

Unfortunately, we did not find the time to prepare for task 3 and 4 so we did not participate.

4 Conclusion

Simulations are already extensively used in the field of engineering, and we believe that they will be used even more in the future. Participating in this year's Field Robot Event gave us a great opportunity to improve our understanding and skills in using these simulation tools. Even though we did not get to see many different robots it was still very interesting seeing the same robot navigate the same field with many different approaches and strategies. The way the event was organized encouraged teams to communicate with each other and share ideas even more than in previous events. We also believe that a lot of what we learned can and will be implemented on real robots in the following competitions.

5 References

[1] E. Pedrosa, "GitHub," University of Aveiro, 2019. [Online]. Available: https://github.com/iris-ua/iris_lama_ros. [Accessed 1 May 2021].

Other issues

Last but not least, we would like to thank the immense support from our sponsors "Mestna občina Ljubljana" and "Epilog".

WEEDINSPECTOR – TEAM FLORIBOT

Moritz Böker [1], Ibrahim Can [1], Benedict Bauer [1,*]

[1)] *Mechatronic and Robotic, Heilbronn University of Applied Sciences, Germany*
[*)] *Instructor and Supervisor*

1 Introduction

On the virtual Field Robot Event 2021 a lot of things were new for Team FloriBot from Heilbronn University of Applied Sciences. For instance, the simulation environment was given with Gazebo but the team had always used a different simulation environment. Also, the whole competition environment with Docker was new for the team, too. For a small team of students who participate in such a competition in their spare time this is a big challenge. The advantages of these new challenges are the new knowledge and expertise.

2 Navigation in a virtual field

In term of navigation the virtual Field Robot Event is not too different from the conventional Field Robot Events in the past. There are the first two Tasks where in "Task 1 – Basic Navigation" the robot has to drive through rows of a given field row by row and in "Task 2 – Advanced Navigation" the robot has to drive through specific rows of a given field. To achieve this in our understanding, robots must be able to master three tasks when navigating through the given fields of the Field Robot Event:

1. Driving along rows
2. Turning at the end of rows
3. Navigating in headland

Tasks 1 and 2 only differ in the complexity of the navigation in the headlands.

A challenge of the for the Field Robot Event 2021 chosen robot Jackal from Clearpath Robotics was its skeed steered drive chain [1]. This challenge mainly affects turning at the end of rows.

2.1 Task 1 – Basic Navigation

As described befor Task 1 and Task 2 do not differ that much for the team of the Heilbronn University of Applied Sciences. For this reason, the difference is manly in the navigation in the headlands. This section describes the navigation in the field.

Driving along rows is done by a really simple algorithm which is only based on the data from the laser ranger. First the $\boldsymbol{ranges}$ of the laser ranger are converted into cartesian coordinates by using the appropriate $\boldsymbol{angles}$ as shown in the following equation.

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{ranges} \times \cos(\boldsymbol{angles}) \\ \boldsymbol{ranges} \times \sin(\boldsymbol{angles}) \end{pmatrix}$$

Two boxes are placed next to the robot. As shown in Figure 1 the in row boxes are on the left and on the right side of the robot. These boxes are used to filter relevant data. The dimension of the boxes can be easily adjusted and depends on the dimensions of the field.

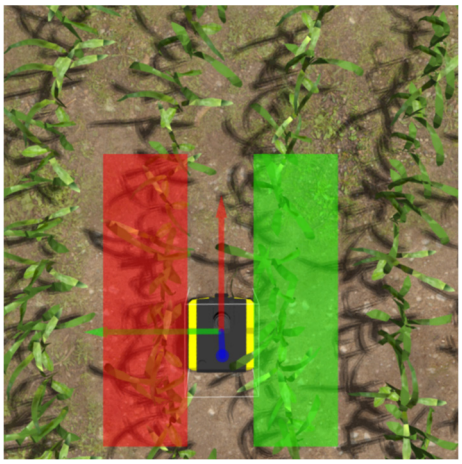


Figure 1: In row boxes

All y coordinates that are in the boxes ($y_{left\ i}$ and $y_{right\ i}$) are used to calculate the mean value of these for each box as shown in the following equations.

$$\bar{y}_{left} = \frac{1}{n_{left}} \sum_{i=0}^{n_{left}} y_{left\ i}$$

$$\bar{y}_{right} = \frac{1}{n_{right}} \sum_{i=0}^{n_{right}} y_{right\ i}$$

n_{left} and n_{right} represent the number of coordinates in the left and the right box.

With these two values, a distance for the robot to the center between two rows d can be calculated as follows.

$$d = \begin{cases} \bar{y}_{left} + \bar{y}_{right}, if\ \bar{y}_{left} \neq nan\ and\ \bar{y}_{right} \neq nan \\ \bar{y}_{left} - \frac{row_{width}}{2}, if\ \bar{y}_{right} \neq nan \\ \bar{y}_{right} + \frac{row_{width}}{2}, if\ \bar{y}_{left} \neq nan \\ 0, else \end{cases}$$

With row_{width} the distance between two rows.

There are four options to calculate d because it can happen that there are no rows on one or both sides and so the mean of the y coordinates can then not be calculated.

The distance of the robot to the center between two rows d is then used to controll the translational and rotational speed of the robot. Tests on the available hardware have shown that it is possible to achieve good results by setting up only a few parameters.

The end of rows is detectet by using threshold of the number of coordinates in the left and the right box. Turning to the next row is described in the folowing section.

As already mentioned, the big challenge of the virtual Field Robot Event 2021 for the Team FloriBot was the simulation environment and especially the competition environment with Docker. In our test environment we had some performance issues we did not recognize and so we had not found the right parameters for the competition environment.

2.2 Task 2 – Advanced Navigation

In this section the focus is on turning at the end of rows and the navigation in the headlands. For the navigation in the field the same algorithm, as described in Task 1 – Basic Navigation, is used.

At the end of the row the robot decides what to do next based on the given pattern for task 2. In task 1, a pattern is used that fulfills the required job for that specific task.

In case the pattern defines that the same row the robot is coming from has to be driven back the robot drives a little straight out of the rows then does a 180° turn on the spot and drives back into the row. To give respect to the skeed steered drive chain of the given robot an abort condition for the 180° on the spot turn had to be found. For that reason, the cartesian laser data are again filtered with a box. As shown in Figure 2 this time it is one wide box called center box that lies centered in front of the robot. The dimension of the box can be easily adjusted and depend on the dimensions of the field.

Figure 2: Center box

All y coordinates $y_{center\ i}$ that are in the box are used to calculate its mean value as shown in the following equation.

$$\bar{y}_{center} = \frac{1}{n_{center}} \sum_{i=0}^{n_{leftcenter}} y_{center\ i}$$

With n_{center} representing the number of coordinates in box.

While turning the mean value moves due to the field conditions from one side of the robot to the other side. The zero crossing of the y mean $\bar{y}_{center}$ is used to abort the turning.

As shown in Figure 3 the robot drives curves for all the other instructions from the pattern because that's faster than the alternative on spot turns.

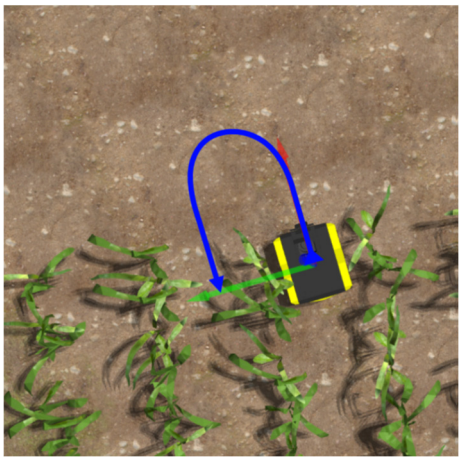


Figure 3: End row turn

To drive into the next row the robot should drive a semicircular curve with a radius of $row_{width}/2$. To give respect to the skeed steered drive chain and the fact, that the robot sometimes doesn´t have to drive directly into the next row the semicircular curve is devided into two parts. The abort condition for the first quarter circle is determined in a similar way as that for the turn on the spot. A box on the side of the robot to wich it has to turn to as shonw on Figure 4 fir a left turn is used to filter the cartasian laser ranger data.

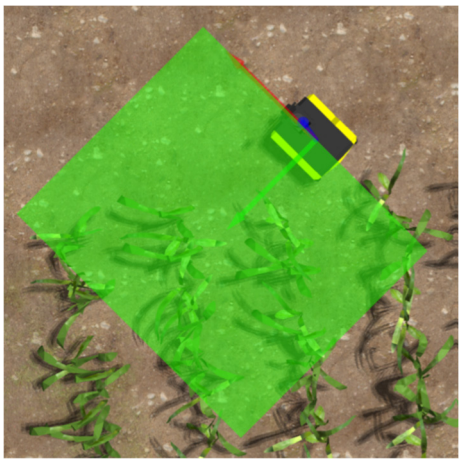


Figure 4: Headleand box for left turn

This time the mean of the x coordinates $x_{headleands\ i}$ that are in the box are calculated as shown in the following equation are used.

$$\bar{x}_{headleands} = \frac{1}{n_{headleands}} \sum_{i=0}^{n_{headleands}} x_{headleands\ i}$$

Due to the field conditions, the mean value moves from the back to the front during curving and the zero crossing is used for the abort condition to deteced the end of the quarter circle.

After the first quarter turn there are two possibilities, either the robot has to enter the next row directly, which is the standard for task 1, or the robot has to pass a certain number of rows.

If the robot has to drive directly into the next row a second quarter turn with the same radius as the first one is attached. For the abort condition to deteced the end of the second quarter circle into the rows the same one as for the 180° on spot turn could be used, the zero crossing of the y mean $\bar{y}_{center}$.

The x-mean $\bar{x}_{headleands}$ is proportional to the angle of the robot to the headlands and so it is in use to drive along the headlands as well as the minimum distance to the row ends.

To determine how many rows are passed while driving in headlands to reach a certain row another box is used. The row detection box is shown in Figure 5. As for all the other boxes the dimension of the row detection box can be easily adjusted and depends on the dimensions of the field.

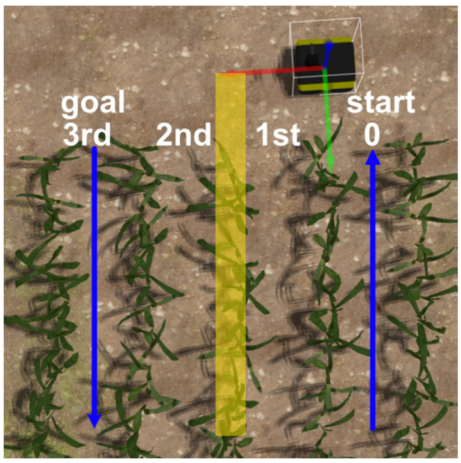


Figure 5: Row detection box

To determin if a row is passed or not a simple method with a threshold for the numbers of coordinates in the row detection box $n_{rowdetection}$ is used.

3 Applications in a virtual field

For Task 3 a feature-based detection and matching algorithm is on a slightly modified version of the jackal robot simulation.

3.1 Task 3 – Weed / object detection and mapping

The algorithm is based on comparing and analysing point matches between the reference image and the target image. If any part of the scene has matches greater than a given threshold value, that part of the image of the scene is selected and considered as the reference object there.

The features can be specific structures in the image, such as points, edges, or objects.

In order to detect the objects in the scene we had to give the databank enough reference images of the objects. It is important to extract as many visual features as possible so that the probability of matching the image and identifying it as an object is high.

Feature-based object detection can be divided into three main components. The feature detection, the feature description and finally the feature matching. [2]

Feature detectors are used to find the essential features of the given image. They are mainly classified into three types:

- corner-based detection, i.e., corner detectors
- blob-based detection, i.e., blob detectors
- region-based detection, i.e., region detectors.

These methods are used to find features such as corners, blob and regions from the reference image.

We used a detector called FAST. It uses an intensity-based method and detects corners. The advantage of this detector is that it is well suited for real-time applications due to its computational efficiency. [3]

Feature descriptors are used to describe the features that are detected by the detectors.

We chose the BRISK descriptor because it has a relatively high matching speed and low memory requirements. Thus, it can also be implemented and tested on weaker computers or virtual machines. [2]

The last component is the feature matching. It is used to establish matches between the reference image and the image from the scene.

The problem with this approach, however, was that not all objects offered sufficient features for a match. The beer can and beer bottle as well as the Coca Cola can could be identified relatively well since many matching points emerged from the logos and the letters. For the Pepsi can and the weed it was rather more difficult to achieve a match.

For the implementation Find-Object [4] is used.

4 Conclusion

Despite the less good performance of the team FloriBot at the virtual Field Robot Event 2021 all team members have gained very valuable experience and learned a lot. For that reason, the team hopes that a virtual contest will be an expansion of the future Field Robot Events or a small separate event maybe in the winter. But nothing is better than driving a real robot through real dirt on real fields, so team FloriBot hopes, that the Field Robot Event 2022 will have a real part.

5 References

[1] S. Rabiee and J. Biswas. "A Friction-Based Kinematic Model for Skid-Steer Wheeled Mobile Robots,". 2019 International Conference on Robotics and Automation (ICRA). 2019. pp. 8563-8569. doi: 10.1109/ICRA.2019.8794216.

[2] D. Tyagi. “Introduction To Feature Detection And Matching”. Medium. https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d. (accessed August 6, 2021)

[3] D. Tyagi.. “Introduction to FAST (Features from Accelerated Segment Test”. Medium. https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65. (accessed August 6, 2021)

[4] M. Labbé. “Find-Object”. introlab.github.io. http://introlab.github.io/find-object/. (accessed August 6, 2021)

JACKAL - WURKING

Rick van Essen [1], Christian Lamping [1], Bart van Marrewijk [2]

[1)] *Farm Technology Group, Wageningen University and Research, The Netherlands*
[2)] *Greenhouse Horticulture, Wageningen University and Research, The Netherlands*

1 Introduction

In contrary to previous editions of the Field Robot Event, the whole event was done in a simulated maize field with a simulated robot. The simulation was implemented in Gazebo. Therefore, all the robot software was made using the Robot Operating System (ROS). For task 1 and 2, a standard robot model was provided, in task 3 an adaptation to the standard robot was made by adding additional sensors.

Chapter 2 describes the waypoint and behaviour controller needed to drive the robot to given waypoints. The inrow navigation is described in section 2.1 and section 2.2 describes the headland navigation. Chapter 3 describes the specific virtual hardware setup for task 3, and in section 3.1 the corresponding methods are explained. We did not participate in task 4 and 5.

2 Navigation in a virtual field

For task 1 and 2, the standard Jackal robot was used. Lidar, wheel encoders and IMU were available for navigation. To let the robot drive, two controllers were implemented: (1) the low-level waypoint controller and (2) the high-level behaviour controller. The waypoint controller makes sure that the robot drives to a specific point in space and the behaviour controller controls the behaviour of the robot by providing waypoints to the waypoint controller.

Waypoint controller
The robot is controlled by target points consisting of an x and y value. When a new waypoint is provided, the waypoint is transformed to the odometry frame. A straight line between the current robot position and the transformed waypoint is drawn. This line should be followed by the robot to reach the target point. A PID controller is used to control the rotational velocity of the robot around the yaw axis, $\dot{\gamma}$, in order to follow the straight line (Hague & Tillett, 1996):

$$\dot{\gamma} = K_p \cdot d_{offset} + K_d \cdot \theta$$

$$K_d = -\zeta \cdot \sqrt{-4 \cdot K_p \cdot v_{forward}}$$

here d_{offset} is the perpendicular offset of the robot from the straight line, θ the angle between the robot and the line, $v_{forward}$ the forward velocity of the robot, ζ the damping coefficient ($\zeta = 0.95$) and K_p a parameter defining the response of the controller ($K_p = -1.3$).

Figure 1 visualises the straight line between the robot position and waypoint, and the path the robot is driving to follow the straight line.

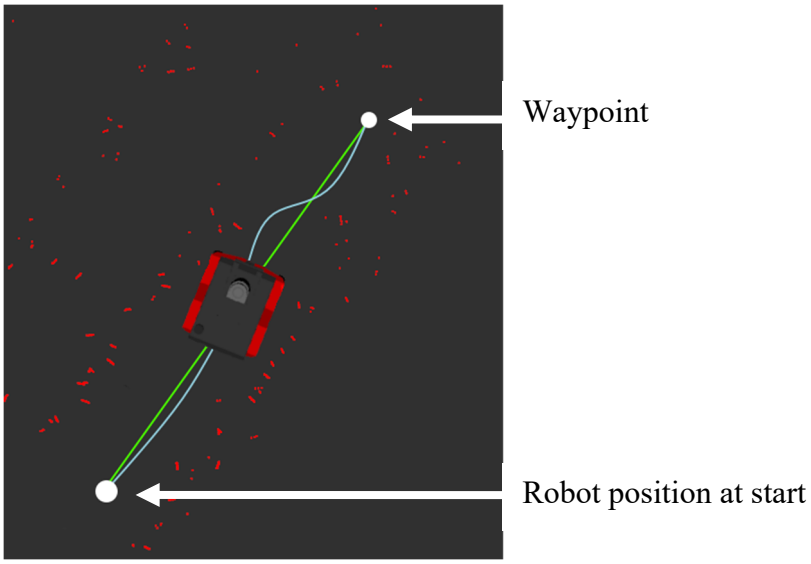


Figure 1. Working principle of the waypoint controller that moves the robot from its start position to a specified waypoint. The green line indicates the line the robot follows using a PID controller. The blue line is a visualisation of the path the robot drives to follow the line. The red dots indicate lidar measurements.

Behaviour controller

The behaviour of the robot was defined in 8 different states and implemented in FlexBE (Conner & Willis, 2017). Figure 2 shows the states and state transitions. During *initialisation*, based on the current task, the headland pattern is loaded. For task 1 and 3, the headland pattern is set to an infinite list of turning left and right. In task 2, the headland pattern provided by the organization is used. If there were no failures, the *drive into row* state drives the robot 0.5 meter straight ahead to make sure that the robot is in the row. The *inrow* state (see section 2.1) drives the robot through the row. At the end of the row, the *drive out row* state drives 0.5 meter straight ahead to make sure that the robot is not in the row anymore. The *iterator* state takes the next headland turn from the headland pattern. If there is no turn anymore, the task is finished. If there is a headland turn available, the turn direction is passed to the *turn out* state. This state turns the robot 90 degrees (using IMU) in the provided direction. The *headland* state (see section 2.2) uses the lidar to estimate the midpoint of the row that the robot should turn in using the number of rows to skip provided by the *iterator* state. After arriving at the midpoint, the *turn in* state turns 90 degrees using the same direction as the *turn out* state. After turning the robot, the *drive into row* state starts again.

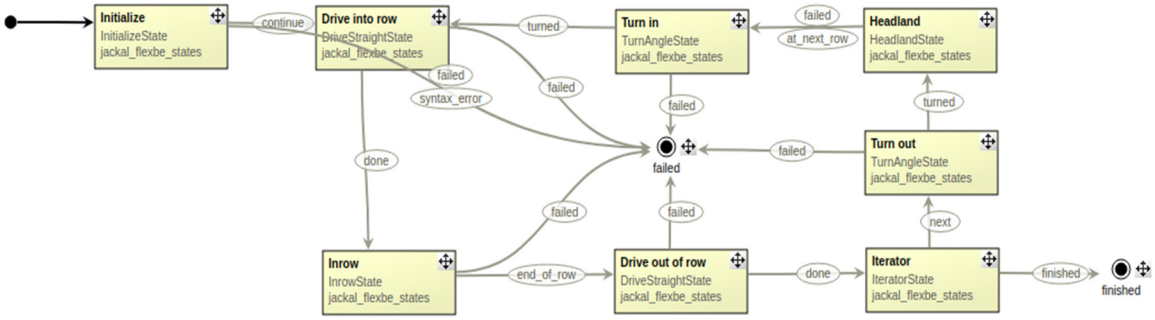


Figure 2. Overview of the implemented FlexBE statemachine with all different states and transitions.

2.1 Task 1 – Basic Navigation

The main focus for basic navigation was to estimate the position of the robot within the row based on lidar data. The data of the lidar is quite noise, making it challenging to accurately estimate the position and angle of the robot in the row. Furthermore, when the robot is rotated with respect to the row, it is hard to detect any row of plants. To overcome these difficulties two methods were tried: a neural network and a line fitting algorithm.

The first method used a neural network consisting a 1D convolution filter and two dense layers. The output of the network is the distance with respect to the middle of the row. This approach worked reasonable for the first virtual field consisting of simple cylinders, but when the field became more complex by adding virtual maize plants, the network was not able to estimate the position with a high accuracy. Therefore, a line fitting algorithm was implemented instead of the neural network.

The focus of the line fitting algorithm was to create a simple algorithm that was easy to understood and without too many parameters. The idea was that when you try 'all' positions of the robot, that the best solutions should have the highest number of points that could be assigned to a left or right row. Figure 3 shows an example of this best solution, with in blue points assigned to the left row and in green the points assigned to the right row. Points were considered to the left row if the x-position with respect to the robot was between -0.45 and -0.3 meter and assigned to the right row when the point was between 0.3 and 0.45 meter. All red points were not included in counting the number of detected points.

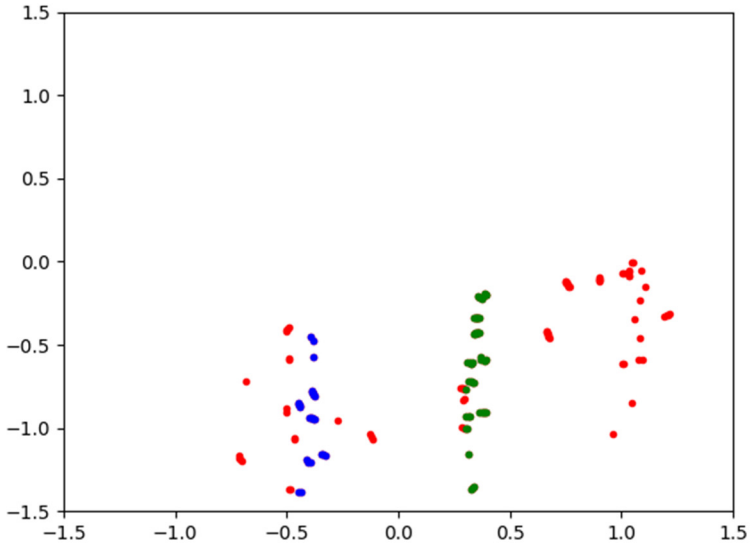


Figure 3. Assignment of lidar points to the left row (blue), the right row (green) and excluded points (red).

Since calculating all possible positions of the robot would be time-consuming, only 7 positions in the x direction and 15 angles were used. As a result, 105 positions were calculated for each lidar message. The position with the highest number of points assigned to the left and right row was assumed to be the actual position of the robot.

In pseudo-code:

```
function estimate_offset_and_angle(lidar_points):
    number_fitted_points = 0
    best_offset = 0
    best_angle = 0
    for offset in all offsets:
        for angle in all angles:
            transformed_points = transform lidar_points using offset and angle

            points_left_row = length(select all transformed_points between -0.45 and -0.3)
            point_right_row = length(select all transformed_points between 0.3 and 0.45)

            if points_left_row + points_right_row >= number_fitted_points:
                best_offset = offset
                best_angle = angle
                number_fitted_points = points_left_row + points_right row

   return best_offset, best_angle
```

Code 1. Pseudo-code indicating how the inrow navigation works.

The best target point, (x_{wp}, y_{wp}), was calculated based on the best fitted row angle (relative to the robot pose), θ, and offset from the middle of the row, y_{offset}:

$$x_{wp} = 0.5$$
$$y_{wp} = \tan(\theta) \cdot (x_{wp} + y'_{offset})$$

where $y'_{offset} = \frac{y_{offset}}{\sin(\theta)}$ is the projected offset of the robot from the middle of the row as visualised in Figure 4.

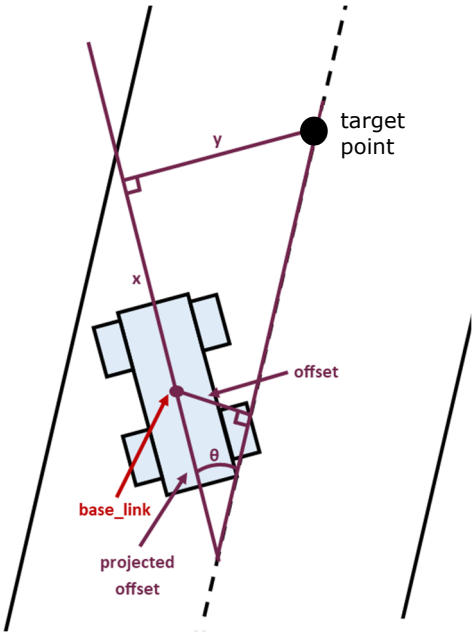


Figure 4. Schematic drawing of target point calculation using estimated offset and angle.

Despite the simple method, the inrow navigation worked good. In between the rows, the controller was able to steer the robot in the right direction. Only at the start of the row, the robot did drive through some plants. However, this problem was caused by the *drive into row* state, which drives the robot straight into the row to make sure the inrow navigation starts in a row (this distance was set too large).

2.2 Task 2 – Advanced Navigation

The main challenge in the advanced navigation task is to detect and count the maize rows at the headland. Using this information, the robot can skip specific rows and make sure that it is in the correct position to enter a specific row. The algorithm to detect and count the maize rows consisted of four consecutive steps: (1) detection of accumulations, (2) line recognition, (3) duplicate removal and (4) calculation of the next target point.

Detection of accumulations

As a first step, all lidar point accumulations (see Figure 5) were combined to a single point to reduce complexity of the data. For that, all points with a distance smaller than 10 cm to each other were summarized to one point by calculating the mean position of these points.

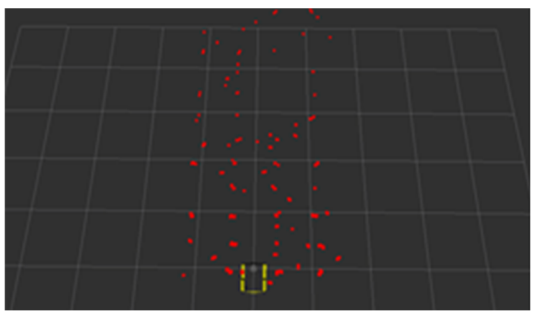


Figure 5. Lidar point accumulations (red dots).

Line recognition
In the second step, all summarized points were combined with each other and the line equation between the two points were calculated. Then, the distances of all remaining points to this line were checked. If a line had more than four other points in a distance of less than 20 cm, it was stored as a candidate line. The other lines were rejected. Furthermore, an angle range was defined to avoid that lines perpendicular to the actual maize rows were detected.

Duplicate removal
Step two resulted in many detected line points that were close together, describing the same real maize line. Therefore, linear regression was performed to all points of the candidate lines to obtain straight lines that can be compared. Then, lines with a distance of less than 20 cm were summarized to one line (see Figure 6).

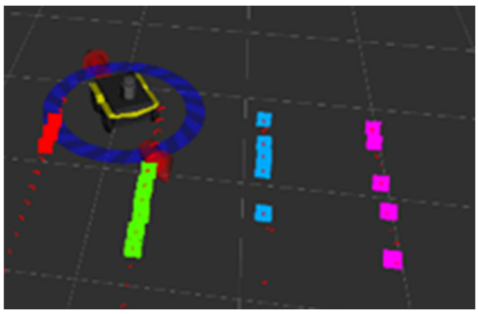


Figure 6. Result of linear regression to obtain straight lines.

Calculation of next target point

All detected lines were sorted by their distance from the robot. Then, the next target was calculated dependent on the number of rows, the robot was supposed to skip. The midpoint between the beginning of the two lines defining the target row was defined as next target.

This method worked well to detect and count lines in the lidar scan points. Weaknesses were only observed when maize plants were occluded from other plants and therefore the row could not be detected by the lidar scan. Additionally, the calculation of line equations for all detected accumulations was time-consuming. To address this challenge, the field of view as well as the criteria to summarize scan points were modified in order to reduce the number of accumulation points. However, there might be more potential for an improved efficiency if the number of for-loops during calculation could be reduced.

3 Applications in a virtual field

In task 3, an adapted version of the Jackal robot was used. Figure 7 shows the adapted version of the robot. Two cameras were added: a camera to detect the QR codes on the pillars on the headland and a camera to detect weed and litter on the soil. The QR camera is mounted on a joint that can rotate, in order to detect the QR code without moving the robot. The detection camera was mounted with an angle of 40 degrees.

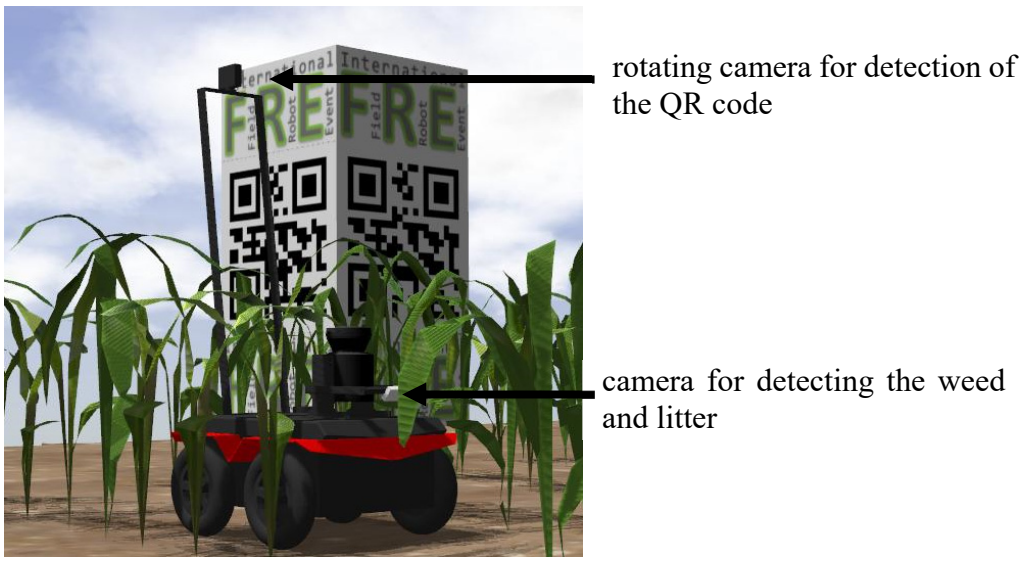


Figure 7. Overview of the adapted Jackal robot with the two added cameras.

3.1 Task 3 – Weed / object detection and mapping

At the start of the task, the pillar pose is detected using the lidar. First the point cloud was filtered using the DBSCAN algorithm (Ester *et al.*, 1996). Then, a line was fitted through the remaining points. The pose of the pillar was calculated using the average of all points within a certain distance to the line. Using this pose, a static transformation between the robots odometry frame and the pillar is made. The name of the pillar (a or b) is read from the QR code by rotating the QR camera until it found the text. The transformation between the pillar and the origin of the field is broadcasted using position of the pillar provided by the organisation.

The detection of weeds and litter was done using the Ultralytics implementation of YOLOv5 (Ultralytics, 2021).To reduce computational time, the small version of the network was used (yolov5s). The network was trained on 320 images of the virtual field, from which 250 images contained weed and / or litter. Training was done for 20 epochs using all default parameters, resulting in a F1-score of 0.9. By optimizing the parameters, better results could be obtained, but due to time limits, the default parameters were used. Figure 8 shows an example detection of litter in the virtual field. By assuming a constant distance of the camera to the soil (a flat field), the centre of the detection was converted to the robot coordinate frame and converted to the field origin frame by making use of the robots odometry. When there was no object of the same class (weed / litter) mapped within an Euclidian distance of 0.35 meters, the detection was mapped.

Figure 8. Detections of litter using YOLOv5 with confidence score indicated.

Unfortunately, during the event the robot did not work due to a few missing dependencies in our Docker file. On our own system (using a small world), we were able to detect most litter (4 out of 5) and half of the weeds (2 out of 4). To avoid double detections (from litter and weeds in the next row), we neglected the litter and weeds detections close to the plants. This resulted in missing a few litter and weeds. Figure 9 shows the mapping of the litter and weeds. There is a small offset between the predicted and ground truth position of the weed and. The general distribution in the field, however, seems to be accurate. The more distance the robot has travelled, the larger the offset due to the accumulating error in the odometry. By scanning the pillars at each headland and updating the field origin with respect to the odometry frame, this error could partly be resolved.

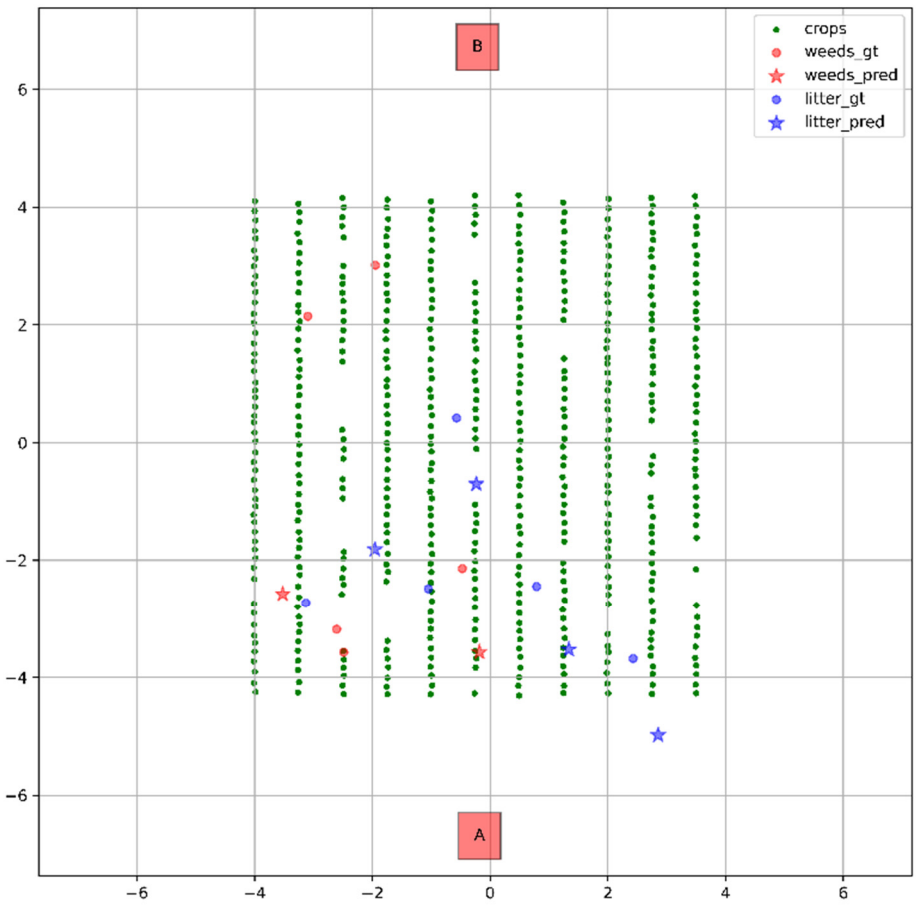


Figure 9. Mapping results on our own system. Note that all predictions have a small positive offset in the x-direction and a negative offset in the y-direction.

4 Conclusion

Our robot software was able to drive through the rows and make turns into next rows on the headland. The inrow navigation worked almost perfectly, however, we did drive too far into the row before starting the inrow navigation. The headland navigation did also work good, but it was computationally heavy. This caused a few seconds of delay on each headland. Detection of weed and litter did work good on our own system, but did not work during the event due to an error in our Docker file. On our own system, we were able to map most of the weeds and litter, but weed and litter close to the plants were missed.

The virtual field made it possible to focus fully on the algorithm and made comparison between different teams possible because all teams used the same robot for the first two tasks. This saved a lot of time for writing the robot model and made it possible to improve basic tasks like navigation. Another advantage was testing, testing a virtual robot takes far less time than testing a real robot. Despite this it was still a challenge to get a good inrow and headland navigation due to the (simulated) noisy sensor readings. Making corners on the headland however, was less realistic. In real life this would have been more difficult due to wheel slip. The biggest challenge of the virtual environment was the computational requirement because the computer has to process both simulation and robot software. This resulted in a really slow simulation most of

the time. However, the software developed this year can be used as a good basis for a (hopefully real-world) event next year.

5 References

Conner, D. C., & Willis, J. (2017). Flexible Navigation: *Finite state machine-based integrated navigation and control for ROS enabled robots.* Paper presented at the SoutheastCon 2017.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise.* Paper presented at the Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon.

Hague, T., & Tillett, N. D. (1996). Navigation and control of an autonomous horticultural robot. Mechatronics, 6(2), 165-180. doi:10.1016/0957-4158(95)00070-4

Ultralytics. (2021). yolov5. Retrieved from https://github.com/ultralytics/yolov5