www.fieldrobot.com/event

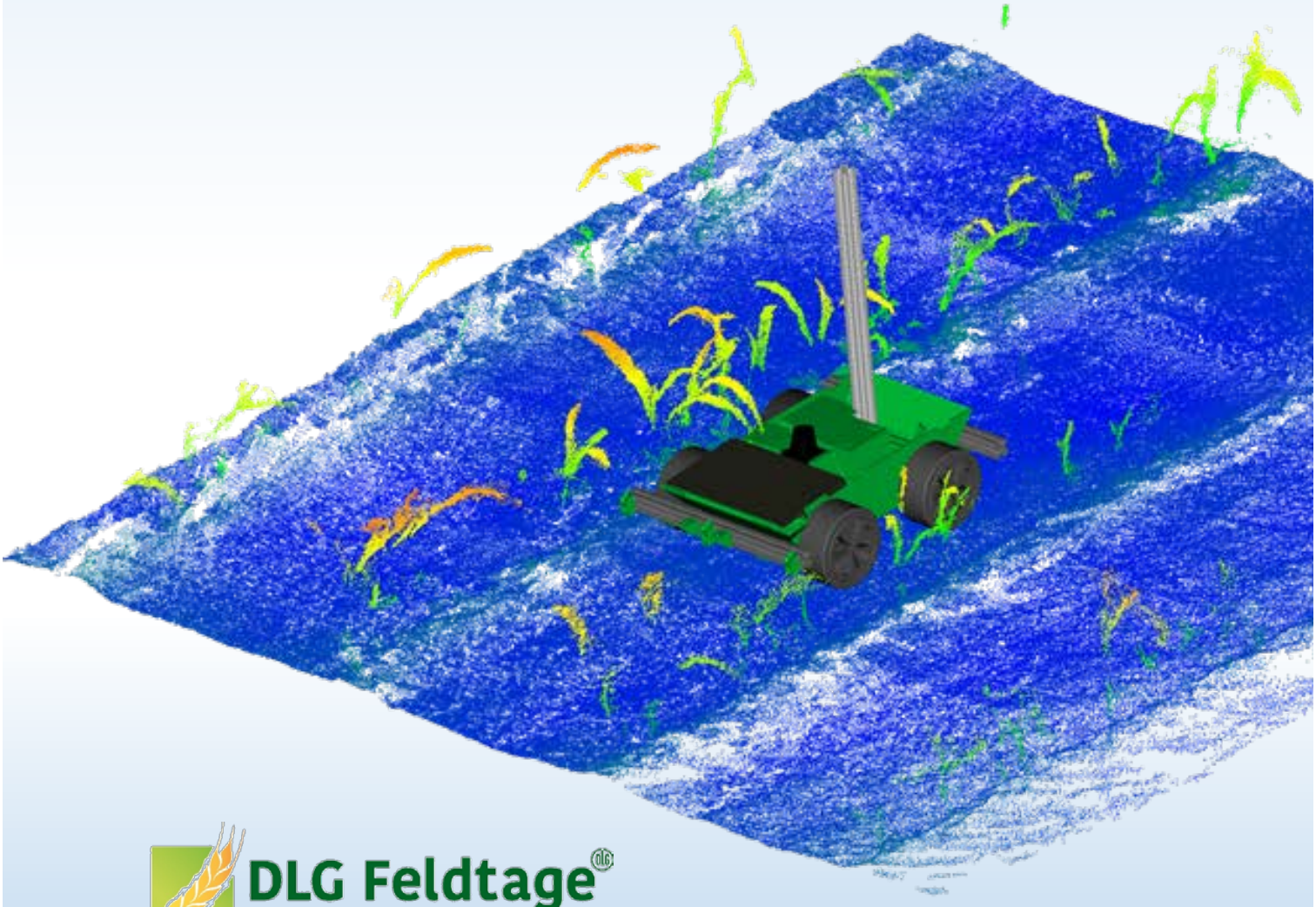
200 YEARS UNIVERSITY OF HOHENHEIM

Field Robot Event

Contest + Design + Demo + Talks

16th edition

Proceedings 2018



DLG Feldtage
Meet the crop professionals

Proceedings of the 16th Field Robot Event 2018

Bernburg-Strenzfeld, Germany
June 12th – 14th, 2018

Conducted in conjunction with the DLG-Feldtage / DLG Field Days

Editors:
M.Sc. Helga Floto
Prof. Dr. Hans W. Griepentrog

Date: February 2019

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

Contact: Prof. Dr. Hans W. Griepentrog

Phone: +49-(0)711-459-24550

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

Index

Sponsors


DLG


CLAAS
Stiftung


KRONE
THE POWER OF GREEN


FENDT


AMAZONE


JOHN DEERE


LINAK
WE IMPROVE YOUR LIFE


Harmonic
Drive AG


EurAgEng


Hochschule Anhalt
Anhalt University of Applied Sciences


rexroth
A Bosch Company


HORSCH
Landwirtschaft aus Leidenschaft

Field Robot Event 2018 - Task Description

Together with the DLG-Feldtage, 12th – 14th June 2018 Bernburg-Strenzfeld, Germany

Remark: The organizers 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.

0. Introduction

The organizers expect that a general agreement between all participating teams is 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 technologies. The teams should even provide support to each other with all fairness.

Any observed or suspected cheating should be made public immediately.

The jury members are obliged to act as neutrals, especially when having connections to a participating team. All relevant communication will be in English. For pleasing national spectators, the contest moderation could partly switch to a national language.

In 2018 five tasks will be prepared to challenge different abilities of the robots in terms of sensing, navigation and actuation: Basic Navigation, Advanced Navigation, Sensing, Weeding Control and Free Style (option).

If teams come with more than one machine the scoring, ranking and awarding will always be machine related and not team related.

All participating teams must contribute to the event proceedings with an article describing the machine in more details and perhaps their ideas behind it or development strategies in general.

During the machine runs for each task no team members are allowed to be in the inner contest area where the maize plants are and close to the robot during the performance. If the robot performance fails, it has to be stopped from outside with a remote switch. To enter the inner contest area is only allowed after (!) the robot has stopped. The control switch activating team member then can go to the machine and manually correct it. When the team member has left the inner contest area only then the robot is allowed to continue its operation. This procedure shall promote the autonomous mode during the contest and make the performance more attractive to spectators.

0.1. General rules

The use of a GNSS receiver is not allowed except for the Free Style in Task 5[1]. The focus for the other tasks shall be on relative positioning and sensor based behaviors.

Crop plants

[1] If you wish to use a GNSS, you must bring your own.

The crop plant in task 1 to 4 is maize (corn) or Zea Mays2. The maize plants will have a height of approximately 20 - 40 cm. The general appearance of the crop plants is location specific as well as yearly specific.

Damaged plants

A damaged plant is a maize plant that is permanently bent, broken or uprooted. The decision that a maize plant is damaged by a machine or not would be made by the jury members.

Parc fermé

During the contests, all robots have to wait in the parc fermé and no more machine modification to change the machine performance is allowed with regard to fairness. All PC connections (wired and wireless) have to be removed or switched off and an activation of a battery saving mode is recommended. This shall avoid having an advantage not being the first robot to conduct the task. The starting order will be random. When a robot will move to the starting point, the next robot will already be asked by the parc fermé officer to prepare for starting.

Navigation

The drive paths of the robots shall be between the crop rows and not above rows. Large robots or robots which probably partly damage the field or plants will always start after the other robots, including the second chance starting robots. However, damaged plants will be replaced by spare ones, to always ensure the same operation conditions for each run.

0.2. General requirements for all robots

Autonomous mode

All robots must act autonomously in all tasks, including the freestyle. Driving by any remote controller during the task is not allowed at any time. This includes steering, motion and all features that produce movement or action at the machine. Stopping and starting function for manual corrections of the machine is the only exception.

During start, the robot is placed at the beginning of the first row. The starting line is marked with a white cross line. Any part of the robot must not exceed the white line in the start. For signaling the start and end of a task there will be a clear acoustic signal. 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, but it must - after a basic repair - as soon as possible brought back into the parc fermé. If the robot fails twice, the robot will be excluded from the task list.

Start & Stop Controller

All robots must be equipped with and connected to one wireless remote START/STOP controller. Additional remote displays are allowed but without user interaction, e.g. laptop.

Preferably, the remote controller is a device with two buttons clearly marked START and STOP. Alternatively, the coding may be done with clear green and red colours.

It is allowed to use a rocker switch with ON/OFF position with hold, if the ON and OFF are clearly marked with text in the remote controller.

Any button of the remote controller may not be touched for more than one second at a time. In other words, a button, which has to be pressed all the time, is not allowed.

The remote controller may contain other buttons or controls than the required/allowed START/STOP inputs, but no other button may be used at any time during any task.

Before the start of any task, the remote controller must be placed on the table that is located at the edge of the field. One member of the team may touch the START and STOP inputs of the remote controller. The possible remote display must be placed on the same table too.

The remote controller must be presented to the Jury members before the run. A jury member will watch the use of the START/STOP remote controller during the task execution.

In each task, the robot must be started by using the remote controller START input, not pressing any buttons on the robot itself.

During any task, while the robot is stopped in the field by using the remote controller, it is allowed to use any buttons of the robot itself, e.g. to change the state of the navigation system.

While the robot is STOPPED and one team member is allowed to be in the field, besides rotating the robot, the team member is allowed to touch the buttons and other input devices mounted on the robot. Other remote controllers besides START/STOP controller are strictly prohibited to be used at any time.

Implementation note: If using Logitech Cordless Gamepad or equivalent as a remote controller, the recommended practice is to paint/tape one of the push button 1 green and push button 2 red, to mark START and STOP features.

Manual correction of robot

One team member is allowed to enter the field after the same (!) team member has pressed the STOP button of the remote controller and the robot has completely stopped (no motion). It is recommended to install some indicator onto the robot to see that the robot is in STOP mode before entering the field in order to avoid disqualification.

The START/STOP operator is also responsible for the eventually manual robot corrections. Due to the fact that it can be difficult for him/her to monitor the robot's behavior from a large distance, another team member can be inside the 2 m area between a red textile tape and the crop plant area (see picture 1 and 2 at the end of this document). This second team member could give instructions to the operator, but this supporting person is only an observer and is not allowed in any case to enter the crop plant area or interact with the robot.

After leaving the remote control on the table, the operator is allowed to rotate - not to move - the robot in the field. The only exception for moving is within the row, where the robot may need to get back to the path if a wheel or track of the robot has collided stem of maize plant, to avoid further damage of plants. Carrying the robot is only allowed after significant navigation errors in order to bring it back (!) to the last correct position and orientation.

In the headland, only rotating to give the robot a new orientation is allowed, no moving or even carrying is allowed at all.

0.3. Awards

The performance of the competing robots will be assessed by an independent expert jury committee. Beside measured or counted performance parameters, also creativity and originality, especially in task 4 and task 5 (freestyle), will be evaluated. There will be an award for the first three ranks of each task. The basic navigation (1), advanced navigation (2), sensing (3), and weeding (4) together will yield the overall competition winner. 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.

Participating teams result in at least 1 point, not participating teams result 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 all four tasks (1, 2, 3 and 4) will be ranked higher.

1. Task “Basic navigation” (1)

1.1. General description

For this task, the robots are navigating autonomously. Within three minutes, the robot has to navigate through long curved rows of maize plants (picture 1 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. There will be no plants missing in the rows. This task is all about accuracy, smoothness and speed of the navigation operation between the rows.

At the beginning of the match it will be told whether starting is on the left side of the field (first turn is right) or on the right side (first turn is left). This is not a choice of the team but of the officials. Therefore, the robots should be able to perform for both options. A headland width of 2 meters free of obstacles (bare soil) will be available for turning.

1.2. Field Conditions

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 push (max 25 mm) out from

the ground, both are installed. In other words, abilities as defined by machine ground clearance and to climb over small obstacles are required.

A red 50 mm wide textile tape is laid in the field 2 m from the plants.

1.3. Rules for robots

For starting, the robot is placed at the beginning of the first row without exceeding the white line.

If the robot is about to deviate out from the path and hit maize plants, the team member with the remote controller must press STOP button immediately. The STOP button must be pressed before the robot damages stems of the maize plants. The team is responsible to monitor the behavior of the robot and to use the STOP button when necessary.

1.4. Assessment

The distance travelled in 3 minutes 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:

Final distance = corrected distance * 3 minutes / measured time.

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 1 meter per plant.

The task completing teams will be ranked by the results of resulting total distance values. The best 3 teams will be rewarded. This task 1, together with tasks 2, 3 and 4, contributes to the overall contest winner 2018. Points for the overall winner will be given as described under chapter 0.3 Awards.

2. Task “Advanced navigation” (2)

2.1. 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 approach these field conditions in the second task.

The rules for entering the field, moving the robot, using remote controller etc. 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 3 minutes while navigating between straight rows of maize plants, but the robots have to follow a certain predefined path pattern across the field (picture 2 at the end of this text). Additionally, at some locations, plants will be missing (gaps) at either one or both sides with a maximum length of 1 meter. There will be no gaps in row entries.

The robot must drive the paths in given order. 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.

The code of the path pattern is made available to the competitors 15 minutes before putting all robots into the parc fermé. Therefore, the teams will not get the opportunity to test it in the contest field.

2.2. Field conditions

Random stones are placed along the path, to represent realistic field scenario where the robot should cope with holes etc. The stones are not exceeding the level of 35 mm from the average ground level in the neighborhood. The stones may be pebbles (diameter <35 mm) laid in the ground and large rocks that push (max 35 mm) out from the ground, both are installed. In other words, the robot must have ground clearance of this amplitude at minimum, and the robot must be able to climb over obstacles of max 35 mm high. No maize plants are intentionally missing at the end of the rows. However, due to circumstances of previous runs by other robots, it is possible that some plants at the end of the rows are damaged. The ends of the rows may not be in the same line, the maximum angle in the headland is ±15 degrees.

No large obstacles in the field and all rows are equally passable. A red 50 mm wide textile tape is laid in the field 2 m from the plants.

2.3. Assessment

The distance travelled in 3 minutes 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:

Final distance = corrected distance * 3 minutes / measured time.

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 1 meter per plant.

The task completing teams will be ranked by the results of resulting total distance values. The best 3 teams will be rewarded. This task 2, together with tasks 1, 3 and 4, contributes to the overall contest winner 2018. Points for the overall winner will be given as described under chapter 0.3 Awards.

Picture 2 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!

3. Task “Selective Sensing” (3)

3.1. General description

For this task, the robots are navigating autonomously. The robots shall detect weed patches represented by red and blue golf tees. You can find further details regarding the tees at the end of this document (Appendix B). Task 3 is conducted on the area used in task 2 with straight rows. Nevertheless, simple row navigation is required and the robot has to turn on the headland and return in the adjacent row. There will be nine (9) weed patches in total with three patches on each of the first three between row spacing. The end of the third row spacing is the finish line for this task 3.

The rules for entering the field, moving the robot, using remote controller etc. are the same as in task 1 and task 2.

3.2. Field conditions

The weeds are objects represented by blue and red golf tees distributed between the rows in the soil. The weeds are placed as weed patches in a squared area of 25 cm × 25 cm. The total number of weeds is 10 per weed patch but there are different combinations regarding the number of the red and the blue tees. The tees are in the soil with only the head or upper part visible (ca. 1 to 2 cm). The weed patches are located in a centered band of 25 cm width between the rows. Robots may drive across or over them without a penalty. No weeds are located within the rows and on the headlands. A possible example is illustrated in picture 3.

3.3. Rules for robots

Each robot has only one attempt. The maximum available time for the run is 3 minutes. The robot should detect the weed patches. The detection of the weed patch with a higher number of red tees shall be indicated by one (!) loud acoustic signal while the detection of the weed patch with a higher number of blue tees must be confirmed by two (!) consecutive and loud acoustic signals. The length of each acoustic signal may not be longer than 1 second.

3.4. Assessment

The jury registers and assesses the signaling as follows:

- True positives (correct signals, correct detection) + / plus 6 points,
- False positives (wrong signals, but weed detection) + / plus 2 point
- False negatives

(No signals with weeds or signals with no weeds) - / negative 2 points.

Crop plant damage by the robot will result in a penalty of 1 point 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. This task

3, together with tasks 1, 2 and 4, contributes to the overall contest winner 2018. Points for the overall winner will be given as described in chapter 0.3 Awards.

4. Task "Soil-engaged weeding" (4)

4.1. General description

For this task, the robots are navigating autonomously. The robots shall remove weeds from three (3) weed patches represented by red golf tees (Appendix B). Probably a soil engaged active tool is needed to succeed in this task. In order to minimize the used energy the tool should be active only on the weed spots. Furthermore, the amount of soil moved away from the spots shall also be minimized. Task 4 is conducted on the area used in task 2 with straight rows. Nevertheless, no specific path sequence will be given as in task 2 and the robot has to turn on the headland and return in the adjacent row. The three weed patches will be placed one on each of the three first row spacing. Thus, the end of the third row spacing is the finish line for this task.

The rules for entering the field, moving the robot, using remote controller etc. are the same as in task 1, 2 and 3.

4.2. Field conditions

The weed patches are objects represented by red golf tees distributed between (!) the rows in the soil like in task 3. The weeds are placed as weed patches in a squared area 25 cm × 25 cm (10 red tees per weed patch). Only the upper part of the tees is visible (ca. 1 to 2 cm). The weed patches are located in a centered band of 25 cm width between the rows. Robots may drive across or over them without a penalty. No weeds are located within rows and on headlands. A possible example is illustrated in picture 4.

4.3. Rules for robots

Each robot has only one attempt. The maximum available time for the run is 5 minutes. The robot shall remove the red tees from the weed patch. Each tee removed from the weed patch counts, each tee moved to the headland outside the crop area counts more and each tee collected on the robot and transferred to the end of the third row spacing counts most. Tillage outside the weed patches will be punished.

4.4. Assessment

The Jury registers and assesses the number of weeds (tees) where they are remaining after the run:

- Tees that are stored and transferred to the finish line + / plus 4 points/tee
- Tees delivered to the headlands + / plus 2 points/tee
- Tees cleared and removed from weed patches + / plus 1 point/tee.

Tilled travelled distance between the rows and outside the patches will be punished with 1 negative point per meter even in an intermittently mode. A tolerance of 10 cm for tillage before and after the weed patches will not be punished. The jury will assess the cleanliness

of the tees, means the amount of soil picked up with the tees and hence transported away from the spots (0 to 10 points penalty).

Crop plant damage by the robot will result in a penalty of 1 point 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. This task 4, together with tasks 1, 2 and 3, contributes to the overall contest winner 2018. Points for the overall winner will be given as described in chapter 0.3 Awards.

5. Task "Freestyle" (5)

5.1. Description

Teams are invited to let their robots perform a freestyle operation. Creativity and fun are required for this task as well as an application-oriented performance. One team member has to present the idea, the realization and perhaps to comment the robot's performance to the jury and the audience. 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.

5.2. Assessment

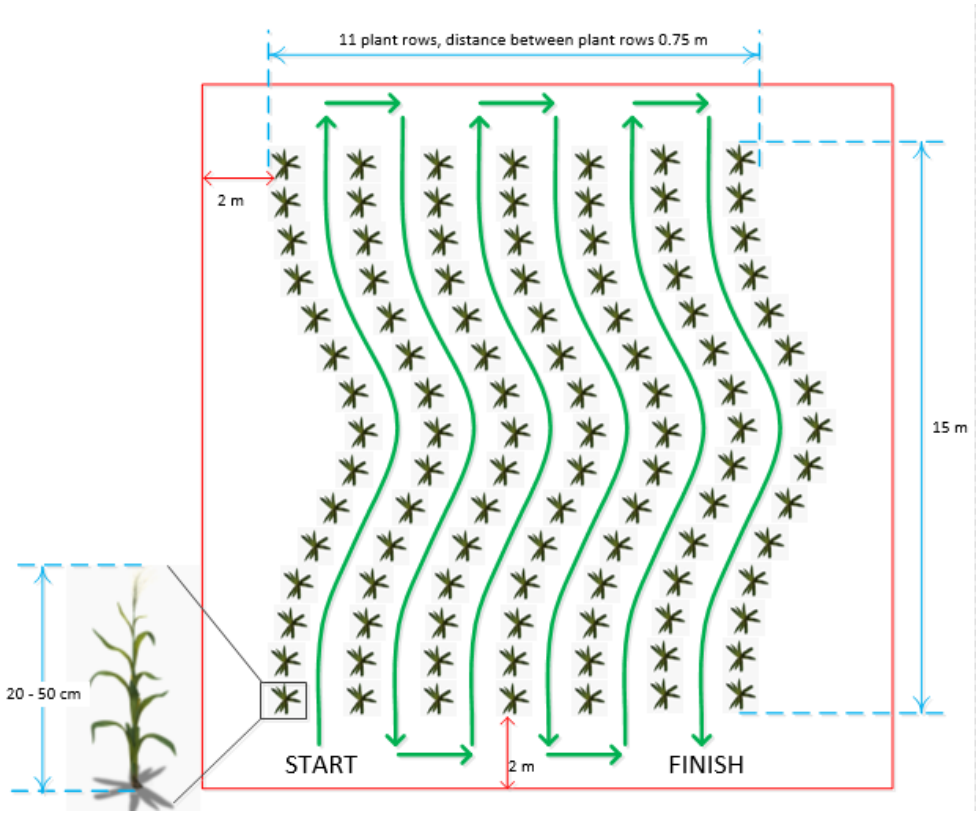
The jury will assess the (i) agronomic idea, the (ii) technical complexity and the (iii) robot performance by giving points from 0 (insufficient) to 10 (excellent) for each.

The total points will be calculated using the following formula:

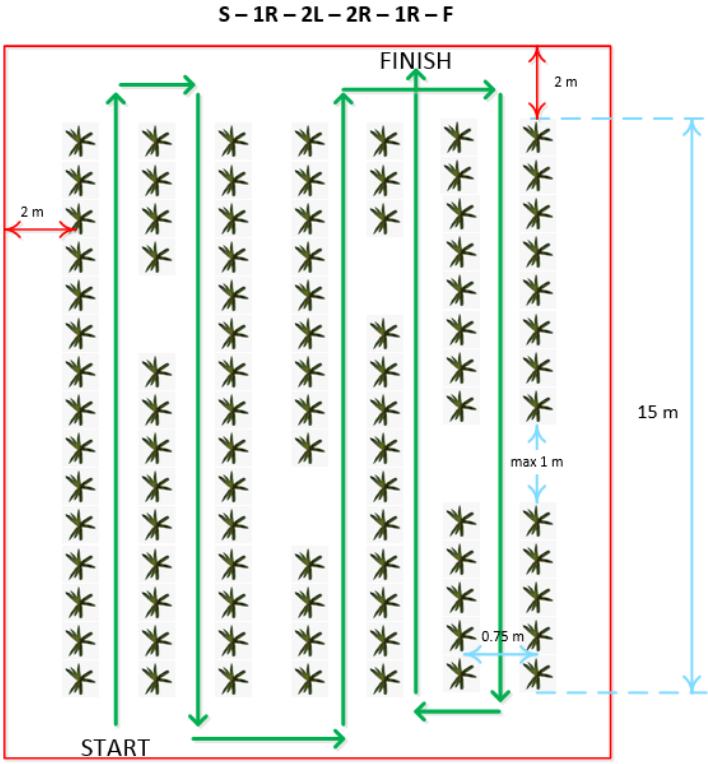
Final points = (agronomic idea + technical complexity) * performance.

The task 5 is optional and will be awarded separately. It will not contribute to the contest winner 2018.

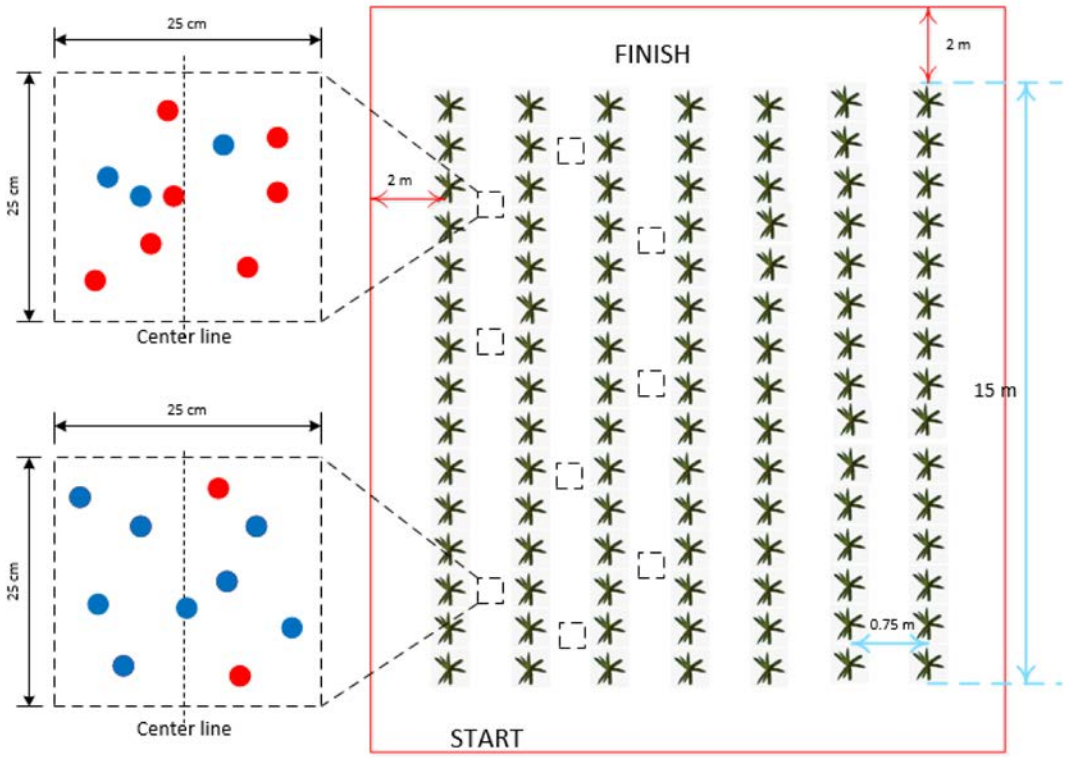
Appendix



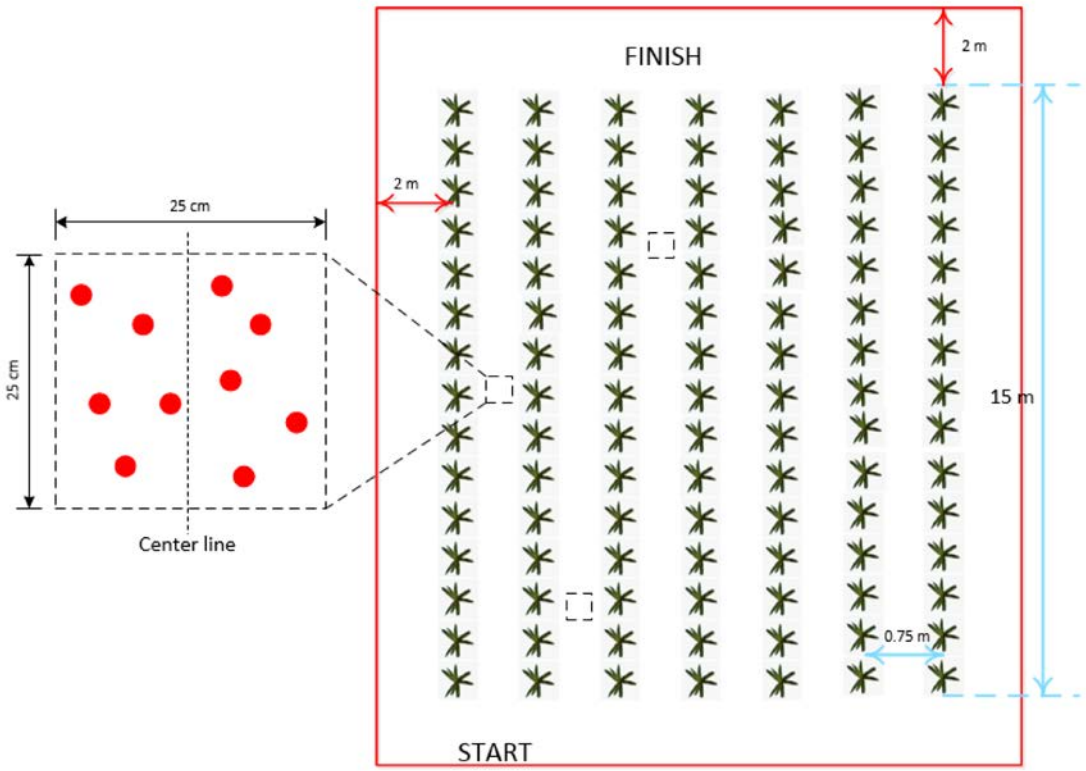
Picture 1 – Dimensions and row pattern for task 1



Picture 2 – Dimensions and example (!) row pattern for task 2



Picture 3 – Possible locations of the weeds for task 3. Two zoom areas of 25 cm x 25 cm of two weed patch (with different combinations of blue and red weeds) are also indicated.



Picture 4 – Possible locations of weed patches for task 4. The zoom area of 25 cm × 25 cm of one weed patch (with ten red weeds) is also indicated.

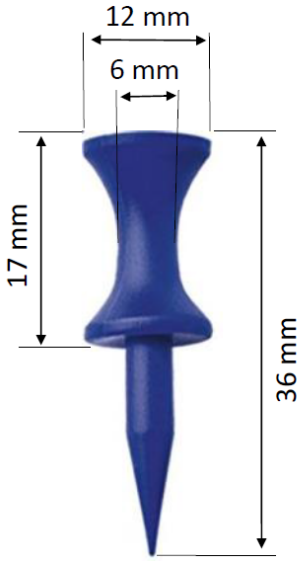
Appendix B

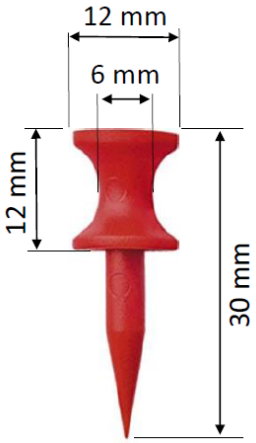
Red tees

https://www.amazon.de/40-Plastic-Step-Tees-Abstandtees-Stufentees-Eisen/dp/B00TE28HT6/ref=sr_1_1?s=sports&ie=UTF8&qid=1523354524&sr=1-1&keywords=Stufentees+rot

Blue tees

https://www.amazon.de/40-Plastic-Step-Tees-Abstandtees-Stufentees-Hybride/dp/B00TE2HOJA/ref=pd_sim_200_1?_encoding=UTF8&psc=1&refRID=MEZEVTJC768PYQTZM0JP





HWG & DP 2018.04.16

STEKETEE BULLSEYE

Alexander van Tuyll-Serooskerken[1], Anna Lisa Nooren[1], Cor Feitsma[1],
Dana Vernooij[1], Harmen van der Vliet[1], Jaap Weerheim[1], Merel Arink[1], Michiel Mans[1],
Robert van de Ven[1], Ronald Konijn[1], Stef Wesselink[1], Jan Morssin[1]

[1)] *Wageningen University & Research, Fram Technologiy Group (FTE), The Netherlands*

1. General robot setup

For robots, an interface with some useful tools has been built. This interface is called Robot Operating System (ROS). ROS enables the development of robots in a modular fashion. Through programming in modules, different developers can work independently on different modules of the robot in their own preferred programming language. Thus, using this module system a program written in C^{++} can communicate with a program written in Python. Also, for ROS a lot of pre-built modules are available for usage, like a state machine package, a Kalman filter package, a navigation stack package and so on.

The robot used in this study uses ROS and thus comprised a lot of different modules:

- An odometry module consisting of a module for the Inertial Measurement Unit (IMU), one for the motor controllers and one for the Kalman Filter combining the data from both odometry sources.
- A navigation module. The navigation module receives data from the state machine about which state is currently being practiced and based on this, different parts of the navigation module are used.
- A (camera-based) vision module for detecting golf tees to fulfil task 3 and 4.
- A collecting device module for picking up golf tees to fulfil task 4.
- A state machine module is in place for managing the whole robot. The state machine integrates all above-mentioned modules. Within the state machine each part of the robot can be launched and controlled.

State machine setup

The primary goal of the robot was to navigate properly through the maize field. This required different navigation states. These states are shown in Figure **1.** Each coloured block represents a different state, in which the robot should behave in a specific way. In the state 'Before Row', the robot should find the row and make sure it drives straight towards this row. In the state 'In Row', the robot should drive smoothly through the row, as quickly as possible and without hitting any plants. For the state 'After Row', the robot should drive straight out of the row to have enough room for the turn. In the state 'Turn Out', the robot should make a turn so that it stands perpendicular to the rows. For the state 'Drive Straight Headland', the robot should drive alongside the rows and skip the number of rows given. The last one is the 'Turn In' state, in which the robot should make a turn to get into the row.

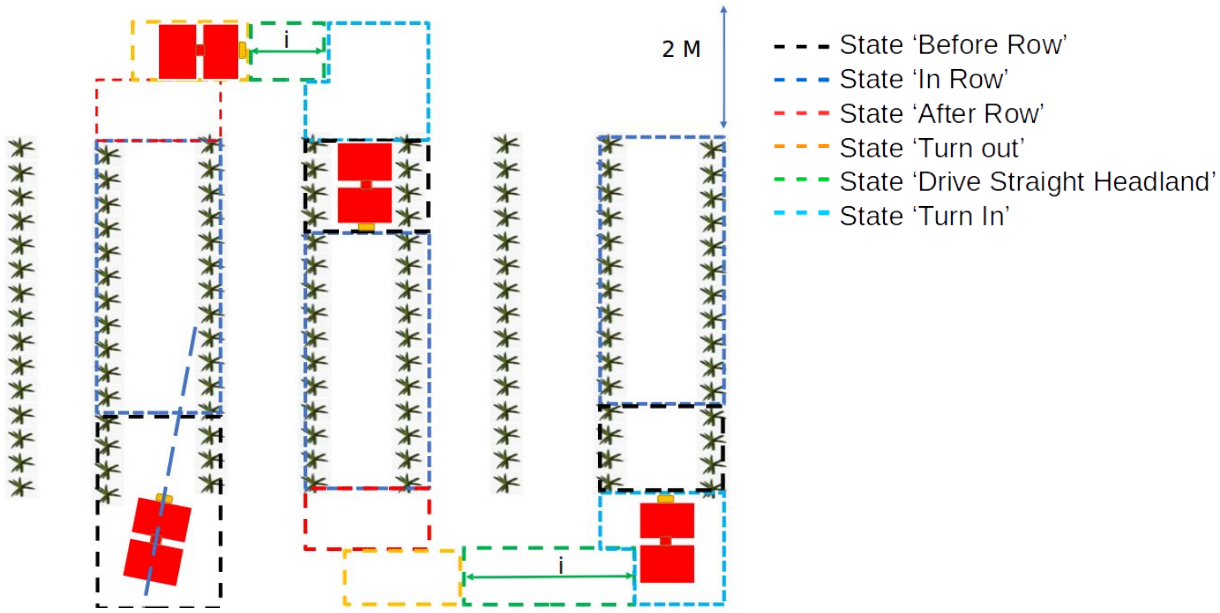


Figure 1 Navigation states

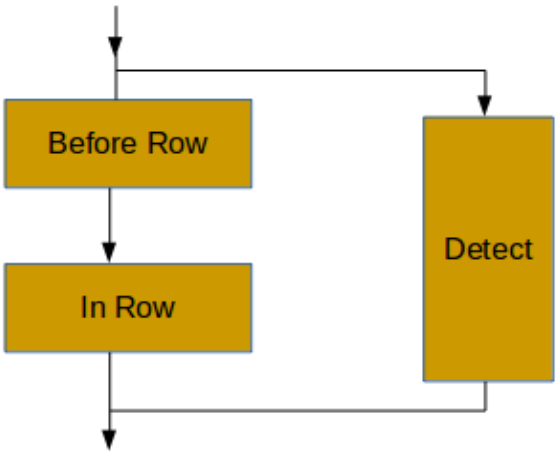


Figure 2 Visual overview of state 'Detect' running parallel to navigation states 'Before row' and 'In row'

The state machine also manages the vision and collecting modules used for task 3 and 4. In task 3, the robot had to detect golf tee patches while driving before and in the row. This required a state parallel to the state 'In Row': the state 'Detect'. During this state, the state machine provides the vision scripts with the data from a camera module. It then retrieves the information from the vision module and connects this with the sound module for the robot to give the right signal for detecting a patch of weeds. Figure 2 visualizes how the detect state of task 3 is integrated with the navigation states. The same setup was used for task 5, although then the state 'Detect' was used to detect missing plants instead of golf tees.

To be able to pick up the detected golf tees to fulfill task 4, a pick-up state was added. Using the state structure for detection, now when a patch was detected, the pick-up state was activated to collect the tees. The navigation and detection were stopped and the robot drove forward, lowered its pick-up mechanism and collected the tees. When this was done, the robot went back to the parallel states as developed for task 3. Figure 3 visualizes the setup to achieve this.

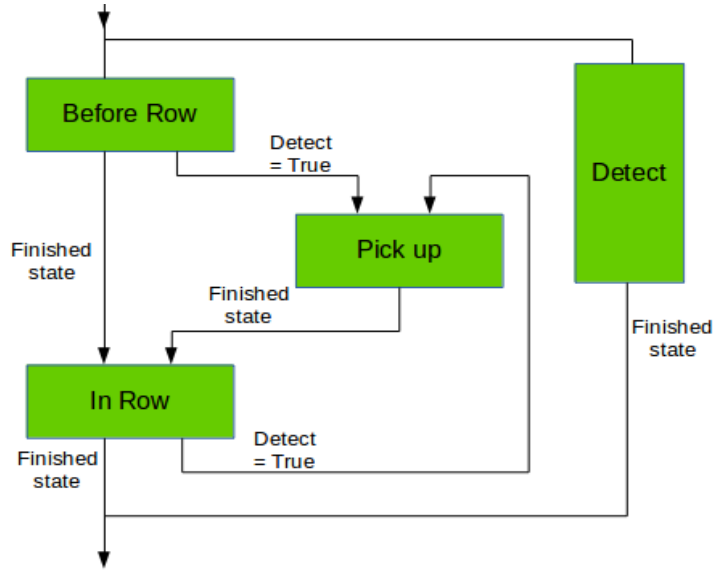


Figure 3 Visual overview of the state setup for golf tee detection and picking up.

In the next sections the principal applied for developing and improving each module of the robot is described. This starts with a section about odometry, explaining how the robot measures its driven distance and how accurate these measurements are. Following that, the development of the navigation scripts is described showing how the navigation module is configured and what calculations are made for the robot to determine its route. Next, the vision module for task 3 and 4 is discussed, followed by the module developed for picking up the golf tees in task 4.

1.1. Odometry

Odometry is a collective term for the determination of a robot's position and velocity, used for its localisation. This year, a Kalman filter was used to give robust odometry by combining different odometry measurements. These measurements came from an inertial measurement unit (IMU) and wheel odometry. GPS was not used since this is not allowed according to the Field Robot Event's rules and regulations.

1.2. IMU

The inertial measurement unit (IMU) is a device that measures linear and angular acceleration using a gyroscope and accelerometer. It was used for odometry on the robot, the most important function being the determination of the yaw, which was tested extensively here. At first, when only used IMUs were available, these had to be tested. In the end, however, only a brand new MTi-300 was installed on the robot.

Materials and methods

The following IMUs were available, all manufactured by Xsens:

- 2 MTi-30s (2015)
- 1 MTi-300 (2014)
- 1 MTi-300 (2018, brand new)

All IMUs were set to give an output of 400 Hz.

The software used was MT Manager, developed by Xsens, on both Windows and Linux computers. On the robot, the IMU was controlled by the Xsens driver node. Data was also obtained through making ROS.bag files. Using summation in the spreadsheet, accelerations and velocities could be converted to distances to determine the drift over a long period of time. These were graphed to visualise the long-term effects of any drift.

Several tests were done on the IMUs:

- Drift with no displacement
- Precise yaw tests
- Magnetic disturbances

Drift with no displacement (2015 MTi-30)

There were two tests that were done to determine the IMU's bias for the yaw:

- moving the IMU with a net-zero displacement
- keeping the IMU entirely still.

These measurements were done over a period of 10 minutes. A perfect IMU would of course give no change in yaw throughout this time. Also, these tests were done with MTi-30s since no other IMUs had been found at that point. However, since after these tests an MTi-300 was found, subsequent tests were no longer done with MTi-30s.

Precise yaw tests (2014 MTi-300)

To test the yaw more precisely, lines with angles in increments of 10 degrees - from 0 to 90 - were drawn on sheet of paper, which was fixed to a table. The IMU was then moved to each of these lines and kept there for a few seconds. This was a good way to test the yaw measurement on its own.

These more precise tests were done to determine the best setting for the IMU: active heading stabilisation (AHS), in-run compass calibration (ICC), neither, or both. ICC removes consistent magnetic disturbances by calibrating the compass. AHS aims to reduce random magnetic disturbances by keeping track of the total magnitude of the magnetic components. The tests were done for each of these settings.

Magnetic disturbances (2018 MTi-300)

To observe magnetic disturbances, the IMU was placed near computers and other electronic devices. In particular, the IMU was placed in different places on the robot whilst running MT Manager on a Windows laptop. Using MT Manager, noise could be observed. This is how it was decided where to place the IMU on the robot.

Distance tests on the robot (2018 MTi-300)

As a final test, the IMU's odometry measurements were obtained by driving the robot for 10 metres, ten times. From the measured acceleration, integration was used to calculate the

distance measured by the IMU. This final test was done once all other hardware had been installed on the robot.

Results

The IMU measures linear and angular acceleration. With this the movement of the robot can be calculated. Below, the outcomes of the IMU tests are described.

Drift with no displacement (2015 MTi-30)

In the graph below, the yaw is shown over a period of 10 minutes. As can be seen, the yaw given by the 2015 MTi-30 fluctuates dramatically. After three and a half minutes of the IMU standing still, the yaw had changed by over 90 degrees, as shown in Figure 4:

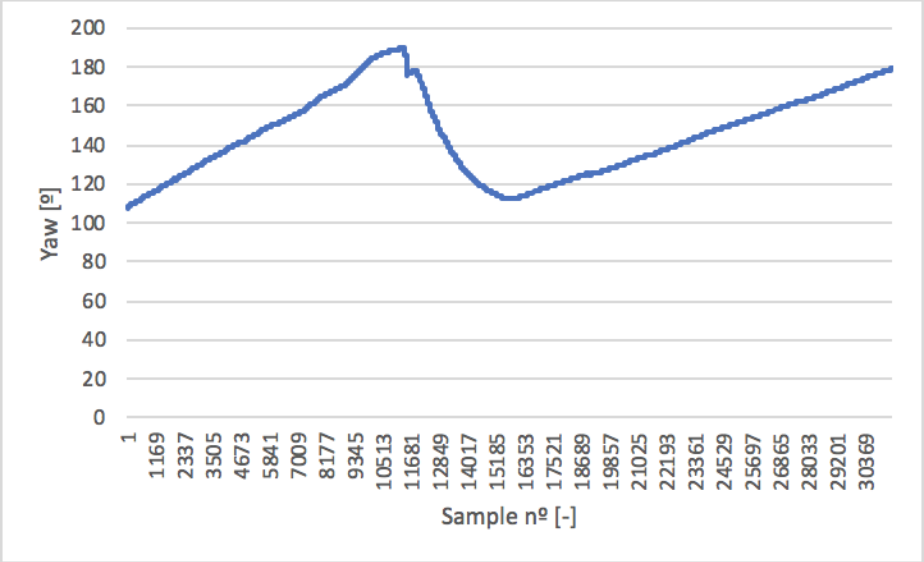


Figure 4 The yaw, in degrees, measured at a sample rate of 50 Hz. The IMU was kept still.

Figure *5* shows the results of moving the IMU around with no net displacement. Here, a deviation can also be seen. After eighty seconds, the yaw had already deviated by around forty degrees.

The conclusion from these measurements was that the MTi-30s were unusable for our purposes. Therefore, subsequent tests were done on the MTi-300s.

Precise yaw tests (2014 MTi-300 and 2018 MTi-300)

What follows are the results for the yaw on the 2014 MTi-300.

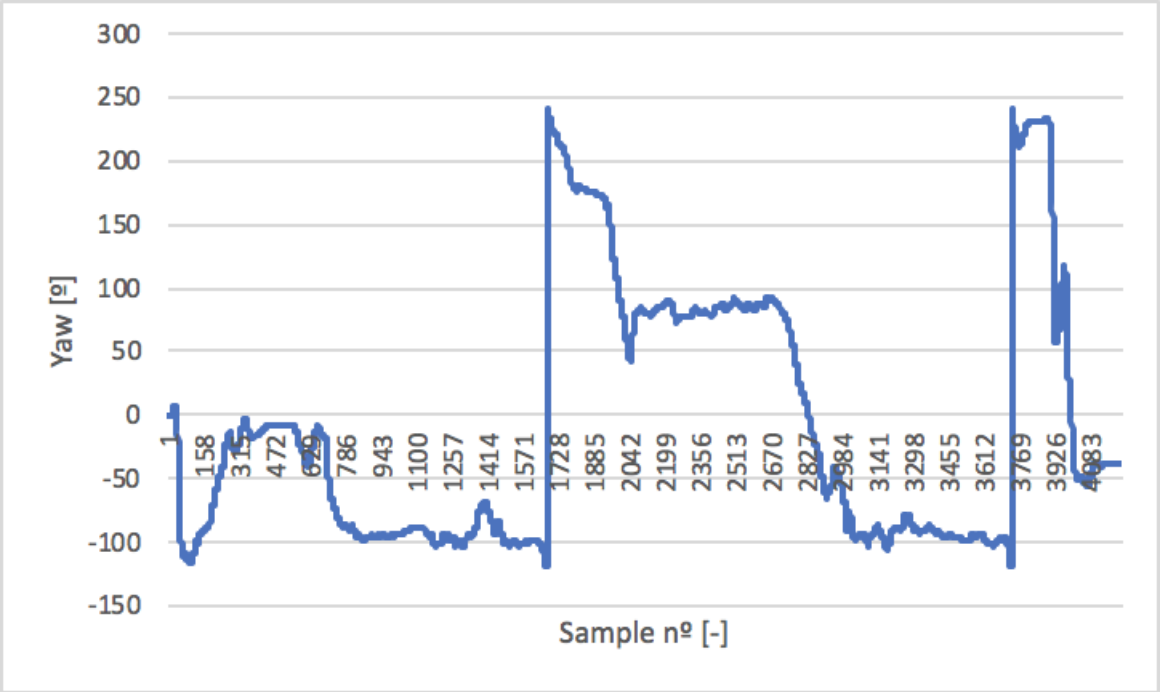
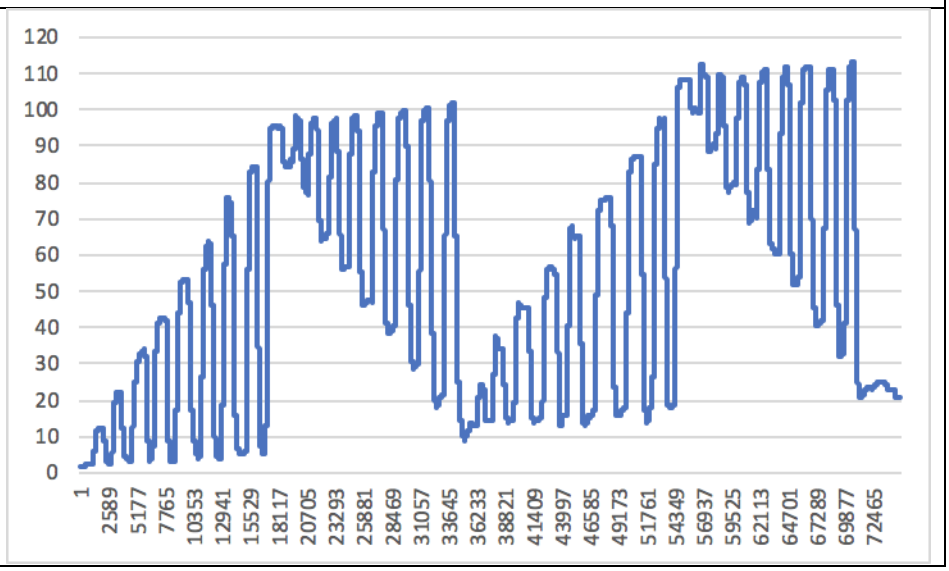
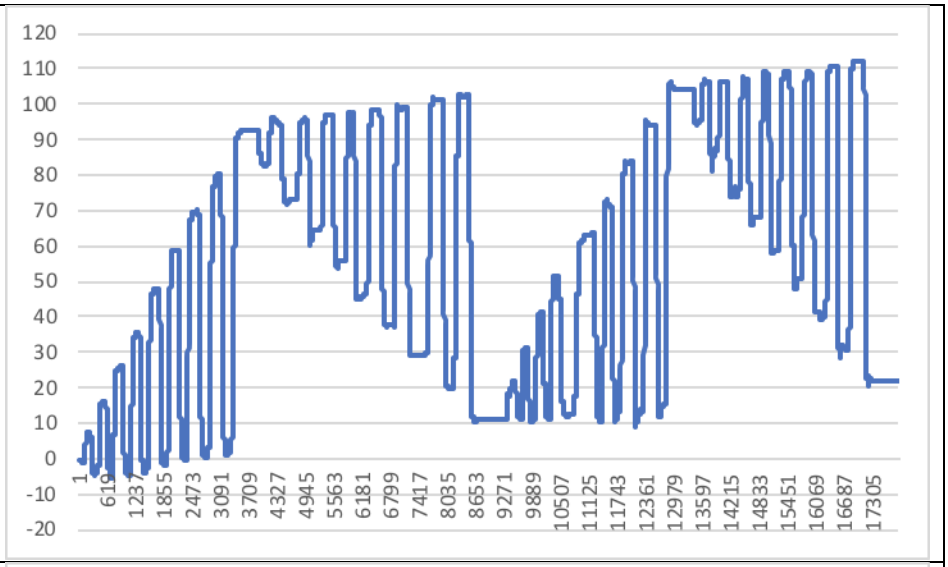
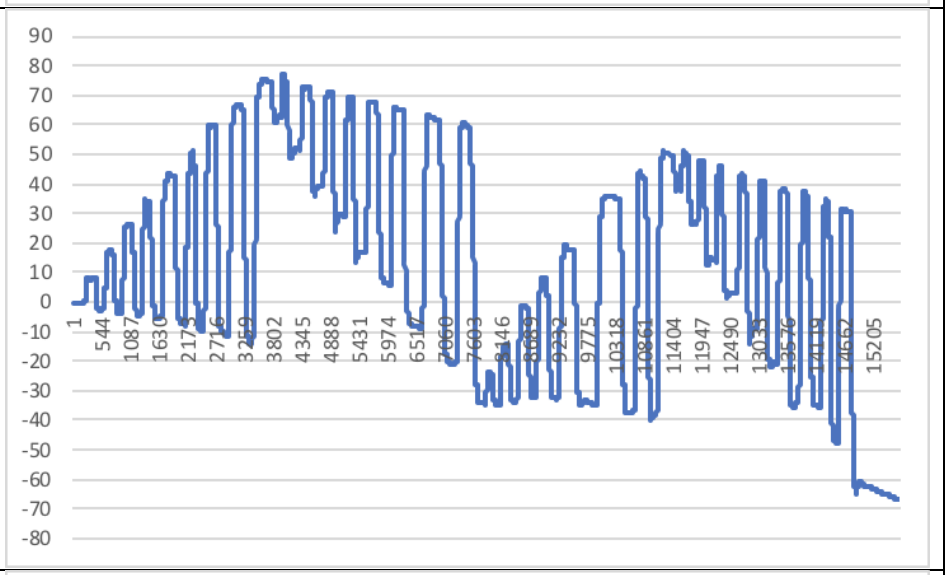
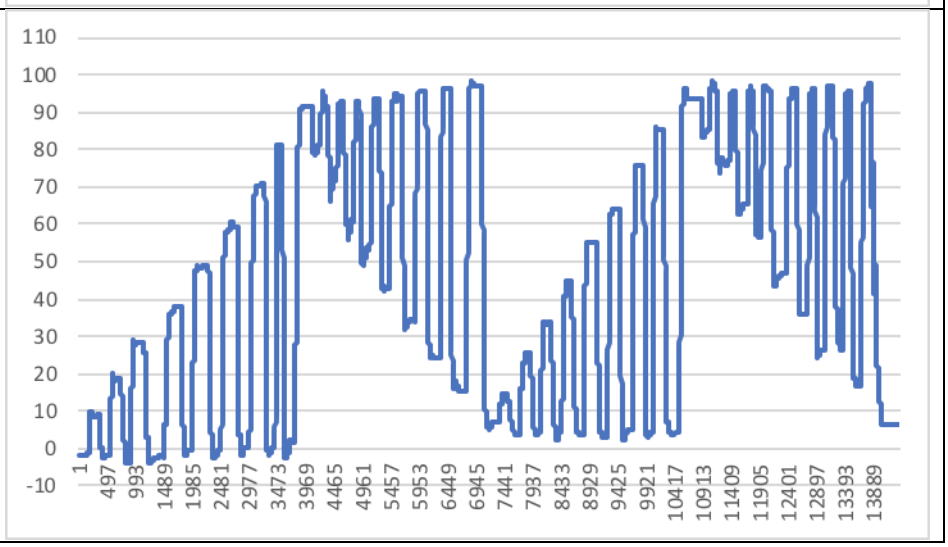


Figure 5 The yaw, in degrees, measured at a sample rate of 50 Hz. The IMU was moved around but ended in the same position and orientation as it began.

Table 1 Results of the yaw on the 2014 MTi-300

	Sample nº (-) against yaw (º)
Default settings	

AHS only	
ICC only	
ICC + AHS	

The best results seem to be with the IMU where ICC and AHS (details explained in materials and methods) are both on simultaneously. AHS gave the second-best results. The default configuration was third and the configuration with just ICC turned on was the worst.

Once the 2018 MTi-300 was available, the same measurements were done, with AHS turned on. The results were noticeably better than with the older MTi-300, as can be seen in Figure 6.

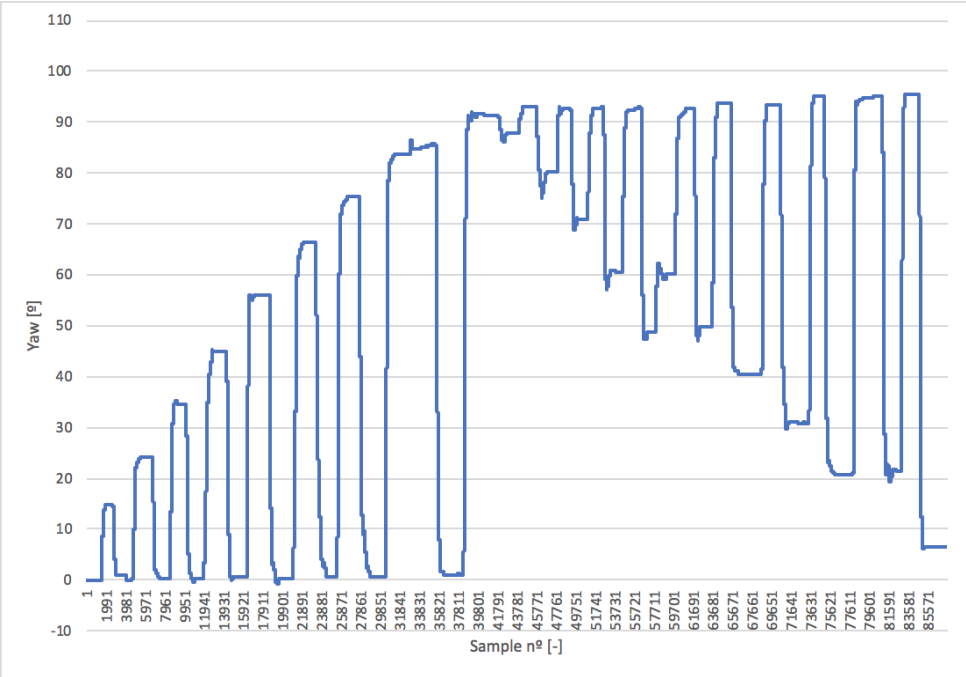


Figure 6 The yaw of the 2018 MTi-300, measured at a frequency of 400 Hz. The deviation, whilst still present, is a lot smaller than on the older IMUs.

Magnetic disturbances (2018 MTi-300)

Near most places on the robot, there were magnetic disturbances, as can be seen in Figure 7 to Figure 10. These came from the motor controllers and the electronics in the cases. The magnetic disturbance between the two main cases on the right-hand side of the robot was very noticeable. However, on the left-hand side between the two cases, the magnetic disturbance was a lot smaller. Therefore, it was decided to install the IMU there. A picture of the placement of the IMU can be seen in Figure 1. That said, new hardware was installed on the robot afterwards, so the relevance of this result is questionable (see 'Distance tests on the robot').

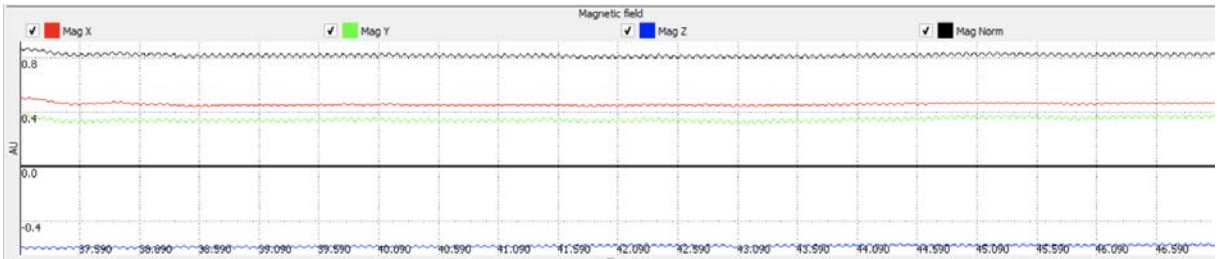


Figure 7 between cases, on the right.

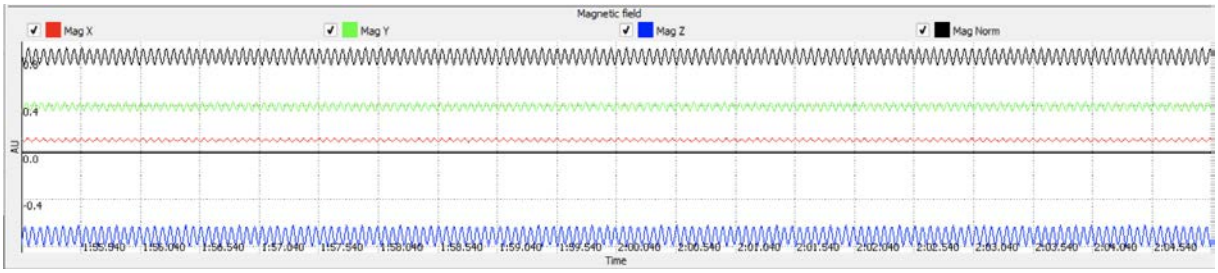


Figure 8 on the rear box of the robot.

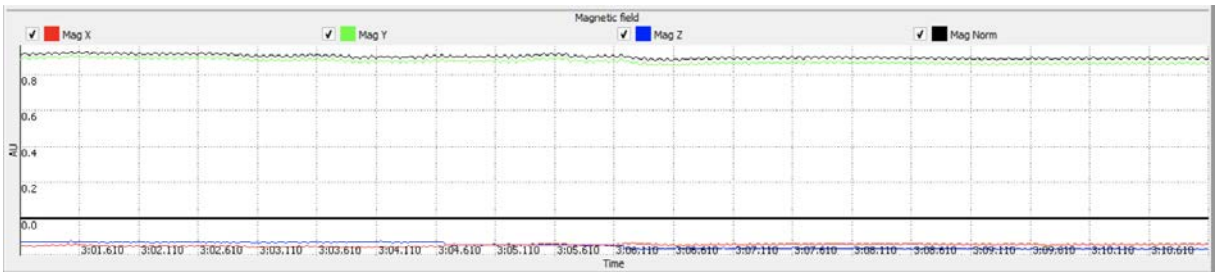


Figure 9 between the cases, on the left.

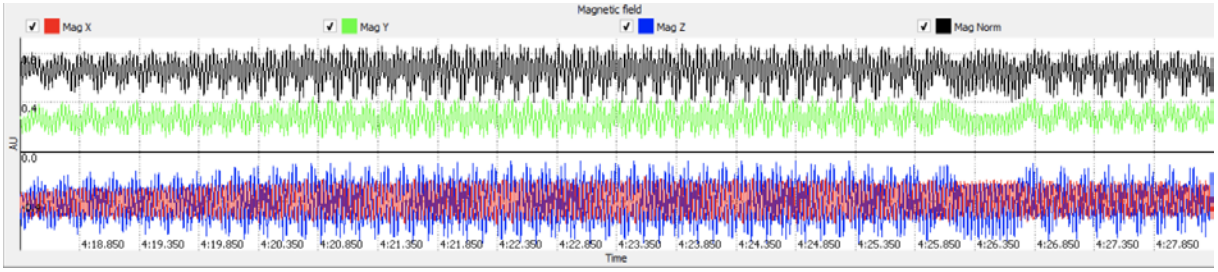


Figure 10 on the front box of the robot.

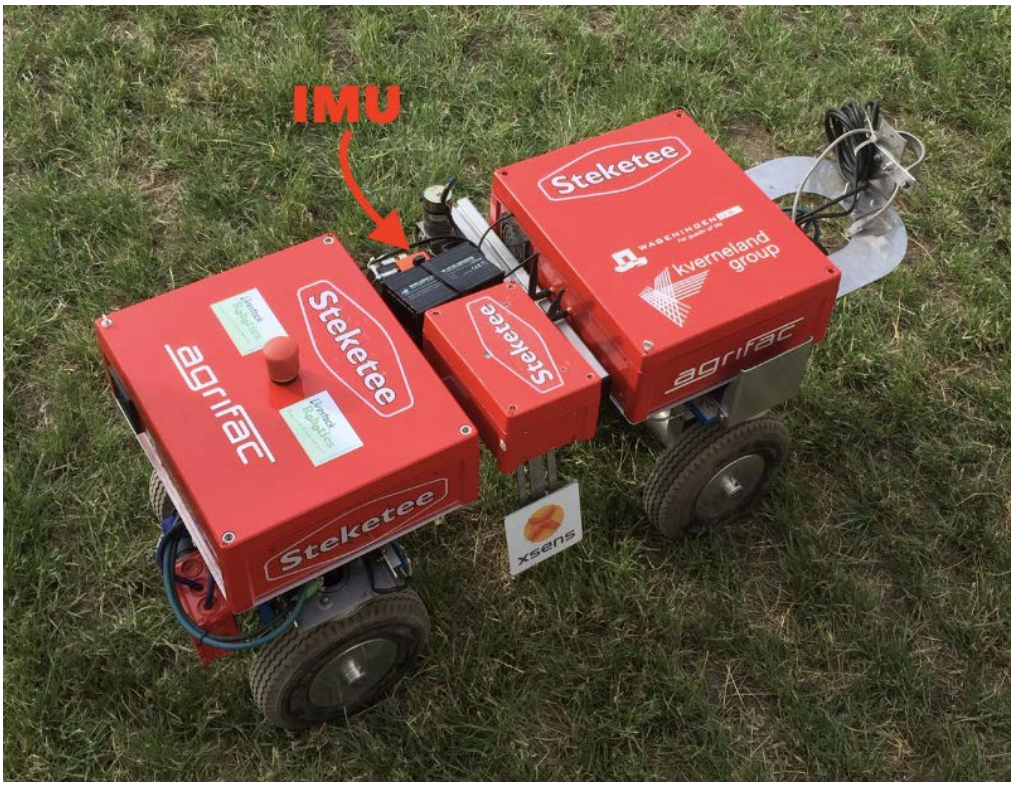


Figure 11 Placement of the IMU on the BullsEye

To summarise:

- MTi-30 (2015): large errors, many of which were unpredictable.
- MTi-300 (2014): smaller random errors than the MTi-30s, and predictable consistent errors which could be filtered out linearly.
- MTi-300 (2018): no consistent errors. Random errors were negligibly small. This IMU was only available after all the other tests were done, and when it could be tested, it was found that this IMU was far better than the others.

Final decision

Although the AHS-ICC combination worked best, in the end it was chosen to only turn AHS on. This is because for ICC to work well, the magnetic disturbance must be constant, as explained in materials and methods. This was likely the case during our testing of angles, but it could not be guaranteed with the IMU installed on the robot.

Distance tests on the robot (2018 MTi-300)

The results of the IMU odometry test show an average distance of 701.4 metres and a standard deviation of 785.7 metres. This is clearly significantly different from the desired 10 metres, with a p-value of under 0.05.

Discussion
All IMUs were tested extensively throughout the development of the robot. One point of improvement could be to do with more realistic testing. The IMUs were always tested indoors. Perhaps a final test could have been done outdoors, as a final check. That said, the IMU seemed to contribute well to the Kalman filter's localisation output when driven around outside. Another point is that the effect of magnetic disturbances on the IMU was tested before the addition of other hardware (Arduino, pick-up mechanism, and so on). This was clearly visible as the last test (distance test on the robot) showed huge deviations in the measurements.

Conclusion
Not all IMUs worked well, but the brand new 2018 MTi-300 did a satisfactory job at estimating the yaw, its most important contribution to the odometry. Magnetic disturbances did vary greatly, so it was decided to place the MTi-300 on the left-hand side of the robot, between the two main cases. That said, adding new hardware was a likely reason why the latest measurements made were completely inaccurate.

Recommendations
This year, the IMU worked well for estimating the yaw. The only improvements would be to make the IMU's setup more robust, to avoid errors. For next year, we recommend trying to build a Faraday cage around the IMU to shield it from magnetic disturbances, of course making sure this does not affect the compass measurements. This may be important as extra hardware is added to the robot (a probable source of new disturbances). What may also be worth trying is installing a second IMU, and through the Kalman filter obtaining a single result which is more robust.

1.3. Wheel odometry

Wheel odometry was used to determine the velocity of the robot in the field. This can be helpful in navigating the robot through the rows.

Materials
In the BullsEye, two different type of motor controllers are used: one to control the wheel motors and one to control the steering motors.

Currently there are four EPOS 24/5 motor controllers in the BullsEye, from Maxon Motor (see Figure 12). From these motor controllers the velocity, position, current and state can be read out.

Figure 12 EPOS 24/5 motor controller from Maxon Motors

To control the steering motors, four Solution Cubed MotionMind Rev2 motor controllers (see Figure 13) are used. The first motor controller is connected to the computer using an on-board serial port.

Figure 13 Solution Cubed MotionMind Rev2 motor controller.

These motor controllers are used to retrieve the steering angle and the velocity of the wheels. Together they are used to calculate the wheel odometry.

Methodology

To let the BullsEye move, a target speed has to be sent to the motors. This target speed is in rpm. However, for the BullsEye, m/s are used instead of rpm because only SI units are used. Therefore, a conversion factor is used to convert the m/s into rpm. This conversion factor was determined as follows: on the floor, a distance of 10 metres was set out. Then, the BullsEye drove these 10 metres at full speed (at which the maximum frequency of rotation). This was repeated 10 times. Based on the average time, the maximum velocity could be calculated. With this maximum velocity, the conversion factor could be calculated.

When the actual velocity could be read out, the odometry could be tested. This was done to check if the BullsEye produced the right odometry. If for example, the BullsEye has a deviation of 10 metres in the odometry, it can be questioned whether the wheel odometry is right and whether it should be used. To test this, the BullsEye had to drive 10 metres in a straight path. This was repeated 10 times to get a better estimation instead of a estimation based on one

measurement. This was then tested if there if the difference between the actual position and the predicted position (so the 10 metres) was significant. So only the position displacement in the x direction was tested.

Results

The maximum speed of the BullsEye was 1.32 m/s with a standard deviation of 0.08 m/s. By knowing this the conversion factor could be calculated. For converting the rpm to m/s, the rpm that is read out should be divided by 4300. At full speed, this gives an output of 1.3 m/s.

The results of the wheel odometry test show that five of the ten measurements did not produce usable data. This on itself is already a problem. The other five measurements show an average distance of 11.00 metres with a standard deviation of 0.24 metres. For these results, there is a significant difference in the wheel odometry for a significance level of α = 0.05.

Discussion

The estimate of 1.3 m/s could likely be improved. It is not very accurate because the way of measuring was not optimal. Two lines with a distance of 10 metres of each other were drawn on the ground. The time was measured with a stopwatch. This was all done by people, so not optimal. However, because it was repeated ten times the estimate is an average of ten runs. Therefore, it was also not very inaccurate.

The results show that the wheel odometry is not working yet. Five of the ten measurements do not show any usable data. This is a problem which should be solved in the future to get a more liable wheel odometry. The cause of this problem could be that the odometry gets a velocity that is out of range which causes the odometry to get disturbed.

For the five measurements that produced usable data, the difference in wheel odometry could be caused by slip. The slip could cause the difference because the measured distance is longer than the actual distance. The wheels turn more so the BullsEye measures a longer distance that is actual travelled. In the calculations, slip is not taken in to account because this depends on the type of floor the BullsEye is driving on. If the BullsEye is driving on a smooth floor or on a sandy soil, the chance of slip is higher than when the floor is a bit rough. In sandy soils the slip can be decreased by giving the BullsEye spike wheels or at least wheels with more profile.

Another cause could be the conversion factor that is used to convert the rpm in to m/s. This was determined empirically. With this conversion factor the velocity is calculated. This velocity is than used to calculate the odometry. If the calculated velocity is a slightly higher than the actual velocity, than the odometry is higher too.

Furthermore, the wheel odometry is only tested for the x direction. During the measurements, the robot only drove a straight line. Thus, only the position displacement in the x direction was tested. Most of the time, the BullsEye drives straight when driving in a maize field. However, on the headland, the BullsEye turns. On the Headland the y direction is important. This was not tested for now.

Conclusion

The maximum velocity of the BullsEye is 1.32 m/s, the wheel odometry of the BullsEye is is far from perfect. The wheel odometry can't be used in the BullsEye on its own. It should be improved to get better results.

Recommendations
The most important thing that needs to be done first, is to check out why the odometry does not always produce usable data. When the odometry is working well, the wheel odometry should be tested better. Now, only it was only tested for a straight 10 metres in x direction. Test the wheel odometry first also in y direction and for turns. To check whether it really functions optimal, the odometry should be tested by letting the BullsEye drive a parkour, measure the actual distance of the parkour and then compare this with the wheel odometry of the BullsEye. If there is no significant difference, or the difference is within the boundaries, the wheel odometry is ready to use. The wheel odometry together with the IMU odometry and the integration with the Kalman filter, the whole odometry should be implemented in the scripts. This can be helpful in locating the BullsEye.

1.4. Kalman filter

To make the odometry useful, the position of the robot needs to be known. It is also useful to know the current velocity of the robot. Our two sources of odometry, the wheel odometry and the IMU odometry do not provide a robot position. To go from the wheel and IMU odometry to a position, it is possible to convert these measurements using differentiation. However, this kind of method resulted in a big error in the IMU odometry when used by the Field Robot team of last year (Aker et al. 2017)**.** A Kalman filter can overcome this. Kalman filters can be used on all sorts of data to get rid of random disruptions in the measurements. Kalman filters can also be used to combine data from several sources to make this data more reliable. In the BullsEye, the Kalman filter is used to combine the two sources of odometry (wheel and IMU). These odometry sources do not give out the robot position, but only the robot orientation, while the odometry we want to obtain should give the position and orientation of the robot. The Kalman filter is also used to integrate these velocities and accelerations to the current robot position.

Methodology
The Kalman filter is implemented by adding the *robot_localization* node to the launch file of the robot. From the wheel odometry, we used X, Y and Z velocity; and X, Y and Z angular velocity.

As the best results of the IMU came from the orientation, it was decided to use the X, Y and Z orientation. We also used the Kalman filter to remove the gravitational acceleration.

In order to test the Kalman filter, the BullsEye was driven a distance of 10 metres 10 times. This was done in order to see how accurate the odometry of the robot was after driving through a row, which is what the robot does the most.

Before the picking up mechanism was added, we tested the behaviour of the filter after a longer time. In order to get the behaviour of the Kalman filter after a longer time, the robot was driven a longer distance. The robot was started in the workshop, there it was driven

outside, where a couple of 180 degrees stand 90 degrees turns were made. The 90 degrees turns were to simulate task 2, where we have to drive on the headland and the 180 degrees turns were to simulate task 1, where we turn immediately into the next row.

Results

When testing the robot with driving 10 metres, there were two distances with a measurement of 5.46 and of 6.62. These were not considered in taking the average and the standard deviation. The average of the remaining 8 measurements was 11.32 m with a standard deviation of 0.13 m. The results show that there is a significant difference in the wheel odometry for a significance level of α = 0.05.

The results of the Kalman filter before the picking up mechanism are shown in Figure 14 and Figure 15 below. The Kalman filter was first tested in a basic setup with the wheel odometry for the linear velocity and the IMU for the orientation. This resulted in unusable data, the position of the robot would become very big immediately. To resolve this, a threshold was added to the measurement of the wheel odometry. This resulted in mostly usable data, however the robot was sometimes still driving sideways or backwards, when in reality it was actually standing still. These results are shown in Figure 14. To improve this, it was decided to add the linear acceleration provided by the IMU to the Kalman filter. These results are shown in Figure 15. These results were deemed satisfactory.

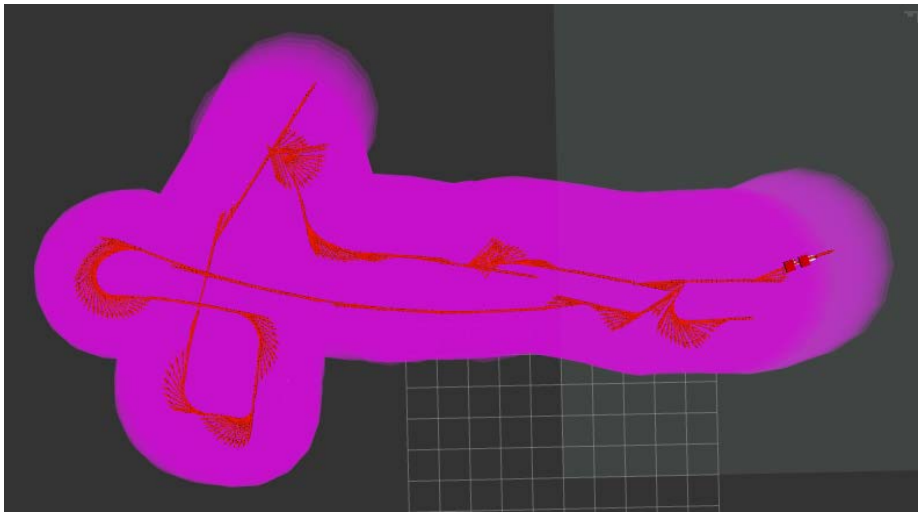


Figure 14 Result of the Kalman filter with the IMU used only for yaw.

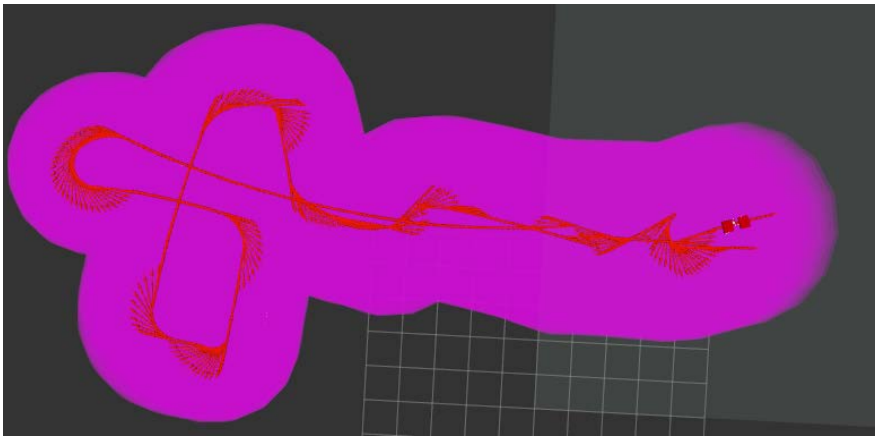


Figure 15 Result of the Kalman filter with the IMU used for yaw and for linear acceleration.

Discussion

The Kalman filter worked quite well, but not perfectly. Not all parameters of the Kalman filter were completely examined. Some additional parameters could have been fine-tuned to improve the quality of the filter. The Kalman filter currently uses the velocity from the wheel odometry. Results could be improved by using the position of the wheel odometry.

Another parameter that could be used is *process_noise_covariance*. However, this parameter contains a lot of values that can be changed, making it quite difficult to calibrate it properly. This can improve the filter more but might take too much time.

Before the picking up mechanism was added, the results of the Kalman filter was satisfactory. However, when after the event the Kalman filter was tested again when driving 10 metres, the result of the filter was significantly different than the 10 metres that was driven.

Conclusion

Before the event, the results of the Kalman filter were satisfactory. However, after the addition of the last components, the results had significantly decreased in quality.

Recommendations

The Kalman filter should be tested better. The results that were found before the event came from a single measurement and after the event the results of the filter were not satisfactory. It is recommended to test the filter in a situation that is similar the how it will be used during the event.

The output of the Kalman filter can be improved by using the position from the wheel odometry and by tweaking the parameters.

1.5. General remarks on odometry

Kalman filter takes different quantities as an input, compared to the outputs of the odometry sources. For example, wheel odometry is given as a position. However, when used in the Kalman filter, wheel odometry is used as a velocity instead. This difference could be worth exploring, as it could lead to inconsistencies when comparing the Kalman filter's output to the raw wheel odometry output.

2. Navigation

For navigation, two separate options were explored. The first involved navigating using the Navigation stack provided by ROS. The other navigation option was to navigate using live navigation. Both options will be discussed in the following sections.

2.1 Navigation stack

The ROS navigation stack is a package specially designed for autonomous navigation of robots (Fathoni, 2018). The navigation stack consists of five different parts, see Figure 16. First, a setpoint in the robot's surroundings is given to the global planner; this uses a given global map to plan a path for the robot (Appeldoorn, 2018). The global planner computes the shortest route to the setpoint, while keeping a given distance to the objects in the map. The map the global planner uses comes from the global costmap (Lamprianidis, 2018). This part takes a map given by the user for the entire environment and ranks each pixel in the map. An obstacle is a forbidden zone, close to the object is also not recommended. This information is used by the global planner to create its path. To be able to interact with the environment while driving, the local planner is used. This local planner uses the plan given by the global planner and information from a local costmap to compute velocity commands for the motor controllers (Salam, 2018). If the global costmap or local costmap consist of too many objects or objects too close to each other, the global planner or local planner will not be able to create a path. This will result in activation of the recovery behaviours. This means the robot makes some turns to get new laser data to try and find new openings (Ken, 2016).

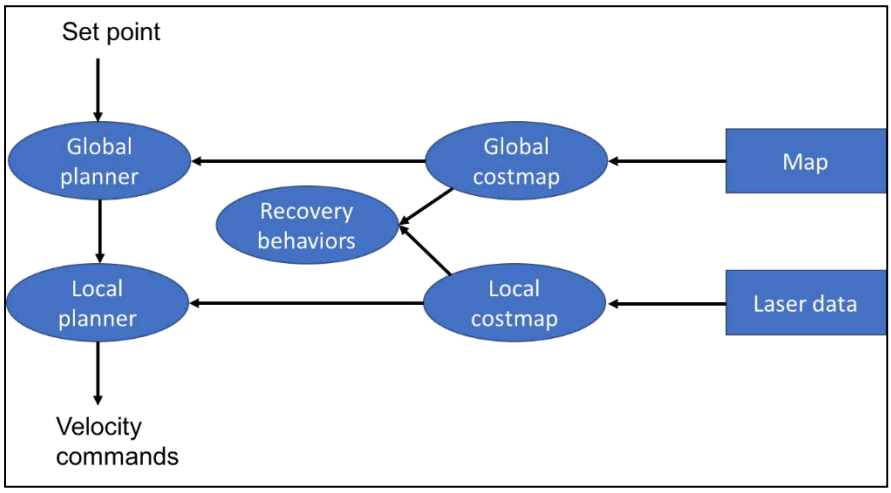


Figure 16 Overview of the navigation stack

Effort was put in trying to get the navigation implemented on the robot. First, the navigation stack was tried using default settings. However, although all inputs (odometry, laser data) of the navigation stack were present, the robot did not move at all. An attempt in solving this problem was to add a global costmap. This map was made using a simple drawing program 'Pinta Image Editor'. This map did not seem to do anything; the robot drove through the lines and the area of impact around the lines seemed to be ignored. A second attempt was done adding the *AMCL* node. The documentation of the navigation stack by ROS made it seem that

integration of an *AMCL* node was not necessary. *AMCL*, adaptive Monte Carlo Localization, is an algorithm to localize the robot. It estimates the possibility of the robot being at a certain point for each point on the map (Fox et al. 1999). A tricky part is the maize field, this does not have many identical points, which makes it hard for AMCL to determine where on the map the robot is. Still, it should be able to distinguish being in the row from being at the beginning or end of the row, or on the headland. However, when the *AMCL* node was implemented for testing purposes, the robot started to move; which it did not do without the AMCL node. Still, it did not work perfectly, because it did not seem to update setpoints. Another plan was that it was unable to update the setpoints coming from the state machine. This resulted in the robot seemingly driving to the setpoint, but then starting to drive around like it was trying to reach the same point again. After that, by means of trial-and-error a lot of parameters were adjusted. Unfortunately, this never resulted in a properly functioning navigation stack.

Conclusion

The navigation stack did not work. The robot could drive, but not how it should. It damaged plants and did not reach its setpoints.

Recommendations

It is highly recommended to dig into the theory and algorithms behind the navigation stack to tackle this problem through the underlying principles of the navigation stack. For FRE 2018, a more superficial and pragmatic approach was used but this was not sufficient. There were some teams at the Robot Event who had it working, so it should be possible for next year to achieve this as well. The navigation stack would make it possible to use global mapping, enabling the route planning in advance. Furthermore, once the navigation stack works, a global map of the contest field can be made. This was not required for FRE 2018, but a global map has many uses. The AMCL part of the navigation stack can be of help to the state machine to determine in which state the robot should be, because the AMCL calculates the possibility of the robot to be at a certain point in the map. This can make navigation more robust.

2.2 Live navigation

For the live navigation different approaches to setpoint calculation were done for each possible state the robot was in.

In Row

In order to come up with ideas to evaluate the options for live navigation within the row, first the code from the previous year was inspected. In the code of the Robatic Team from 2017 we found that for navigation within the row, the previous approach was to try a few different angles and latitudes of the robot and see which one would give the least obstacles in a straight path down the middle. Then, the route with the least obstacles would be considered to get to the desired position. Then a goal velocity would be passed on to the robot to reach this position. A downside of the method was that only 5 different options for angles and latitudes were considered and thus the navigation wasn't ideal, since, there are infinite positions in which the robot could be.

Since, the previous method was not ideal, a new method was found. This method took all the points from the laser scanner within 40 degrees behind the laser scanner up to 40 degrees in

front of the laser scanner on the left side and the right side. The 40 degrees behind and in front of the laserscanner have been set in this way since an equal distribution of points was required for the median. When you for example consider -40 degrees to +60 degrees, and the robot is not straight in the row, the median is not the right distance to the side anymore. The -40 to +40 degrees range is the biggest range possible when looking equally backwards as forwards, because of limitations of the laser scanner mount plate. A sketch of this area can be seen in Figure 17.

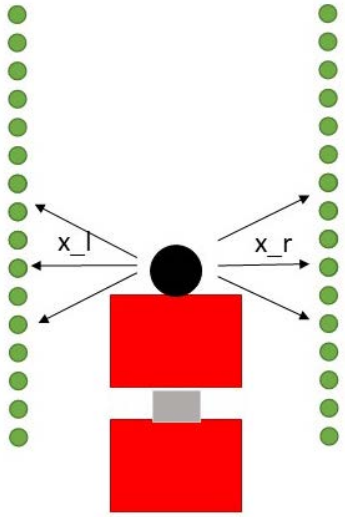


Figure 17 Sketch overview of in-row navigation

Next, the length of each of the laser scanner rays of interest was calculated using the cosine of the angle of the ray. This was put into a list, and all rays above a certain length were discarded. The cut-off point was set to 0.6 m, because at this distance there was no possible position for the robot to have seen these points in its current row. Since the rows are ideally at 37.5 cm away from both sides of the laser scanner, the size of the laser scanner is neglected here. Therefore, when the robot's orientation in the row is off, a margin of 22.5 cm to each side is built in to make corrections towards the middle of the row. For example: When the robot is completely at the left of the row, the row on the right will be at most 60 cm away, any laser beams going further than the 60 cm are going through the row and thus are not useful to determine the middle of the current row. Also, all rays with a length of 0 m were discarded as these would have probably been a miscalculation or a leaf brushing against the laser scanner.

From the remaining points the median was taken for each side of the robot because the median is less sensitive for outliers than the average. The medians were then subtracted from each other ($x_l - x_r$) in order to determine the deviation of the robot from the middle of the row (dev_x). This dev_x was then passed on as the distance the robot needed to travel to the right or left. This way the robot would always move slightly forward while steering clear of the plants to its right and left side.

Drive straight headland

In 'drive straight headland' the goal was to drive straight over the headland during the advanced navigation task, task 2. The idea here was to detect the rows and count them one by one. When the sum of the detected rows is equal to the required number, the robot would go turn back into a new row. The drive straight headland script did not only include the counting of the rows, it also includes keeping distance from the rows at the headland. This to make sure that the turn into the destined row would be successful.

The counting of the rows was done by visualising the laser scanner data and detecting the rows. To prevent the script of looking at the wrong side, all data that is on the other side of where the robot expects plants was discarded. Now the closest points at the headland were classified as maize plants. These were counted as rows and in that way the robot knew how many rows it has passed. However, the robot could not 'follow' the maize rows passing. Therefore, it was hard to distinguish between the maize rows and the eventual navigation was done on timing how long it takes to pass a certain number of maize rows.

To keep distance, the distance to the closest maize plants on the side where they were expected was determined. This was classified as distance to the rows. Hereafter, the deviation was determined at the headland. In the code was programmed that the distance to the maize rows had to be 0.8m. When the calculated distance was different from this 0.8m the steering angle was corrected and so the distance was corrected to 0.8m from the maize rows. A sketch of this navigation situation can be seen in Figure 18.

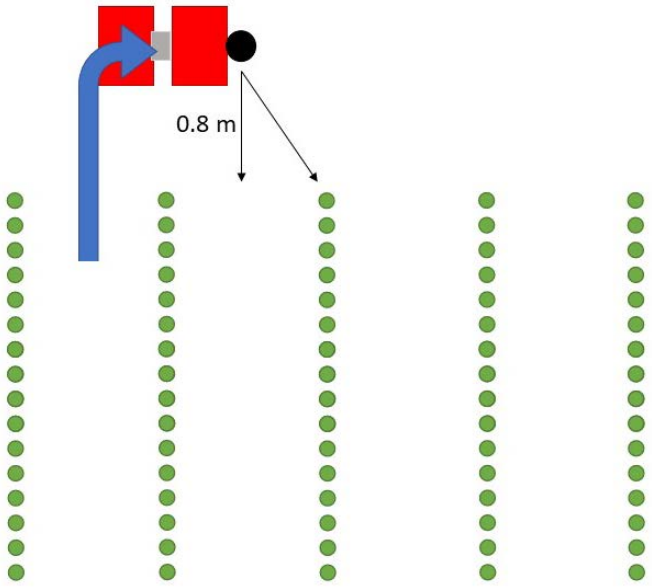


Figure 18 Sketch overview of Drive straight headland navigation

Turns

The turns were made using timers, because using odometry or IMU data did not work. The wheels were put in a certain angle and the timers were set to a specific time and then the motors were put on until the timers were finished and a turn was made.

After row

In the after-row state the robot should move a small distance forward. This is done by giving the robot a goal of 50 cm forward and checking with help of a timer if this goal is reached. By sending the same velocity to the wheels, the distance of 50 cm is empirically determined.

Results

Eventually the live navigation scripts were used to drive through the maize fields. In the script for in-row navigation, the deviation from the middle of the row was calculated and in that way it was passed on to the motor controllers. However, when the robot fully corrects its deviation, the steady state in the middle of the row is never reached. When the robot was put in the middle of the row at the beginning the first meter went well because the corrections were not very big. Whereas, it was visible from the beginning onwards that the deviations from the middle of the row were increasing as the robot passed through the row. This year, this was solved by dividing the deviation from the middle of the row by two. This meant that when x_l was 45 cm and x_r was 30 cm the robot would not steer towards a point 15 cm to the left but to a point 7.5 cm to the left. During testing, 14 out of 15 times the robot would not hit any plants in the straight rows of 15 metres. When testing in the curved rows, the robot had a lot more trouble with a curve to the right than one to the left. The curve to the left was just as successful as the straight row navigation. When the curve of the row was towards the right, the robot failed 3 out of 15 times. The reason for the failure with the curve towards the right is not known. These results were obtained with a low speed of the robot of 0.33 m/s. When increasing the speed, the deviation started to increase again, and the robot started hitting plants continuously, even when adjusting the division factor. Therefore, the speed was set back to its original value of 0.33 m/s.

Discussion

The live navigation only navigates the robot through the field by using laser scanner data. More sensors can be used to navigate the robot through the field. For example, the search for headland points was done constantly during the headland state. The rows in the Field Robot Event are at least 10 m long, so with the help of odometry data the search for headland points could be delayed until the robot had covered at least 9 metres. This could save time in the state and thus the robot may be able to travel faster through the rows. That said, this has the disadvantage that when a turn goes wrong and the robot is returned to its last correct position it first needs to travel the 9 metres all over again.

The turn did function. However, there is much to improve. Not a single sensor was used to make these turns. We did not manage to use sensors, such as laser data and odometry to make the turn. However, this is potentially a more robust solution than the current one. The current solution of using timers and putting wheels in certain angles is not robust and needs to be reconfigured under different circumstances.

For the ‘drive straight headland’ state one of the reasons that it did not function was that the robot was not able to distinguish the same row when it drove only a short distance. The current method consists of timing how far the robot drives over a certain time. Hereafter, the row width was determined and thus, the timing could be set for the number of rows. This works well but is far from robust.

Conclusion

Live navigation did function properly. The in-row navigation did guide the robot well through the rows and the turning at the headlands did function properly. When the robot reached the after-row state the navigation also functioned. However, this was programmed with hard code without using any sensor data. The drive straight headland state functions well but is far from robust due to its time-based distance measurement.

Recommendations

For the live navigation, it is recommended to make the robot drive faster during the 'InRow' state. Currently, only four maize rows of the contest field could be fully run through in 3 minutes. This is quite slow. The robot needs to work on its speed in order to remain competitive.

For the turns, it is recommended to let the sensors, wheel odometry and IMU, and the timers work together. In this case the robot would stop turning when the sensors give through the information that the robot has turned within a certain time slot.

For the 'drive straight headland' state, it is recommended to do more testing. When it works well enough, it may be useful to try to obtain the distance from the odometry or try to recognize and count the rows.

3. Vision

In this section, the detection system is discussed that was used for task 3 and task 4. The camera has to deal with several requirements. First of all is the event outside, so the camera has to deal with different weather circumstances. The second requirement is that is has to detect circles with a diameter of 12 mm. Furthermore, we need to have enough pictures of one patch, with more than one picture it is possible to compare the pictures to reduces errors. The scripts that were used to detect the golf tees were written in Python. For the scripts we used standard OpenCV functions.

Materials

The third task of the event was a vision-based task. During this task we had to recognise golf tees and their colour. For this vision part of the Field Robot, we used the following materials:

- Camera: Allied Vision Manta G-235C
- Lens: Kowa LM5JC10M
- Cable I/O: Hirose Type HR10A-10P-12S

The camera with lens is connected at the front of the robot, above and in front of the laser scanner. It is pointed directly at the ground, which makes all pixels mostly squared and equal in size. This isn't the case when camera is under a slope, then the shape of a pixel will be a trapezium and the pixels close to the robot will be smaller. With the camera directly pointed to the ground the circles are all equal and easier to detect. To run the camera in Python we used standard Vimba API's and Pymba. Pymba is a standard Python wrapper to use Allied Vision cameras in Python.

The used camera and lens has the following specifications:

Table 2 - camera specifications

Camera	***Allied Vision Manta G-235C***
Resolution	*1936 (H) x 1216 (V)*
Sensor (type)	*Sony IMX 174 (CMOS)*
Sensor size	*Type 1/1.2*
Pixel size	*5.86 µm × 5.86 µm*
Max. frame rate	*50.7 fps (full resolution)*

Table 3 - Lens specification

Lens	***Kowa LM5JC10M***
Focal length	*5 mm*
Lens type	*Fixed Focal Length*
Iris Range	*F 1.8 – 16*
Mount	*c-mount*

Methods

For task 3 and 4 it was needed to recognise patches with golf tees. For task 3 the robot needed to recognise blue and red tees together and for task 4 only red tees. Golf tees are round with a diameter of 12 mm. The colour and the shape are the properties we used to detect tees. Because for both tasks almost the same had to be recognised, the same code could be more or less used.

The tee recognition starts with retrieving an RGB image (matrix) from the camera, followed by splitting this image up in three: the red, blue, and green layers of the image. The three colour columns are then transformed to their relative form, by dividing them by the sum of the three.

To filter out most of the background, the formula for excessive green is transformed to a formula for excessive blue and red. An excessive filter is used to strengthen the colour relative to the background. We used this filter to reduce the number of pixels with a little bit of red in it. After this a lot of noise is in the image. The noise is probably coming from white pixels, white pixels have an RGB of: red 255, green 255 and blue 255. So when one filters to blue or red (or green), white pixels will also be displayed. To get rid of this, an opening and closing filter is used, with a kernel of respectively 8 and 5. These kernel sizes are set to new amounts found by testing. Different kind of soils and different weather circumstances lead to different sizes for open and closing kernel operations. What remains are two black images with white circles (Figure 19). These are the tees: one for the blue tees and one for the red tees.

Figure 19 Tees after filtering. Left is blue tees, right is red tees

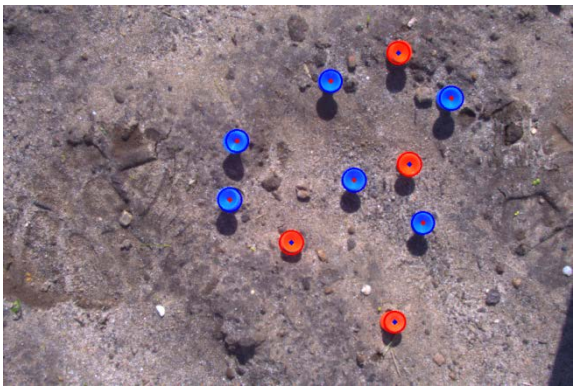


Figure 20 - original images with detected

To detect the tees' location (by detecting circles in the binary image), *houghcircles* was used. *Houghcircles* is a standard function from OpenCV. It was also possible to use a blob analysis, but a blob analysis took more time to detect the circles. Therefore, we used the *houghcircles* to process more pictures. In the *houghcircle*- function several parameters can be adjusted. For example: the minimum and maximum radius of the circles, the distance between circles, and how high the circularity is. After this, red or blue circles are formed around the tees and the location of the tees are saved. The result of this can be seen in Figure 20. The code will now return how many red and blue tees are detected.

Tees are not always detected. For example, if some blue tees are not detected, the programme could return that there are more red than blue, while this is not true. To overcome this problem, an adaptation is made in the code. When 9 or 10 tees are detected, it is almost certain the programme correctly returns whether there are more blue or red tees. One of them gets 2 points. When there are between 5 and 8 tees detected and of one colour more than 5 tees are detected, the colour with the most tees gets one point, the same is true when 11 tees are detected, the one with the most tees gets one point. In all other cases no points are given. When 4 tees or fewer are detected, 'number of tees' gets one point. For every camera frame, this loop is done, and all points are added. When 'number of tees' exceeds 5 the programme will return whether blue is larger or red.

For task 4 the recognition of blue tees was removed from the script. Other than that, the detection remains the same.

Results

For task 3, the team came fifth. During the task we were able to detect 4 patches correctly and give the corresponding signal correctly. 6 times the robot did not recognise the patch or recognised it incorrectly. The vision components of task 3 were also used for task 4. This time the vision part worked flawlessly; all patches were detected correctly.

Table 4: test results, vision test

Test	Veldtest 1	Veldtest 2
Aantal foto's totaal	10	10
Fotos 10/10 herkend	8	9
Fotos 9/10 herkend	2	1
Test	Veldtest 3	
Aantal patches in filmpje	3	
patches goed herkend	3	

Before the event, the detection was tested several times. Testing with the robot wasn't an option because it was needed for testing other tasks, so the detection was tested with connecting the camera to a laptop. Testing was done in the sun, with much light; but also in the shade, with little light. During testing this did not differ a lot, so when testing the detection script before the Event, nearly all tees were detected every single time, as can be seen in Table 4. Veldtest 1 and Veldtest 2 contain pictures whereas Veldtest 3 contains videos of 3 patches.

The speed of the detection script was measured several times. When testing the camera connected to the laptop, it was able to analyse 15 to 30 frames per second. With the camera on the robot, we did not measure how much frames per second were analysed, but we could see from the time between driving over the patch and the hearing of a sound that it had analysed less than 15 frames per second.

The problem for not recognizing some patches on the event was because the light intensity changed rapidly. This had the following result: with too much light intensity the pictures became totally white. With a too low intensity, the picture was too dark to recognize tees. A way to solve this was to set the camera to automatically adjust the exposure. This improved image quality, but because we made this adaptation right before the contest, there was not enough time to adjust the parameters.

Discussion

The camera settings used during the event are not optimal. It was hard to set fixed settings due to the weather changes. Every cloud is different, so we had to use the auto-exposure function of the camera. Auto-exposure sets the exposure time of the camera automatically. This means that if there is a lot of light, the camera will take quick pictures. If there is little light, this will take more time, so more light can fall on the photocell. The auto-exposure worked very well, but with the limited amount of time we had to test it, it was not perfect.

Another problem of the vision part is the usage of Python. We used Python with a standard wrapper, because there is no standard software for Python to use an Allied Vision camera. The available software is made for C++. Therefore, it may be better to use C++.

Due to the other processes running on the robot, the camera died for limited amount of time. The detection script worked flawlessly when used on its own. When other processes are running on the robot, which are always running to drive and navigate, the camera died. To

solve this problem, we made it start up right after it dies, but it takes time. This lead to missing frames during the detection, what eventually lead to no detection of a patch.

In the fourth task of the event all patches where detected well. Not all tees where collected which is probably because of the speed of the running script. It returned the patch detection too late. This resulted in a picking up zone right behind the tees. With the last patch, the detection was a bit faster, so it was able to collect 3 tees. The delay could be due to false positives obtained by the script.

However, before the event we tested a lot with the detection script and also with maize leaves in the camera frame. With correct parameter values, no tees would have been recognised on the plants' leaves.

When tuning the parameters, we also tested the time it took to analyse one frame. This gave differences every time the program was run, so this also gave different recognition times for task 4. When testing with the camera connected to a laptop, the script could reach 15 to 30 frames per second, but on the robot this speed couldn't be reached when testing on the event. When testing the device, it lowered sometimes in time, but often far too late. It is likely that the time of detection influenced the collecting.

Conclusion

It can be said that the vision module worked pretty well. During the event the system recognized 4 of the 9 patches, with the other patches the robot made some small mistakes. Some patches were detected, but the robot gave a wrong acoustic signal. Once, the robot turned on the headland and it recognized a maize plant as tees. During the tests, the vision worked very well, but during the event the brightness of the environment varied a lot and the camera died several times, which influenced the detection of tees.

Recommendations

There are some major points which have to be looked at if used next year: influence of light with the camera, calibration time for the parameters used in the detection script and how to open the AV camera.

When calibrating the parameters right before the task took place, we suffered quite a lot from influence by light, which was fluctuating a lot. In order to have not too much or too less light for the camera we set the camera on auto exposure. The negative part of this is that the parameters were a bit different every time the exposure was changed. This made that the tee recognition didn't work optimally. When in coming years something with camera vision is done, there should be looked at something to reduce the influence of fluctuating lighting conditions.

There wasn't enough time to calibrate the tee detection parameters with fluctuating light. Most of the calibration was done the day before the task. In this way the chosen parameter values were not good enough to mitigate the fluctuating lighting conditions. For upcoming years more testing has to be done under the same circumstances as when the camera is on the robot and as during the event.

In addition, there has to be worked on a script to open the camera in C++. C++ is better compatible with the AVcamera, and should be more reliable for consistent image grabbing.

4. Collecting device

Materials and methods & results

The approach from Engineering Design Methods by Nigel Cross (2008) was applied for this design of task 4. This systematic approach splits the main problem in parts which can be solved individually. Combining these sub solutions will lead to a solution for the main problem. The four steps that were performed from this method are:

- Establishing functions (function analysis)
- Setting requirements (brief of requirements)
- Generating alternatives (morphological chart)
- Evaluating alternatives (testing)

The three design steps involving a house of quality, stacked Venn and objective tree, were left out.

Function analysis

A function analysis of the to-be-designed system was made to express the overall function for the design in terms of the conversion of inputs into outputs, like a black box. This analysis of functions can be seen in

Table 5 Function analysis collecting device

Function number	***function***	***input***	***output***
1	*Lowering actuator*	*Signal to lower*	*Lowered actuator*
2	*Lifting actuator*	*Signal to lift*	*Lifted actuator*
3	*Grubbing tees*	*Tees in the soil*	*Grubbed tees*
4	*Putting tees in storage*	*Grubbed tees*	*Tees in the storage*
5	*Store tees in storage*	*Tees in the storage*	*Tees in the storage at the end of the task*

Brief of requirements

The brief of requirements is a method to specify the minimal performance of the to be designed actuator for task 4.

Index	*Description of requirement*	*Fixed/ Variable*	*Min*	*Max*	*Goal*	*Unit*
1	*Distance of the actuator to the patch of tees when lowering*	*Variable*	*0*	*10*	*5*	*cm*
2	*Distance driven further after the last tee is picked with lowered actuator*	*Variable*	*0*	*10*	*5*	*cm*
3	*Number of tees arrived till end of task*	*Variable*	*0*	*30*	*30*	*numbers (tee)*
4	*Number of tees arrived till end of row*	*Variable*	*0*	*10*	*10*	*numbers (tee)*
5	*Number of tees grubbed per patch*	*Variable*	*0*	*10*	*10*	*numbers (tee)*
6	*Number of tees stored in the storage per patch*	*Variable*	*0*	*10*	*10*	*numbers (tee)*
7	*Time needed for pick-up action*	*Variable*	*0*	*40*	*30*	*s*
8	*Speed of robot beyond picking up tees*	*Variable*	*0*	*1.3*	*.25*	*M/s*
10	*Quantity of soil collected with the tees*	*Variable*	*0*	*100*	*0*	*% (soil cm3/ tees cm3) *100*

The soil engaged weeding may only take place on the spot where the tees are placed. The actuator may only be lowered and lifted 10 cm max before and after the patch of tees. Otherwise punishment would take place if weeding occurs outside the zone. To build some margin 5 cm is chosen as goal.

For scoring points, you need to carry tees till the end of the row and till the end of the field (end of task). Then you get 2 points per tee arrived at the end of the row and 4 points per tee arrived at the end of the task. For maximization of the scoring, the max number of tees is chosen as goal.

To get those points, first successfully picking up of the tees is needed. Because the tees are put halfway in the soil, the tees first need to be grubbed. After that the tees need to get in the storage to make it till the end of the task. Here also max number of tees is chosen to score at best.

For better detection of the camera is chosen for the speed of 0.25 m/s. Then there is 2 minutes left for picking up 3 patches of tees. The max time for picking up a patch is 40 seconds. But to build in some margin 30 seconds is the goal if driving through the field takes longer, but not too fast because the change of success with picking up tees is bigger if you take the time.

Furthermore, you will get punish for soil you take till the end of the task together with your tees. Therefore, minimization of soil is desirable.

Morphological chart

The generation of a range of design solutions was done finding solutions for all key-functions and ordering them in a Morphological chart. This chart can be seen in Combinations of solutions resulted in 'total solutions'. Key-functions were selected from the function analysis must comprise enough functionality to meet all requirements.

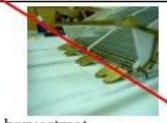
function									
Lowering actuator	 Linear actuator	 gear	 winch						
Lifting actuator	 Linear actuator	 gear	 winch						
Grubbing tees	 Acrobaat wheel	 brush	 hoe	 prong	 Stone collector	 rake	 Clamping band		
Putting tees in storage	 Pick-up	 conveyerbelt	 harvestmat	 brush	 scraper	 rake	 Combine header	 Stone collector	 Clamping band
Store tees in storage	 Container	 Container with holes	 prong	 Clamping band					

Figure 21 Morphological chart collecting device

Evaluation of alternatives

For the lowering and lifting actuator we had already 2 options available. Because the number of tees collected successfully by the collection device was not really depending on the way the device is lowered and lifted, only that it works, we choose to save money and pick an actuator that we already had. We had a gear on an electric motor which runs through a rack and a winch driven by an electric motor. The problem with the electric motors we had was that if we switch of the power, the motor almost lost all his resistance. With the gear and the rack this would have gave a problem. The device would have fall to the ground because of gravity. Witch the winch this wouldn't be the case, because it was driven by a worm wheel and a worm. The perpendicular transmission of the worm and worm wheel result in a resistance causing the collection device not to fall by gravity.

The best way of evaluating the alternatives is to build is simplistic version of the different collecting mechanisms (the last three functions). Therefore, we built two pins next to each other to simulate a prong, used a scraper to grub tees, and made a brush on a cordless drill to simulate a rotating brush. We counted the number of tees grubbed and the quantity of soil grubbed with the tees. The brush scored well on both points. Therefore, we chose to build a collecting device with a brush.

After the testing we made a 3D model in SolidWorks (Figure 22) and thereafter we started to build our collection device. When we made the 3D model we thought it would be useful to put our mechanism in front of the robot, but once we had physically built our detection device we decided to put in the middle of the robot, between the wheels. We had two reasons for that. First, if the collecting device would be placed in the front or in the back of the robot, it needs to fold away to a place above the maize. Otherwise the robot would become too long for good steering with low plant damage at the headlands. Also, the collection device would be in the field of view of the robot. Because folding away above the maize wouldn't make an easy and stable mechanism for lowering and lifting we choose to not do that. In the second place if we would have placed it in the front of the robot, backwards driving was very likely due to the delay of the recognition. Backwards driving costs time and will make collection harder if the wheels drive over the patch an extra time. Therefore, the actuator is placed in the middle for a stable mechanism and no extra extension on the robot. The placement of the device is shown in Figure 23.

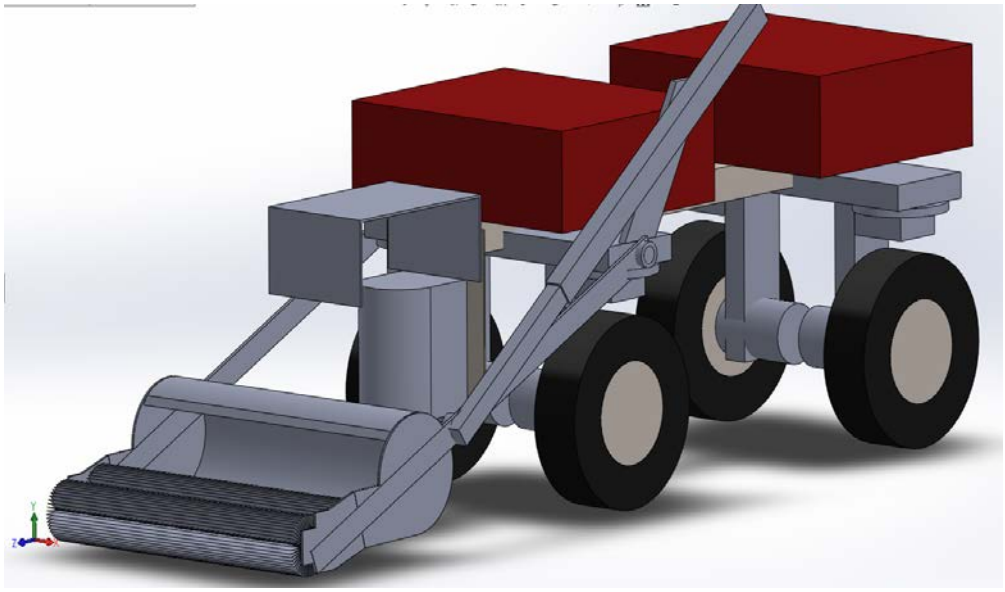


Figure 22 CAD drawing of the robot together with the collection device

Figure 23 picture of the mechanism in between the wheels

Results

The collecting device was tested by hands. We connected the electric motor of the brush with a switch to a battery to simulate collection by the robot. We made 3 patches of 10 tees and picked them with the collection device in our hand. We repeated it 3 times to explore if there are some specific situations where picking up went wrong, the results from these tests can be seen in Table 6.

One time the collecting device slid over a tee, because it was standing in a ditch. The other not successfully picked up tee was because the brush threw a tee back in the maize. But

overall, we had a success of 97.8%. The testing was done on dry sandy soil and about 50% of the volume collected consisted of dirt.

Table 6 Collection results of the collecting device

Repetition	Successfully picked up tees	Tees not picked up	Tees fall out
1	30	0	0
2	29	1	0
3	29	0	1

At the event 3 tees were picked up. The collecting device was not placed correctly in front of the patch and grabbed 3 tees and a lot of clay soil which didn't fall out the gaps made in the storage after testing on the sandy soil. So, the successful pickings were 10% and the soil was about 90 % of the volume in the storage.

Discussion

The collecting device worked well. The problem of the collecting device is the amount of dirt which was collected at the event. This depends on the kind of soil. When it is dry, most of the dirt will fall through the slots in the bucket. But not all the clods in a clayey soil, like the one at the event, falls through the slots.

The success rate of 10% at the event is also low. However, all the tees that were in the working range (3 tees) were picked up. The bad placement of the collecting device by the navigation, on top of the patches instead of in front of them, caused the low percentage of success.

Conclusion

The collecting device grubbed the tees as it should, only the storage concerning the specific clay conditions during the event was not optimal. The low score on success was caused by bad collaboration of the collecting device, the detection and the navigation.

Recommendations

For task 4, more time should be scheduled in the upcoming years. Most of the device was made in the last one or two weeks. As a result, some of the parts of the device were suboptimal. That's partly because delivery time of decent components is way longer than 2 weeks. Admittedly nothing broke during the contest, but some parts like the driveshaft could be better.

5. Discussion

In this chapter, the results at the event are discussed and the whole performance of the robot at different tasks is evaluated. With the robot as a complete system with the integrated modules, the cohesion between the modules is evaluated. Herewith the performance of the complete robot can be discussed.

Task 1 – "Basic Navigation"

The robot started out relatively slow but navigated through the row without hitting any plants. Before arriving at the headland, the robot stopped moving and wheel spin could be observed

in only one wheel (left rear). The suspected cause was a failure of the motor controllers. The team tried to move and restart the robot, but to no avail.

Points awarded: 18
Place: 11th

However, the communication between these modules seemed to work fine because the robot barely hit any maize plants. The error in the motor controllers was most likely caused by shocks from the bumpy surface, which caused hardware problems. With this the communication between the motor controller package and the state machine fell out. With no control of the motor controllers the robot was not able to drive the way it was meant to. This resulted in a robot which was not able to complete task 1.

Task 2 – "Advanced Navigation"

During this task, the in-row navigation worked perfectly. Since the headland turn was being done based on time, and not odometry, there were however a few issues. The number of rows skipped was incorrect.

Points awarded: 21
Place: 8th

During task 2, driving through the rows worked well. However, the 'advanced navigation' part caused some problems. The navigation scripts were not ready for the advanced navigation. Since the turns and drive straight on the headland of the robot were based on time, this requires quite much testing and aligning. Unfortunately, lack of time and tests made this a hard way to succeed the task. Due to the fact that the navigation script for the headland was not working the way it should, the integration with the state machine failed. If the odometry and navigation were combined better, the whole robot should have performed better than the way it did.

Task 3 – "Selective sensing"

Four of the nine patches were successfully detected. For the other patches, the robot either gave the wrong signal or did not detect anything at all. The main problem was that blue tees proved easier to detect. This was because of the lens not being optimally adjusted for the lighting conditions. When driving on the headland, there was also a false positive caused by interfering maize plants.

Points awarded: 24
Place: 5th

During task 3, navigation through the maize rows and the turns on the headland worked quite well. This was the first time at the Event that navigation worked properly**.**

Task 4 – "Soil-engaged weeding"

Despite having tested this task minutes before the start of the competition and it working perfectly, there were some issues. After detecting tees to be picked, the robot did not drive

back far enough to get to the patch. As a result, only three tees were picked up successfully. However, the detection worked flawlessly this time round.

Points:	28
Place:	2nd

During task 4, the navigation straight through the maize rows worked fine. But the integration of the state machine, vision, collecting device and navigation worked not fully the way it should have worked. The vision module detected the tees too late. That made the robot driving too far away from the tees. Because the navigation was time-based and not based on odometry or other positioning sources, there were problems with positioning the device on the right spot. As a result, the robot was not fully successful in carrying out this task.

4. Conclusion

Overall the BullsEye performed well. With an overall ranking of 6th out of 14, the BullsEye had a moderate performance. The integration of the modules worked in some cases fine. During the second day of the event, the navigation and state machine module worked well together, which makes that the robot had a good navigation in the maize rows and on the turns during task 3 and 4.

The integration of the odometry, vision and the collecting device module was not that successful. The odometry was completely ignored and not used in any task, since these results weren't reliable enough in time for the integration to still be tested. Together, the navigation, state machine, vison and the collecting device modules did not work fine due to little time for testing the complete integration. Problems in different scripts made that tuning was not perfectly. However, with nice results at the event (5th in task 3 and 2nd in task 4), the BullsEye performance at these tasks was good.

5. Recommendations

To let the BullsEye perform better, it is important to start earlier with testing in the field. With this, practical problems came earlier to the surface and it is easier to counter these problems. Together with earlier testing in the field, testing with a simulation environment could cause fewer problems. With a simulation environment, several things, like odometry and vision, could be tested without the physical robot. In this case, more problems could be tested at the same time.

An interface on the robot for selecting states may be helpful to help the robot after it fails to do one task during the event. An addition to this is to build in a back button, so that after the robot is stopped with the emergency stop, it can go back a state.

Also, a lot of scripts in the robot currently work at the maximum possible frequency, leading to these scripts having a rate of around 4000 Hz. These rates are not necessary for most programs and ask a lot of the computing power of the robot. In future work it is thus recommended to limit the rate of each script individually.

The way the state machine sends the command for the direction of turning during the 'Turn Out' and 'Turn In' state can be changed. One of the possibilities is to make a list and let it only take the next direction after the state 'In Row' is finished. This could possibly prevent the robot from updating the direction of turning too quickly.

With collecting tees for task 4, the navigation code should be improved. Right now, it detects a patch and drives back a for certain amount of time at a standard speed. Due to various detection times of the vision script you don't end at the right place before the patch with the collecting device. It would be better to detect the tees in the image with their x and y coordinates. With these coordinates and the transformation frame, the robot knows exactly where the patch is relative to the robot. Then the robot knows exactly where the tees are and can collect every patch correctly if it collaborates in a good way with odometry.

Overall, all time-dependent factors should be replaced by fixed distances or measurements, to avoid delays in the scripts influencing the result. Also, it is important that all used variables are defined in a parameter file. In doing this, mistakes with adjustments of variables in different scripts can be prevented.

6. References

Van den Aker, T., R. van Essen, A. Hulsman, T. Kaarsgaren, D. Otten, C. Schep. 2017. Report field Robot Event 2017. *Farm Technology Group.*

Fox, D., Burgard, W., Dellaert, F., & Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, *1999*(343-349), 2-2.

Fathoni, A. (2018, June 04). Setup and Configuration of the Navigation Stack on a Robot. Retrieved July 03, 2018, from http://wiki.ros.org/navigation/Tutorials/RobotSetup

Appeldoorn, R. (2018, June 29). Global_planner. Retrieved July 03, 2018, from http://wiki.ros.org/global_planner

Lamprianidis, N. (2018, January 10). Costmap_2d. Retrieved July 03, 2018, from http:// http://wiki.ros.org/costmap_2d

Salam, R. (2018, March 8). Base_local_planner. Retrieved July 03, 2018, from http://wiki.ros.org/base_local_planner

Ken, M. (2016, December 12). Move_base. Retrieved July 03, 2018, from http://wiki.ros.org/move_base

CARBONITE

Jacob Schupp[1], Lukas Locher[1*]

[1)] *Schülerforschungszentrum Südwürttemberg (SFZ), Standort Überlingen, Germany*

[*)] *Instructor and Supervisor*

1 Introduction

1.1 Institution

The aim of the German Schülerforschungszentrum Südwürttemberg (English: Students Research Centre) is to build a platform to promote pupils with deeper interests in mathematics, informatics, natural science and technique. Pupils can participate in different projects to proof and train their individual skills, with the aim to take part in different contests, as for example the field robot event.

1.2 Team members

The members of the "Carbonite-team" are pupils from the same school and they got to know each other when working at the SFZ location in Überlingen. Usually on Friday afternoon the SFZ pupils come together to fulfil challenging task like constructing, designing and programming a field robot.

1.3 Objective of the team

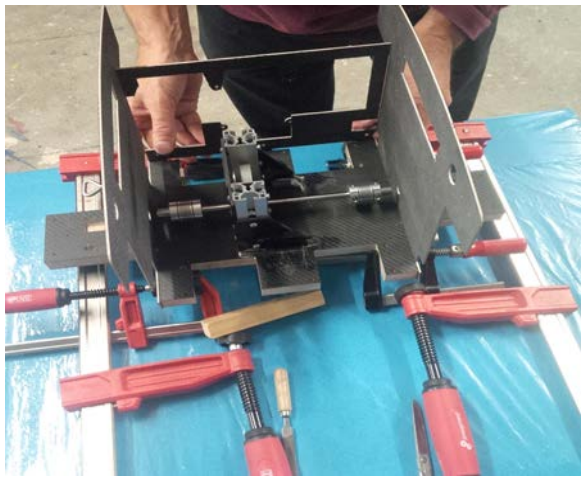
The main objective of the "Carbonite field robot team" is to enjoy constructing, designing and programming the robot while learning lots of things besides the school curriculum. The students make experiences in computer aided design and manufacturing, handling different sensors, data acquisition and evaluation, microcontroller and electronics, different communication protocols, drive control and computer vision concepts. They can bring in their own ideas when programming the robot and learn a lot of object oriented concepts and many aspects of team driven software development. State the objectives of the work and provide an adequate background, avoiding a detailed literature survey or a summary of the results.

2 Mechanics and Power

2.1 Chassis

After the SFZ Überlingen had participated successfully twice with their old robot „Soifakischtle", we decided to develop a new robot. There were several issues we wanted to change in our new design. The main problem was that the axles had too much slack and the servo-motors less power so the steering failed. The new robot has a length of about 90 cm and a width of just 44 cm (wheelbase). The whole body and "chassis" are milled of carbon-fibres with a thin layer of plywood. Afterwards the materials were glued with epoxy resin. Carbon-fibre is a very light and solid material, which is often used for aviation application. The

thickness of the outer walls is only 2 mm. For the ground plate we glued a 10 mm and 8 mm thick end balsa wood together and laminated it also with carbon-fibres. The weight of about 16 kg enables more hours of operation with a 16Ah LiPo accumulator. Furthermore, the light weights increase our off-road performance and facilitate the navigation also through muddy corn rows while bad weather conditions. Another feature of our design is various 3D printed parts which were used for different tasks like the fixture of sensors etc. To close the top of the robot chassis we designed a light roof. We manufacture this roof with 4 layers carbon-fibres. To glue the 4 layers of carbon-fibres we used vacuum technology. At all corners of the roof you can see a 3D printed component, these components are for holding and for quick remove of the roof.



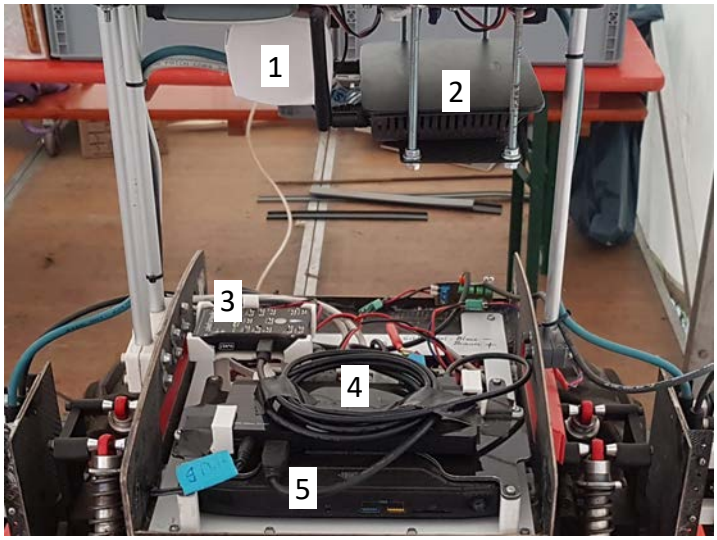
2.2 Drive Train

The motor we use is a brushless RC motor from the company Robitronic which is controlled by a VESC-Controller. The powertrain consists of motor a self-built gear block. The gear block goes to a shaft and this shaft makes the steering unit in the front and at the back turn. The robot has a 4 wheel drive. A special feature of our robot is that we can control every wheel individually. Because we have got an extra dynamixel-servo for every wheel.

3 Controller Architecture

3.1 Computer and other Hardware

The main control of the robot is done with an Intel NUC PC (Core i7, 32 GB-Ram) and an extern SDD (250 GB) made by Samsung. Most sensors and actuators are connected by USB. As operation system we installed Ubuntu Linux 16.04 and ROS-kinetic on the NUC. All sensor and actuator data is processed within the ROS-framework.



1. Siren
2. WiFi-Bridge
3. Gyro
4. USB-hub
5. Intel NUC

3.2 Software and strategy

Due to the short time range we could not test and develop a lot of new software for navigation. We mainly transfer some concepts from former SFZ-Teams used on former field robot events to the new robot.

3.3 Sensors

Laser Scanners

Navigation is done with a Sick Tim 571 laser scanner in the front and the rear. Data acquisition and processing of the scanners is done by the original ROS-driver

Gyro from a Pixhawk Flight Controller

Another SFZ-Team in Überlingen was dealing with a drone. They have experience with a Pixhawk flight controller. From this team we knew that the controller supports ROS communication with the mavros package. Therefore, it was quiet easy to get the sensor data of the angular velocity sensors of this flight controller.

Image processing

For image processing we use a Jai Go 5000 C USB 3.0 camera with a wide-angle-lens and a focal length of 1.8 mm. The basic configuration of the camera is done with the Common Vision Blox GeniCam Browser from Stemmer Imaging. For image data processing we wrote a ROS-Driver. The ROS driver uses the Common Vision Blox Framework (CVB) from Stemmer Imaging to acquire images from the cam. The driver delivers these images to the ROS image processing pipeline for further image processing.

The siren and the servos

We developed a PCB with a STM32F0 microcontroller to control the siren (for task 3 and 4) and the servos. These servos can turn pointers on top of our robot, which indicates the

selected state of the robot. For the communication with the STM32F0 controllers and the Intel NUC we use a USB2UART converter. A ROS-Node was developed to send commands to the STM32. Any other ROS-Node can publish topics for this node to control the weeding assembly. For the configuration of the STM32- microcontroller we use the CubeMX- software from ST-Micro. This java based software can generate projects for further C-programming within the Eclipse development environment (equipped with the right plugin, called “system workbench for stm32” from openstm32.org). You can download CubeMX and this plugin for free and both runs on Linux (and Windows).

4 Results and Conclusion of Machine Performance

The main task we have is that we had no chance to add components for special applications till now! The use of duct tape, cable ties is not satisfying for designers but quiet a standard in prototyping. So if we ever build a new robot we make sure to improve.

Another thing we gone work on is that we navigate with two laser scanners. These laser scanners are not designed to be used outdoor. They easily break due to the vibrations, which result problems from driving through open terrain. The costs of the laser scanners are very high when they break up every two years. In the future we want to navigate only with a camera system, which is on a long run cheaper due to their robust.

In our opinion, the use of carbon-fibre laminated plywood as composite material for the chassis was a good decision. It’s very light and strong! Carbon-fibre is not just good for the chassis but it is also suitable to design other components for our robot in the future.

The robot has got four servos for the steering so that we can control every single wheel. This has the advantage that we are not only able to drive a “U”-turn but also drive diagonally forward and backward. With this function we can drive our “Y-move”, which let us turn very fast at the end of a role. Y-Move means that we drive forwards diagonally out of a row and drive backwards diagonally into the next row.

To allow more than one team member to program at the same time we have a so called git repository which has the advantage that you can always access any file, distributed programming is possible. This saves time and avoids an annoying overlapping of the files. You can use the git-repository to create extra development to test the self-written software first of all.

5 References

The team was supported by

- Schülerforschungszentrum Südwürttemberg
- The mikromakro Project of the Baden- Württemberg foundation,
- Stemmer- Imaging
- Company Sick

THE GREAT CORNHOLIO

Thomas Ludemann[1], Steffen Hellermann[1], Andreas Linz[1*]

[1)] *University of Applied Sciences Osnabrück, Faculty of Engineering and Computer Science, Germany*

[*)] *Instructor and Supervisor*

1 Introduction

The following work is intended to describe the hardware and software used by students of the University of Applied Sciences Osnabrück for the 16th annual Field Robot Event. The paper begins with a general mechanical overview of the referred to entry, “The Great Cornholio”, followed by a more in-depth description of hardware used, including specifications, and an overview of the software algorithms used to accomplish general functionality and the applicable tasks for the competition. A conclusion then follows to summarize the findings of the design process.

2 Mechanics and Power

Mechanics

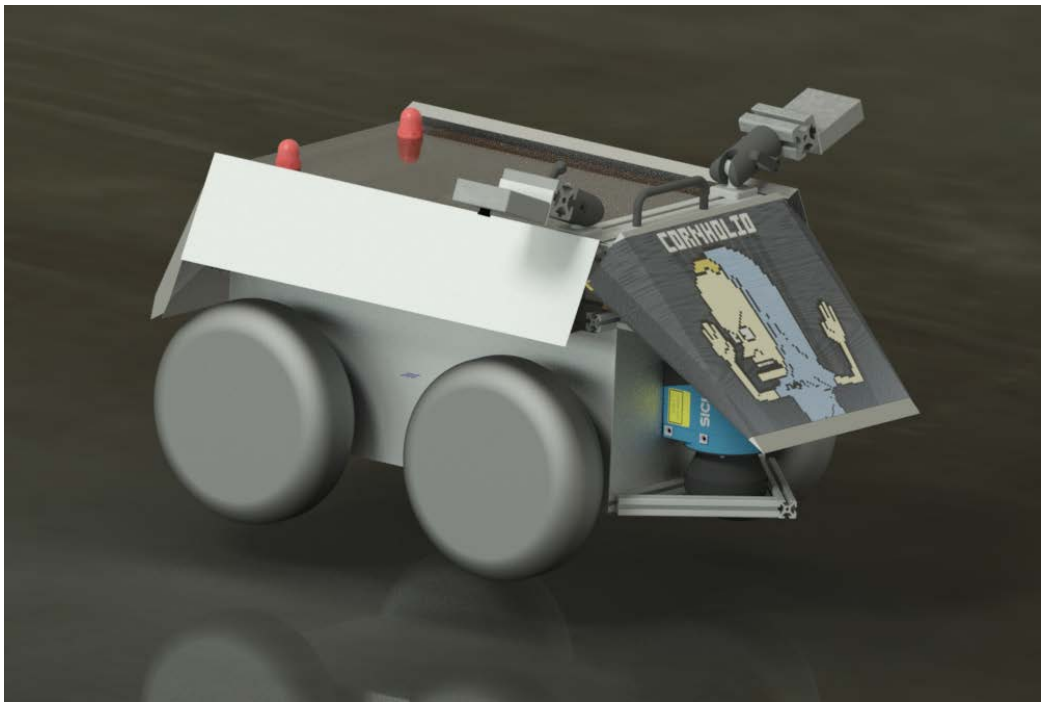


Figure 1: CAD draft of the Robot

''The Great Cornholio'', whose CAD draft can be seen in figure 1, has an overall size of 54x69x44cm. As it is typical for robots based on ''Volksbot'' - the robot-platform of the Fraunhofer Institute – Cornholio's case relies on item profiles, making it easy to mount sensors, motors, etc. The profiles and the cover panels are made of aluminium. As a light metal, the usage of aluminium saves weight so the robot will be able to move faster. When the top plate is removed, two claps provide fast access to the fuse-boards and cables.

Two Maxon motors with an epicyclic gearing power the robot with a torque of 15Nm. One wheel on each side is connected to the motor shaft by a claw coupling. The other wheel is then connected to the first by a chain gear. Separating the drive sections makes it possible to control each side of the robot independent of the other as it is typical for skid drive. An advantage of skid steering is the low turn-radius, thus optimizing manoeuvrability. This behaviour is useful in navigating through the narrow curved rows of the contest environment.

As already mentioned, the robot's case consists of item profiles. Item offers a special ball joint fitting in their profile system that we used to mount our camera. It can be slided on a horizontal line using the clamp lever to change the camera's roll-pitch-yaw angle. This provides great flexibility for the image processing sensor's field-of-view. The camera can be shifted simply by loosening the lever providing flexibility and adaptive capabilities in a changing environment (e.g. from the laboratory to field conditions) within seconds.

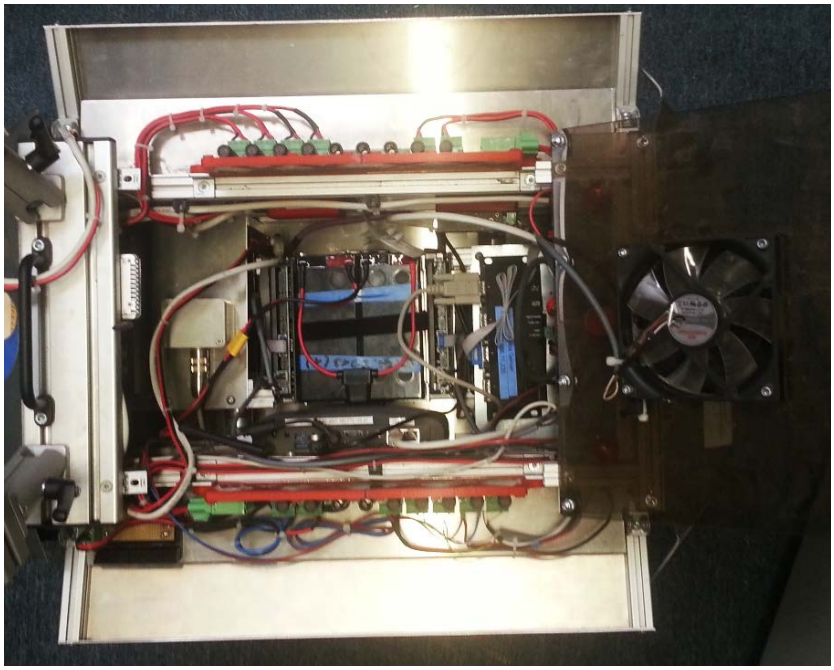


Figure 2: Inner assembly of the robot

Figure 2 shows the inner assembly of Cornholio. The robot is accessible through a PMMA lid, which is translucent. In the lid itself a PC-fan is embedded. The fan creates a positive pressure inside the robot to keep particulate material out of it.

The inside of the robot consists of the actuating elements, controls, a sensor and the power supply system. Right below the lid on the left and right side are the fuse-boards. Below the

fan, the device server is fixed with adhesive hook-and-loop tape, which is connected to an aluminium plate. Just below the plate one of the motors is mounted. If there are any problems with the motor, the plate can be removed. Another plate is mounted over the second motor, which is used to fasten the IMU. The motor-controllers are positioned right next to the motors. The battery is stationed in the middle right between the two motor-controllers. They are held in place by aluminium brackets and plates which give a lateral bearing of the battery. This allows a fast access and replacement of the battery. The PC is placed on the left side right next to the battery. It is attached to an extra plate, which is mounted on the frame of the robot. The Ethernet-switches are placed in the front and rear of the robot. They are mounted in an upward position, making the status LEDs observable.

Power supply

The power supply is based on two 12V lead batteries running in series to provide 24 V DC. They are directly connected to a power supply board that provides three fuse secured voltage levels - 5V, 12V and 24V. The 5V and 12V voltage levels are realized by board intern step-down converters and have a maximum current of 5A. The 24V level is able to provide 20A without conversion. Additionally, there is a buck converter connected to the 24V port of the power supply board that provides additional current to the 12V output port.

Fuse Board

The fuse board connects all electrical components of the robot with their corresponding operating voltages and protects them from exceeding voltage specifications as well. The fuse board supports three different voltages - 5V, 12V and 24V. Every route is fused separately with easily accessible fuse holders for easy exchange.

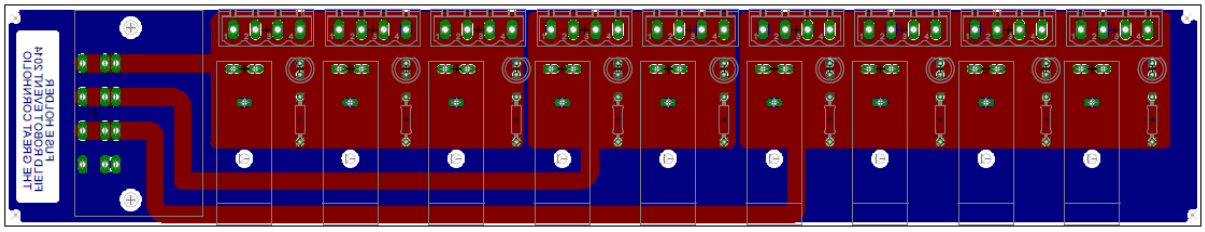


Figure 3: PCB-design of the fuse boards

The proper functionality of each fuse is displayed by a red LED. A working fuse bypasses the LED. The bottom side of the fuse board is moulded with epoxy resin. The advantage of this layer is the higher stability and contamination protection of the entire board. The electrical parts are connected with this board through a four-pin connector. This allows a hard-coded pin combination to prevent the interchange of operating voltages.

Figure 4: Fuse board

3 Controller Architecture

3.1 Computer and other Hardware

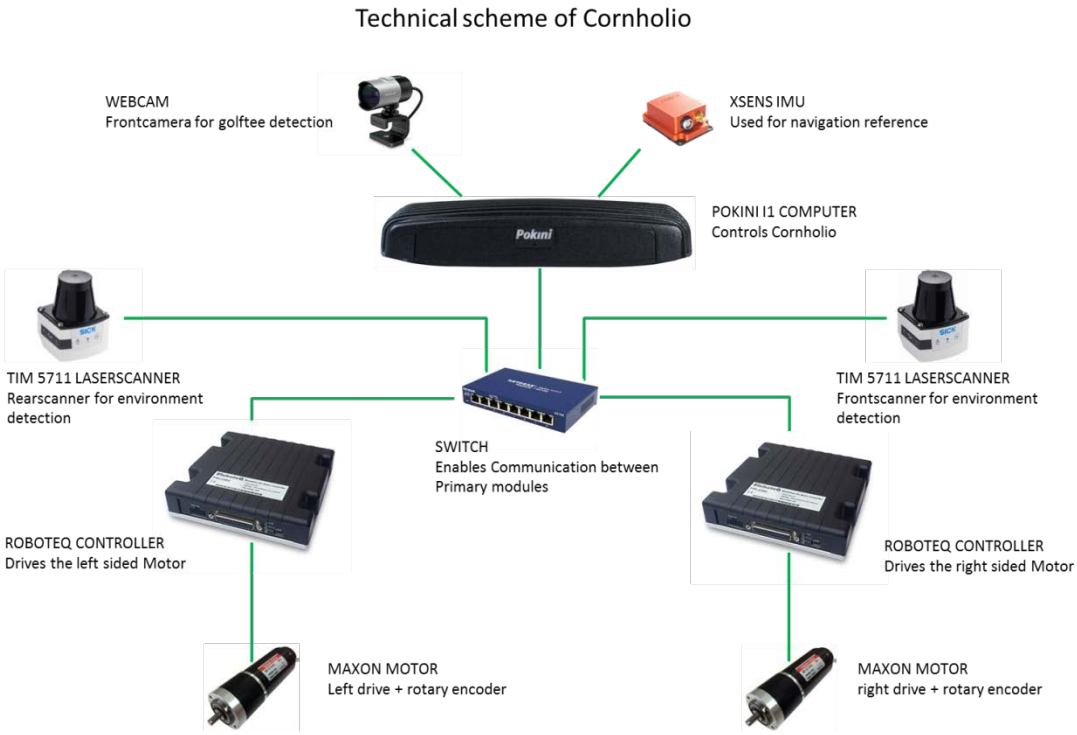


Figure 5: Primary hardware

(images may differ from the actual components, sources see below)

Figure 6 shows the important components of Cornholio. The components will be descriped below.

Computer:

- Model: Pokini-i
- Processor: Core i7-3517UE
- RAM: 8GB DDR3 SO-DIMM
- HDD: 120GB SSD
- Bluetooth integrated
- Passive cooling

Figure 6: Pokini i PC
Source: http://www.pokini.de/images/banners/pokini-i/1_pokini-i-composing.png

The computer is used to control the whole robot. It is connected to all devices by Ethernet.

Motor:

- Model: Maxon RE 40
- Nominal voltage: 24 V
- Nominal speed: 6930 rpm
- Nominal torque: 170 mNm
- Nominal current: 5.77 A
- Stall torque: 2280 mNm
- Stall current: 75.7 A
- Max. efficiency: 91 %

Figure 7: Maxon motor

Source: http://www.maxonmotor.com/medias/sys_master/8797180919838/RE-40-40-GB-150W-2WE-mTerminal-Detail.jpg

Motor controller:

- Model: Maxon Motor EPOS2 70/10
- supply voltage VCC: 11…70 VDC
- Max. output voltage : 0.9 • VCC
- Max. output current Imax (<1sec): 25 A
- Continuous output current I_{cont}: 10 A
- Switching frequency: 50 kHz
- Max. efficiency: 94%
- Max. speed: 100 000 rpm

Figure 8: Motor controller

Source http://www.maxonmotor.com/medias/sys_master/root/8797301768222/PRODUKTBILD-EPOS-2-70-10-375711-Detail.jpg

For the motor-control there are two integrated motor-controllers used to manage the speed and position of the wheels.

Display:

- Model: Electronic Assembly EA KIT129J-6LWTP

- Supply voltage: 5 VDC
- Resolution: 128x64
- Integrated touch panel
- Programmable with PC
- Connected by RS-232

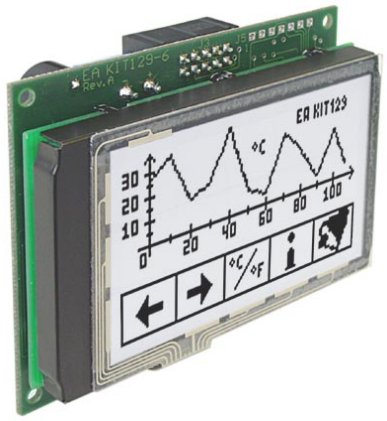


Figure 9: Touch display
Soure: http://www.lcd-module.de/pdf/grafik/kit129-6.pdf

The display with integrated touch panel generates a graphical user interface used to control the robot's functions.

Device server:

- Model: EX-6034
- Supply voltage: 5 VDC
- Chip-Set: Samsung S3C4510B
- Data transfer rate Serial: 50 Baud up to 115.2KBaud
- Data transfer rate Ethernet: 10/100Mbps

Figure 10: Device server
Source: http://www.exsys.de/media/images/org/EX-6034.jpg

The device server builds a connection for all serial-connected devices to the Ethernet.

Digital I/O converter:

- Model: EX-6011
- supply voltage: 5 VDC
- Digital Input Lines: 4, CMOS/TTL Compatible
- Digital Output Lines: 4, 2 non-inverted and 2 inverted
- Data transfer rate Ethernet: 10/100Mbps

Figure 11: Digital I/O converter
Sourcehttp://www.exsys.de/media/images/org/EX-6011.jpg

The digital I/O converter is used to switch loads controlled by Ethernet messages.

3.2 Software and strategy

In order to add modularity to our system, every hardware part of the robot communicates directly or indirectly over TCP/IP protocols. This set up allows to access, test and control the system or any part of it from any computer over a simple wireless router.

The control software

The core of the software is based on the Robot Operating System (ROS) and makes use of the built-in state-machine functionality to perform the main control procedures. The software initially starts the state-machine server and decides which action is to be carried out depending on the present task and determination of the robot's state. Each action defines a set of different parameters and instructions to be carried out in order to reach a specified goal. Once the end of the action is reached, the action returns completion data and reverts control to the state-machine server. Examples of actions are row navigation, cease motor action and so on.

The navigation algorithm

In task one to four our navigation was the same. The employed navigation algorithm has proven reliable despite, or thanks to, its apparent simplicity. The navigation algorithm reduces the space in front of the robot into an occupancy matrix of four rows and four columns. The exact number and dimensions of the cells may change during operation to suit the current environment and task at hand. When the ranges of the laser scanner exceed a given threshold within a cell, the cell is labeled as unavailable and the robot will alter its course accordingly.

Since the last event our robot performs a U-turn in the headland, thanks to the reliability of our IMU Sensor. The outer wheels are getting more torque from the motor than the inner wheels while turning into the new row.

Row Counting in the headlands:

In previous events we experienced difficulties to approach a row over a long distance. We decided to implement a new technique to count the rows until the distance is driven. It makes the operation much more reliable. We used two laser scanners, one at the rear and one at the front of the robot. With the combined data from the scanners, our system is able to build a simple map which only consists the rows. With a rear and a front check the system makes sure to turn our robot into the desired row. Due to this improvement we avoided difficulties with tolerances of the individual row width.

Golf tee detection:

Image processing is done using the OpenCV (Open Source Computer Vision) library. Using the ros usb-cam node, our Intel-Realsense Camera publishes images into our controlling system. The software converts the original image into the HSV space. After converting, our program finds contours related to the shape of golf tees. With a deep learning algorithm the robot could do a probability check for every image. This means a combination of the quantity of fitting contours and a red/blue check in the HSV space. If the algorithm decides that a cluster of golf tee is in front of the robot an audio signal is released.

Picking up golf tees:

We built a brush out of zippers. As the robot detects a cluster of golf tees the brush is lowered onto the ground. A peripheral motor drives the brush to push the golf tees into a container.

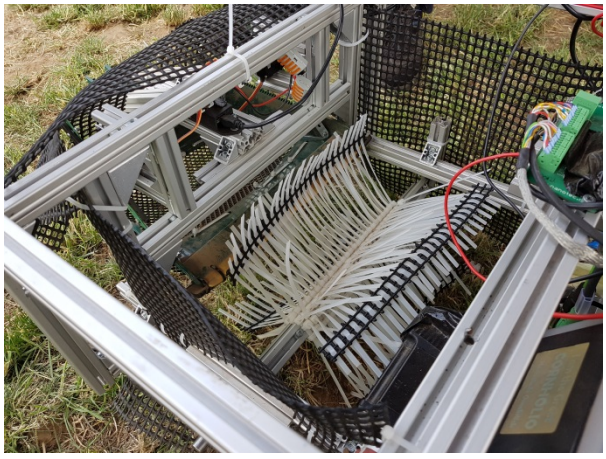


Figure 12: Brush rear mounted

Freestyle

We simulated a crash of our robot in the field. A new robot which was finished building at the event, towed the actual robot in the headlands. We used a self-built towing hook on the new

robot. It was also controlled by member of our team which used an XBOX controller and a monitor who displayed an image feed from the new robot.

Figure 13: New (yellow) robot is towing its predecessor into the headland

3.3 Sensors

Intel-Realsense D415
We used this camera for detecting the blue and red golf tees. It`s advantages are stable image capturing and a given set of software libraries.

Laserscanner

For the detection of plants and obstacles there is a SICK laser scanner TIM 5711 placed at the rear and front of the robot. The SICK laser scanners are using the “time of flight” method for measuring the distance between the reflecting object and the scanner. This sensor is capable of measuring the surrounding area in an arc of 270° about its vertical center axis.

IMU

The robot uses the Xsens IMU (Inertial Measurement Unit) which incorporates accelerometers, gyros and magnetometers. The sensors are used to determine the speed and position/orientation of the robot.

Incremental encoder

The motion of the motor itself is measured with an encoder which is integrated into the motor. The information of the encoder and the information from the IMU are used to increase the accuracy of the positioning.

4 Results and Discussion of Machine Performance

In preparation for the event we updated our OS linux to the lastet version. This has led to increased system stability. We also used new laser scanners and a new technique to count the rows in the headlands. With some optimizations our robot was very stable in navigating through the rows and in the headlands in every task.

For the golf tee detection we used a highly experimental development. Previous test showed proper functionality but during the event a connection loss occurred between components. Due to the loss there was no signalling for the jury when our robot had found a cluster of golf tees.

For picking up the golf tees in the last task our mechanical device was working as intended. Due to the compressed soil in the rows our brush had problems digging out the tees.

In our free style task we used our just in time finished new robot, everything was working as intended.

Recommendations

For further events our navigation is fine, we can also increase the speed of our robot. Even the variation of a "small" parameter can lead to new problems so it should be tested out carefully. Also new developments should follow a designing codex based on our experiences to avoid errors which are occurring (for the first time) during the event.

5 Conclusion

The weather conditions on the field were good. We put a lot of effort in improving the navigation in the rows and the headlands which payed off. There is still room for improvement regarding next year: Increase the driving speed and set up a designing codex to avoid unnecessary errors during the event.

All in all, this year our robot performed solidly but there is still room for using the maximum potential of the robot. Maybe our new robot will participate in further events. It is based on our actual one but it has been improved in every detail.

6 Web based References

Accessed on 14.09.2018

- Figure 5: Primary components:
  - WEBCAM: https://www.edv-buchversand.de/wacom/image.php?cnt=22&id=mz-2049
  - XSENS IMU: https://www.xsens.com/wp-content/uploads/2013/11/Mti-g-700.png
  - TIM 5711 LASERSCANNER: https://static.generation-robots.com/2932-large_default/tim351-sick-indooroutdoor-laserscanner-fuer-short-range-anwendungen.jpg
  - SWITCH: https://brain-images-ssl.cdn.dixons.com/6/2/00939526/u_00939526.jpg
  - ROBOTEQ CONTROLLER: https://www.roboteq.com/images/stories/virtuemart/product/fim2360-cover.jpg
  - MAXON MOTOR: https://www.maxonmotor.com.au/medias/sys_master/root/8811015503902/erin-4-web.jpg
- Figure 6: Pokini i PC: http://www.pokini.de/images/banners/pokini-i/1_pokini-i-composing.png
- Figure 7: Maxon motor: http://www.maxonmotor.com/medias/sys_master/8797180919838/RE-40-40-GB-150W-2WE-mTerminal-Detail.jpg
- Figure 8: Motor controller: http://www.maxonmotor.com/medias/sys_master/root/8797301768222/PRODUKTBILD-EPOS-2-70-10-375711-Detail.jpg

- Figure 9: Touch display:
  http://www.lcd-module.de/pdf/grafik/kit129-6.pdf

  Figure 10: Device server:

  http://www.exsys.de/media/images/org/EX-6034.jpg
- Figure 11: Digital I/O converter:
  http://www.exsys.de/media/images/org/EX-6011.jpg

Team sponsors:

- AMAZONEN-Werke H. Dreyer GmbH & Co. KG
- Sick AG
- Xsens
- iotec GmbH
- Electronic Assembly GmbH
- IBS
- MÄDLER

KAMARO - BETEIGEUZE

Henri Hornburg[1], Thomas Friedel[1], Erik Wustmann[1], Anton Schirg[1]

[1)] *Kamaro Engineering e.V. at Karlsruher Institut für Technologie, Germany*

1 Introduction

The Beteigeuze Robot was built to perform demanding tasks in the maize field to enhance productivity.

2 Mechanics and Power

The prototype robot measures 50 x 100 x 65 cm, weighing approximately 40kgs. All of its four wheels are driven, enabling it to drive with a maximum of 2 m/s. Both front and rear are steered, so it's turning radius reduces to 50 cm.

In order to fulfil the requirements of a robot driving in a field the drive chain was designed as a 4-Wheel-Drive with a single, central electric motor which can provide a torque up to 9Nm per wheel. The power transmission flows on two self-designed differentials in the front and in the back of the robot. Each axle mounting has its own suspension ensuring a smooth ride in rough terrain. Front and back axis can be steered independently therefore diagonal movements are also possible.

It is powered by two 6-cell LiPos with a total capacity of 10 Ah. The motors maximal power is 220 Watts. The robot is equipped with two SICK lidars, two Intel Realsense cameras and a fisheye camera. However, for this contest, only the front lidar and the fisheye camera were

used. Its own movement is measured by two absolute encoders and a BNO055 imu for acceleration and orientation of the robot.

3 Controller Architecture

The robot software is implemented on top of the ROS Software Stack. This means that the software is separated in so called nodes which solve small parts of the overall problem. There is a crawl row node keeping the robot in the middle between two rows of corn, a turn node which manages the switching between the rows. A state machine orchestrates the nodes to achieve the correct interplay for the given tasks.

3.1 Computer and other Hardware

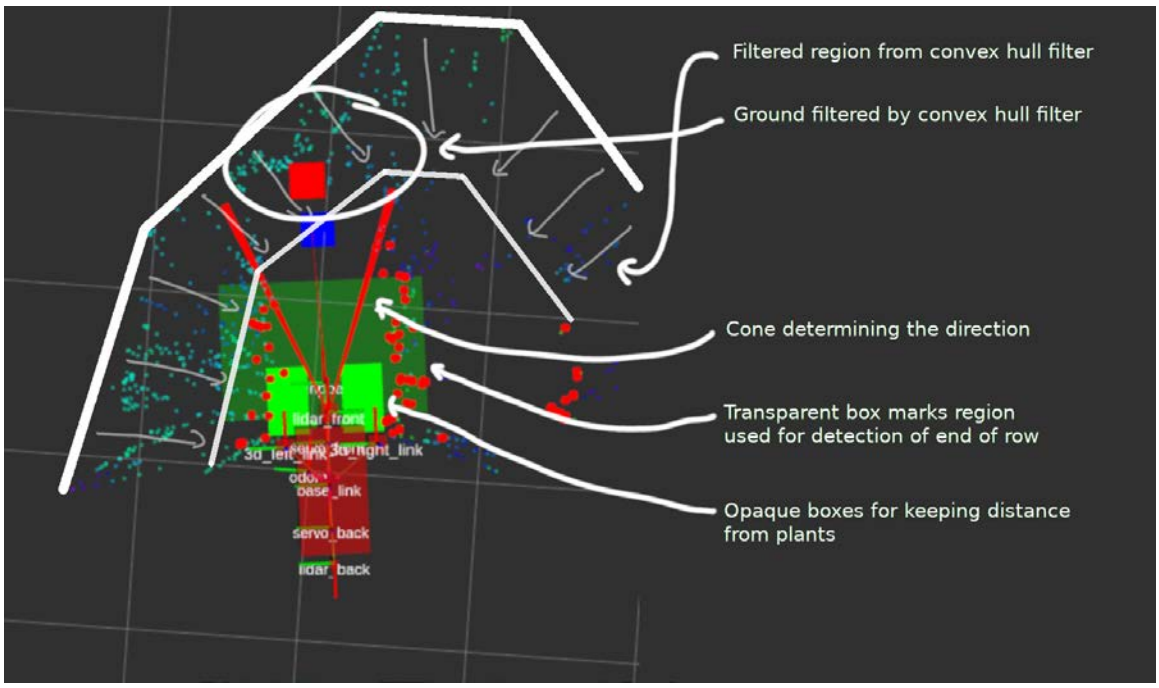
The central computing unit is a desktop computer mainboard in the mini-itx format with an Intel i5 quadcore processor and 8 Gb of ram. Low level hardware interaction and motor control via CAN-Bus is done on an ARM Cortex M4 Board. The microcontroller communicates with the computer via USB.

3.2 Software and strategy

3.2.1 Row Crawling

The only sensor used for traversing the row is the laser scanner.
The laser scans are filtered in the following ways:

- The robot itself is removed from the scan
- The convex hull of the laser points is calculated using Graham's Scan Algorithm. The hull is shrunk by a fixed amount and all points outside the shrunk hull are removed.



This is done to remove the ground from the laser scans in case the robot is tilted so that the laser is looking into the ground.

For driving through the rows we search for a free cone of 1.3m length in the filtered laser scan, beginning from the centre. This gives us a line in the direction of the row.

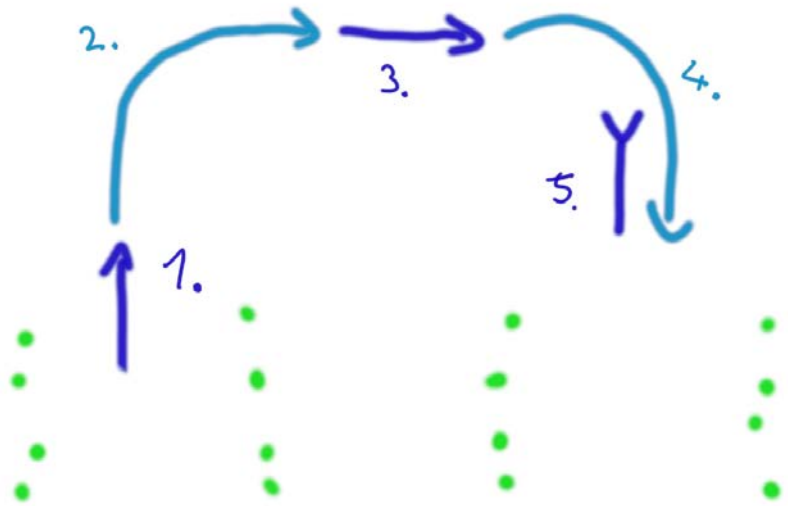
We shift this line to the left/right by an amount calculated from the ratio of the distances to the plants on the left/right. We clamp the distances to a maximum value to not be too confused by missing plants. The distances are calculated by growing a rectangle until more than three points are inside it. This helps to avoid taking shortcuts through the plants in curves.

We then use a PID controller to steer onto the point on this line that is 1m in front of the robot.

3.3 Row Turn

To detect the end of the row, we count the number of laser detections in a box in front of the robot. If the number is below a certain threshold for a long enough time the robot switches state from row-crawling to row-turn.

By fusing measured orientation from the BNO055 imu sensor with motor encoders we get a pretty good odometry. Our row-change algorithm relies heavily on that.



The row-change is realized by dividing the turn into five substates (see image):

1. Move Out: To get a safe distance from the crops we drive a bit forward.
2. Turn Out: We perform a 90° turn into the desired direction.

3. Move Along: We drive parallel to the fields edge for the calculated distance `(number_of_rows * avg_row_width) - (2 * turn_radius)`. Note that this distance can be negative (e.g. only one row), in this case the robot will drive backwards.

4. Turn In: We perform a 90° turn in the same direction to face the field again.

5. Move In: We aim for a certain distance perpendicular to the field edge compared to where we started to turn. This provides the row-crawler with enough space to do corrections on entry without damaging the first crop.

During all substates (except Turn Out/In) the robot compares its expected position with the position given by our fused odometry relative to the turns origin point. Any lateral differences are corrected by turning all wheels in the same direction (moving diagonally).

We also compare the expected row positions directly next to us with the laser scan during the Move-Along state, and correct the route accordingly. However, only for long turns (number of rows > 4) do these corrections have any considerable effect, the fused odometry is good enough in most cases.

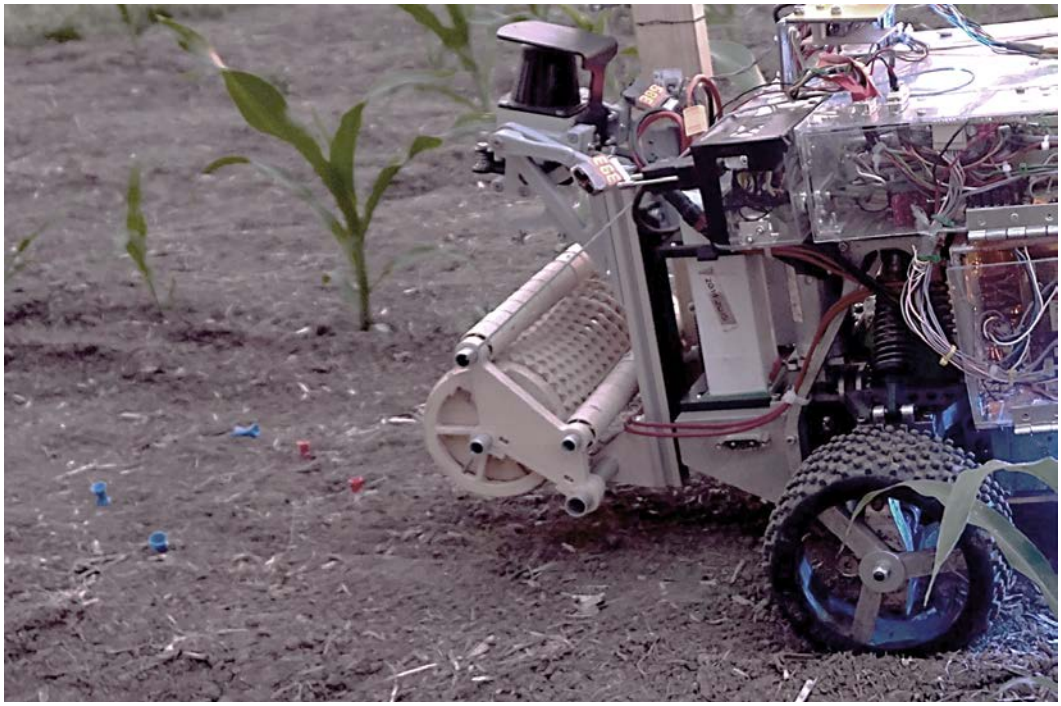
3.3.1 Golf Tee Detection

The golf tees are detected using a fisheye camera at the front of the robot. Red and blue pixels are extracted, filtered using erosion and dilation and converted into blobs. The blobs are

counted, some sanity checks are done and when the golf tees are detected for multiple frames the horn is sounded. This worked very well.

3.3.2 Golf Tee Removal

For golf tee removal we designed a 3d printed spiral that is supposed to extract the golf tees from the ground when pressure is applied using a spring mechanism. It is lifted from the ground when no golf tees are seen by the robot. This worked reasonably well, but only about 20% of the tees got stuck in the spiral, most fell down again.



4 Conclusion

We have not changed much compared to the last years. The big size of our robot makes it difficult to traverse the rows without damaging plants. On the other hand, it provides enough grip to pull extensions like the golf tee removal tool. Maybe next year we'll compete with our new robot which might provide some advantages concerning size and manoeuvrability.

DTU MAIZERUNNERS

Raffaele Barucci[1], Thomas Thuesen Enevoldsen[1], Carsten Dalacker[1],
Ramon Buchaca Tarragona[1], Alejandro García-Vaquero Velasco[1], Henning Si Høj[1,*],
Nils Axel Andersen[1,**]

[1)] *Technical University of Denmark (DTU), Automation and Control, Denmark*
[*)] *Supervisor* [**)] *Instructor*

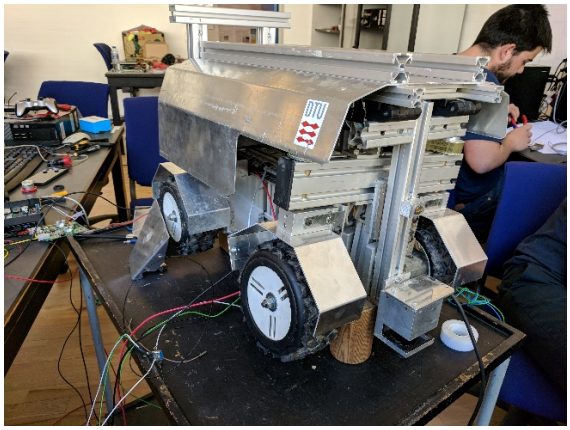
1 Introduction

The DTU Maizerunners is a newly formed team in 2018. We are a group of 5 master students enrolled in the course Advanced Autonomous Robots at the Technical University of Denmark (DTU). Participating in the Field Robot Event is considered the final project for our coursework in Advanced Autonomous Robots. The team consists of students with many different backgrounds from their bachelor's degrees, these being Electrical & Electronics Engineering, Mechatronics Engineering, Mechanical Engineering and Industrial Engineering.

Part of the work done as preparation for FRE2018 was electrically overhauling an existing robot from the University. This included completely replacing the main processing unit and the substitution of the old hardware interfaces with custom new and modular ones.

We would like to thank Henning Si Høj for a tremendous amount of support both administratively and technically. Much of the electrical rework of the machine, especially with the PCB design, would not have been possible without his experience, assistance and guidance. We would also like to thank the CLAAS foundation for supporting the electrical rework with 1500 EUR.

2 Mechanics and Power



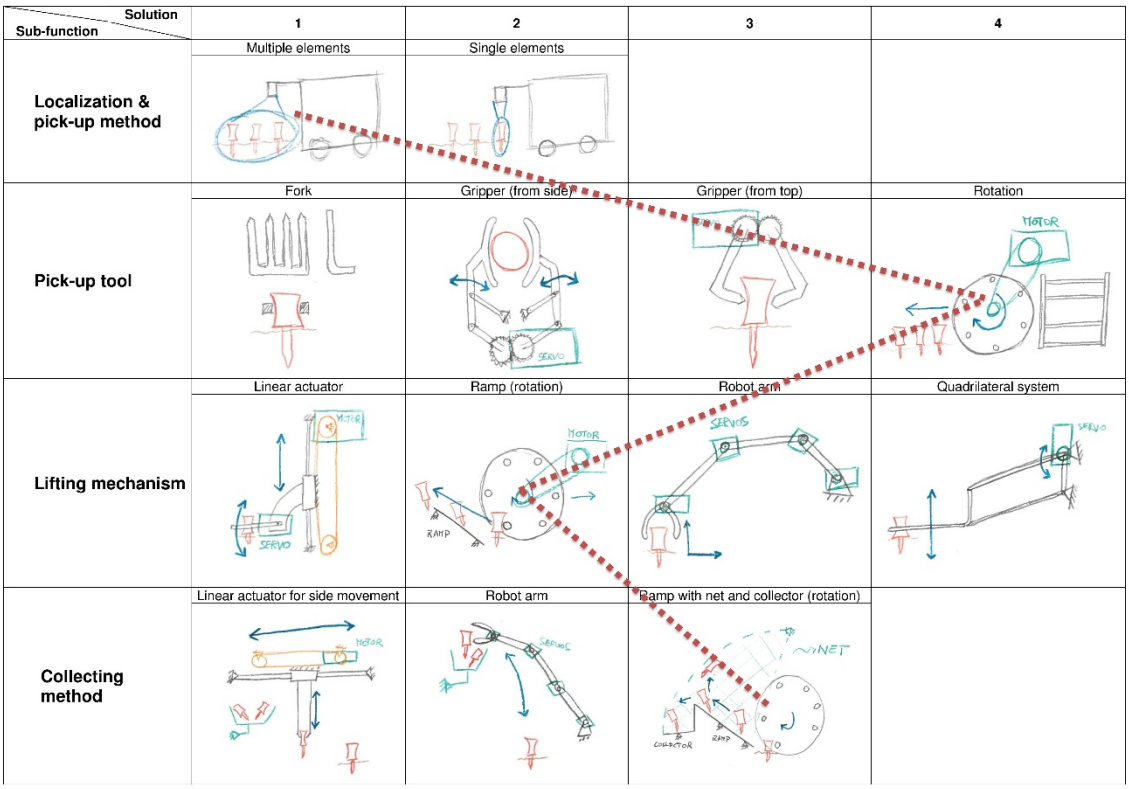
Robot Specifications			
W x L x H (cm):	33 x 55 x 44	Weight (kg):	~30kg
Commercial or prototype:	Prototype	Total no. of wheels / no. driven wheels:	4 / 2
Drivetrain concept / max. speed (m/s):	Rear-wheel drive, Ackerman steering, ~2m/s	Turning radius (cm):	~50cm
Battery type / capacity (Ah):	Lead acid / 14Ah	Total motor power (W):	200W
No. sensors internal / external: Sensor(s) type(s):	6/2 Gyroscope, wheel encoders, camera and lidar.		

The table above provides a mechanical overview of the robot. The main construction of the robot consists of aluminium profiles of various sizes, with separate metal containers for the electronics. To protect the internal components, various covers made of sheet metal cover the surfaces of the robot. The right figure above shows the robot before its visual upgrade, where all surfaces are covered in vinyl, this process also improved its resistance to different weather scenarios.

The robot consists of three components: the main chassis, the camera extension and the tee extraction device. A key element of the robot design, both mechanically and electrically, is the emphasis on modularity. To solve the four given tasks a combination of the above mentioned three components was utilized.

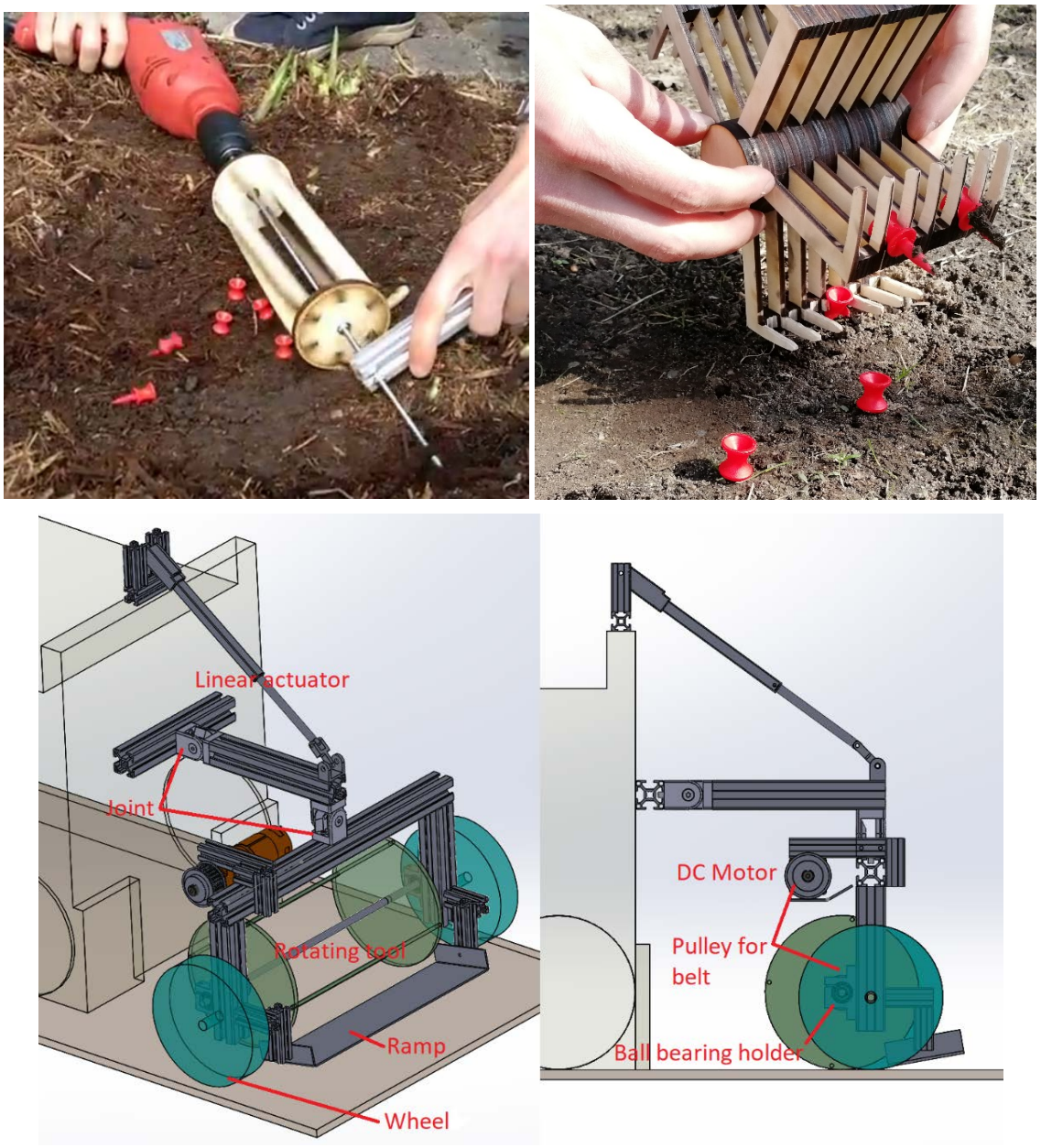
The main chassis of the robot without any of the beforementioned extensions, consists of the bare-minimum sensors for successful navigation, these being rear-wheel encoders and the front-facing lidar. The camera extension is a detachable arm, visible

on the left figure, that is required for our applied strategy in both task 3 and 4. The final component, the tee extraction device, is an electrical-mechanical extension designed specifically for the solution of task 4.



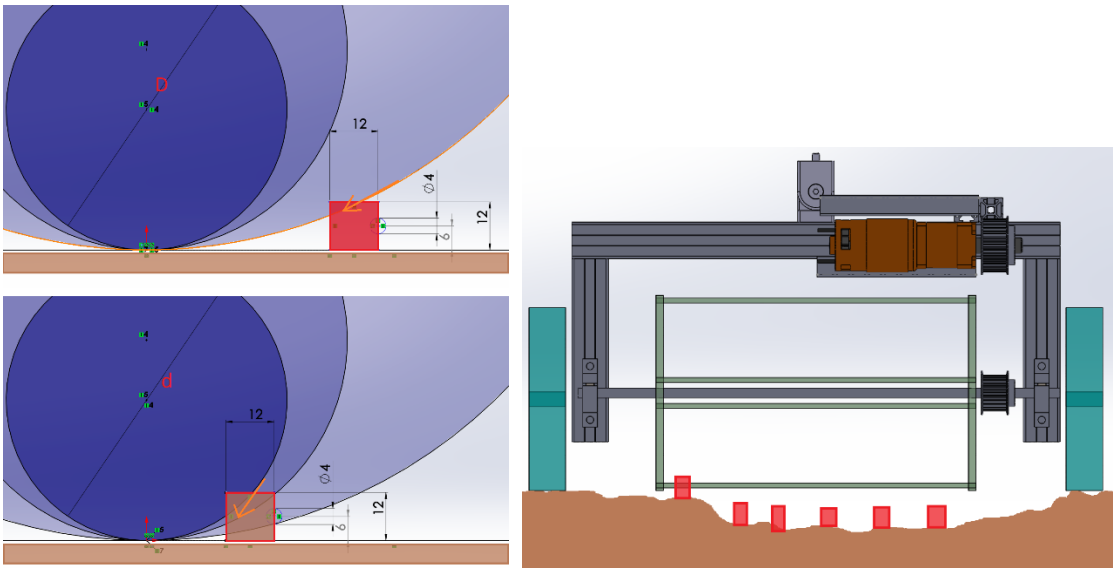
The figure above shows different ideas for the mechanical solution of task 4, this being the tee extractor. The left-hand vertical columns show the division of the different sub-tasks of task 4, where the up to 4 suggested methods are displayed. The red dotted line indicates the chosen path for solving the task of picking up the golf tees.

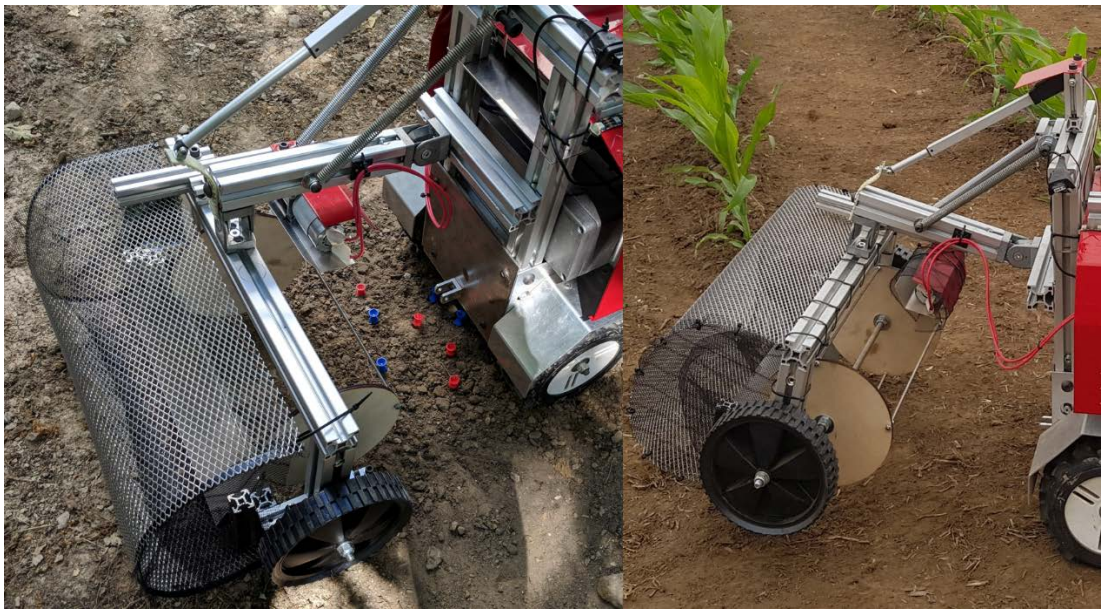
To arrive at the indicated path, each proposed method was rapidly prototyped and tested to evaluate its performance with respect to the other proposed methods.



The figure above shows the CAD model of the combined mechanism, from the different sub-mechanisms stated above, for tee extraction. The final implementation also has a net attached to catch the collected tees, this will be shown later in the section. To finalize the tee extraction mechanism, some possible problems were analysed and optimizations were made. Since the extraction procedure relied heavily on the spindle, different size variations of the wheels where analysed.

Due to the nature of the extraction process whilst using the spindle, the tees must be placed on somewhat even ground for the mechanism to properly engage with the found tees. Since the competition's rules specified that the patch in which the tees are located would be relatively even the extraction method was determined to be sufficient.





The above image shows the final implementation of the tee extractor with the before mentioned net installed. A detailed explanation of how the extractor interacts with the rest of the robot will be discussed later. Since the extractor was only required for the final task, it can with ease be taken on/off due to the modular mechanical and electrical design.

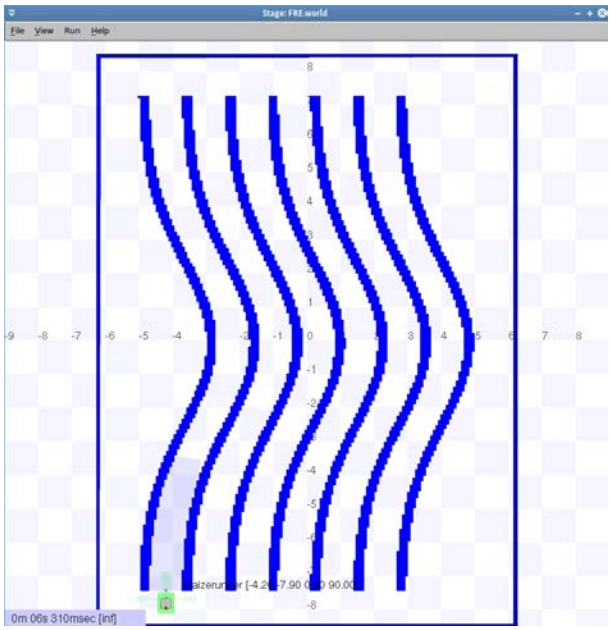
3 Controller Architecture

The core design principle for the robot is modularity. The two main components in the electrical design is an Intel NUC for the main processing unit and a Teensy3 as the main hardware interface.

3.1 Computer and other Hardware

The computing unit that is installed is the Intel NUC7i3BNH, where the operating system has been installed on an M.2 SSD with the possibility of adding an additional full form factor SSD. The operating system is chosen such that it is compatible with Mobotware, which in this case is Ubuntu 16.04 LTS. Mobotware is DTU's own plug-in-based software platform for robots, which was used for basic navigation and dealing will the given tasks of the competition. Mobotware uses the language SMR-CL which stands for Small Mobile Robot Control Language. It is a simple interpreted language; the main structure is based on states that the robot may take.

The figure below shows the simulator used, in order to validate the basic functionality of the robot, before doing testing on the real robot. This was important since the entire competition was based around autonomous navigation. Since that the crop row navigation was a fundemental part of the competition. The inner workings of the algorithms behind the navigational software will be explained in further detail later.



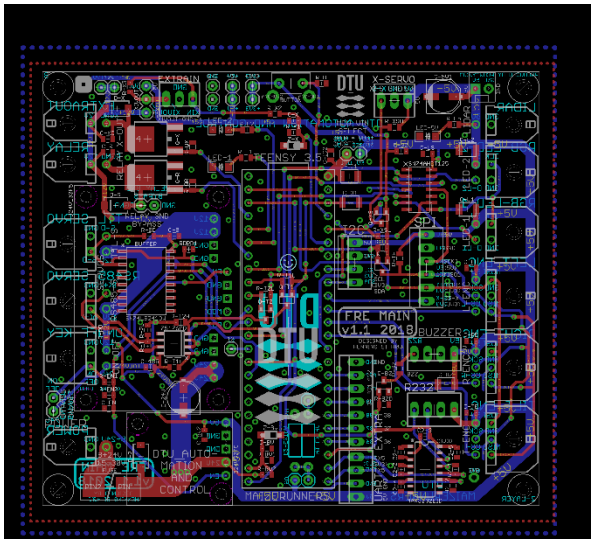
For the Teensy3, the interface between the NUC and the other hardware, a custom PCB was created and fitted with molex connectors. These connectors are attached to all the robot's components, making it easy to service, reconnect and replace components in the case of failure. The connectors ensured the modularity for the tee extractor, since the tee extractor was only used for the final task. The tee extracter is connected to the main robot using three connectors, one for the servo, spindle motor and power. The Teensy deals with all tasks that are not directly related to steering and driving.

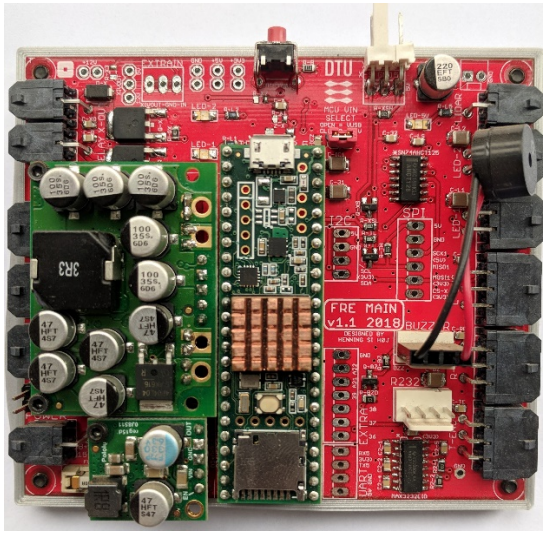
The controller for the rear motors is from the previous generation of the robot and is basically a black-box due to its age and lack of documentation. The steering is done using to USB hardware provided by the servo manufacturer (Dynamixel). For future generations, the PCB has been designed in such a way that the blackbox rear motor controller can be rebuilt and

included in the Teensy3 programme, the same for the front servos used for the steering. The PCB has the following features.

- Channels for the front left, right and tilt encoders
- Regulated 5V and 12V supplies (D24V22F5 and D24V150F12)
- 2 Channels for LED strips
- Battery voltage indication
- TTL Servo support (For one type dynamixel servo)
- RS485 Servo support (Another type dynamixel servo)
- I2C, UART, SPI breakouts
- Buzzer and LEDs for status indication
- Relay switching
- Extra analog in, digital out

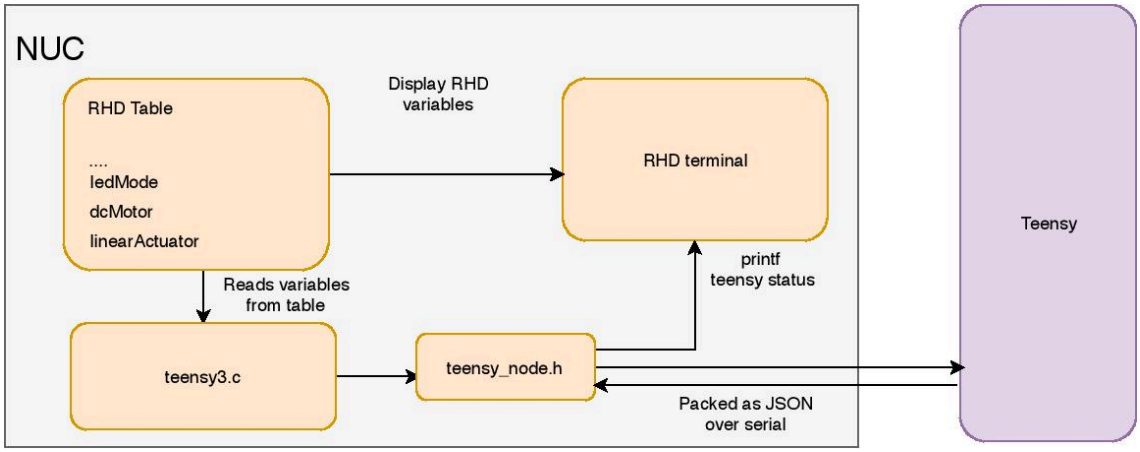
A key feature is the channels for the LED strips. The LEDs served as important visual feedback during the competition, for both the audience and judges, but also the team members. The LEDs were used inorder to indicate the state of the programme from a distance, using custom turning animations, colour detection animations and tee extraction animations.





The above figures show the custom PCB, both as an unpopulated schematic, but also as a populated PCB with all the female Molex connectors.

3.2 Software and strategy



One of the core concepts in Mobotware are the RHD variables, these variables are the ones which can be interacted with from the SMR-CL scripts. Above is a figure showing the communication procedure between the variables and the Teensy3, this implementation of the communication between the Teensy and NUC is unique for this robot.

The SMR-CL scripts can directly command the robot to drive and steer, since Mobotware contains written plug-ins for the rear motor controller and the steering servos.

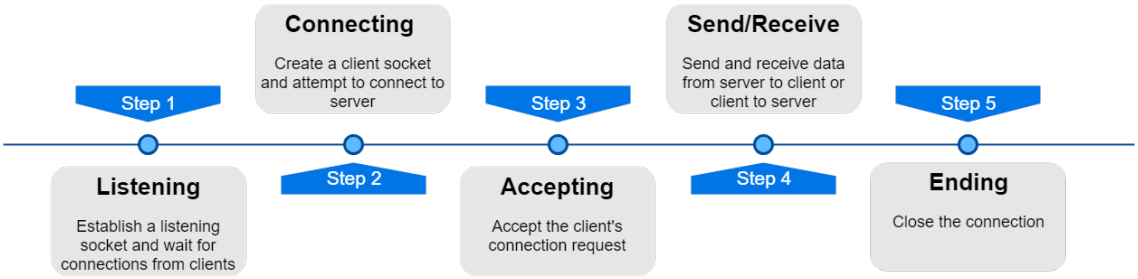
The most important part of the software for the competition was the autonomous navigation on field. The autonomous navigation was achieved by further iterating on an existing plug-in (*rosebot*) for Mobotware, that uses the lidar measurements to determine whether it should continue to follow the rows or if it has reached the headland and must begin turning.

The autonomous navigation was extensively tested using wooden planks as rows and fake plants, which can be seen above. During every available moment at the competition, the autonomous navigation system being tested. This was to ensure that such an important and fundamental part of the system was working. Since the overall performance for all tasks depended on being able to autonomously navigate between the rows.

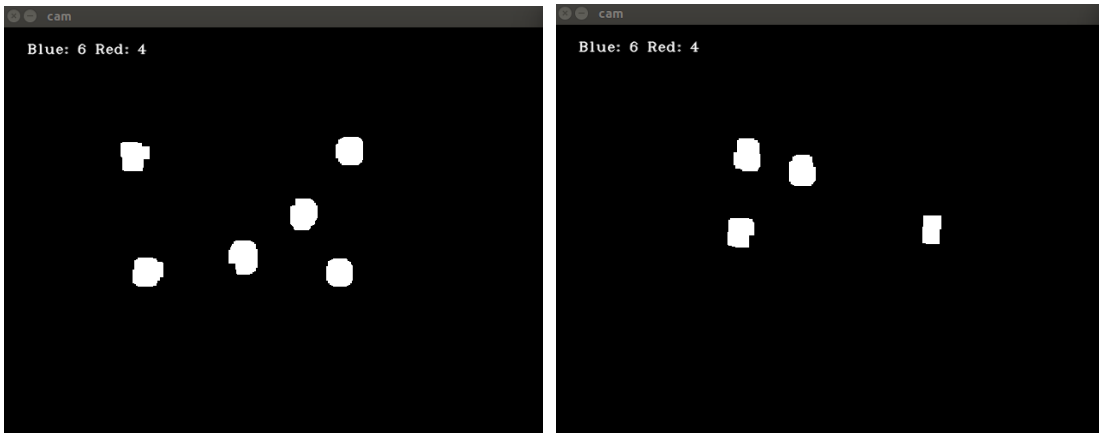
Task 1 was solved by using the information generated by the *rosebot* plugin to determine the current whereabouts of the robot. Before starting the script, the robot requires knowledge about the number of rows and the direction of the first turn. The robot then navigates using the environmental information from the lidar until it reaches the headland state. It will then continue to turn purely based on feedback from the rear wheel encoders (odometry) and re-enter the row once it has identified the opening using the lidar. This process is continued until it has navigated the stated number of rows.

Task 2 was solved as an advanced version of Task 1, where additional bookkeeping variables and filtering is introduced to accurately determine the state and next steps for the robot.

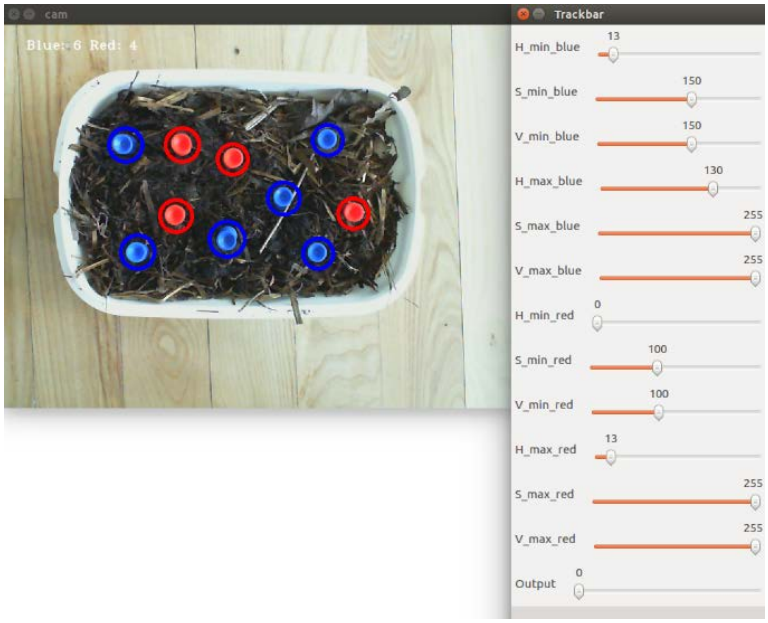
Task 3 was solved by combining the autonomous navigation with the camera extension for the robot, such that a computer vision algorithm could be applied. The processing of the golf tees was handled by a separate program, where using a socket server the information regarding the number of tees detected from each colour respectively was transmitted to the SMR-CL script.



The computer vision algorithm works as follows: First, the captured image is converted from the original BGR (blue, green, red) image to an HSV (hue, saturation, value) image. The reason for this is, that it is difficult to use BGR for the image analysis, because the B, G and R components are correlated with the amount of light shining on the object. HSV allows for a more intuitive calibration with the hue, saturation and value with respect to changing lighting conditions. To reduce noise from the captured image the OpenCV functions *erode* and *dilate* are called. Testing with different numbers of iterations showed that it is best to call both functions two times. The noise can be kept to a minimum and the abstraction of the tees is enough. After this procedure there are two binary images available where an OpenCV function is used to find the identified objects. By fitting an eclipse and checking the radius to a threshold, each contour in the binary images are either saved to an array and counted as a tee or discarded as noise.



To calibrate the threshold values for the three HSV parameters for the blue and red golf tees, trackbars on a graphical user interface was implemented for easy to use calibration. With these sliders the user can adjust the minimum and maximum values for hue, saturation and value for blue and red to the current lighting conditions.



The above image showcases the parameter tuning interface, which was very helpful since the tuning could be performed easily at the competition in case the lighting conditions changed due to a change in weather. Once the values are updated using the interface, the configuration file is loaded in such a way that it is accessible when running the autonomous routine.

Task 4 was solved by combining the features developed for task 3, the tee detection algorithm, alongside the mechanical tee extractor. Once the patch of tees had been identified by the computer vision algorithm, the robot was commanded to drive forward a given distance based on pure odometry, where then the extraction tool is lowered and engaged with the soil whilst the robot moves forward.

3.3. Sensors

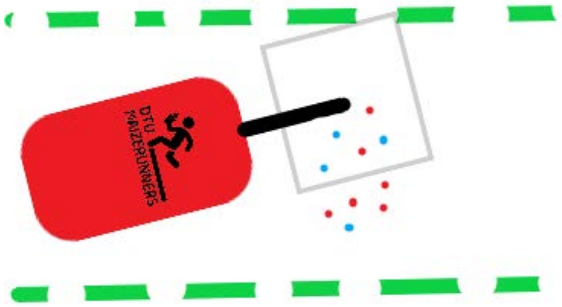
For all tasks the front facing lidar was used for the navigation, which was a crucial part of the system. Alongside the utilized lidar, rear wheel encoders served as feedback for the navigation between the rows when there was not enough feedback information available from the lidar. To solve the third and fourth task, the added camera enabled the use of OpenCV, which also was a crucial part to identify and deal with the golf tees in a timely manner.

4 Results and Discussion of Machine Performance

Task 1: The robot was able to cover a total distance of 74.3 meters in 3 minutes. Due to damaging two plants, resulting in 2 meters penalty, the final distance covered was 72.3 meters. This resulted in 2nd place for the task. Compared to the winning team Carbonite, who covered 135m before penalty and 129m after, our robot was significantly slower. There were a few reasons for this, the first one being that overall speed of the robot was limited by the lack of suspension and the uneven terrain. Going too fast on the uneven field would possibly cause damage to the internal components of the robot, due to the lack of any kind of suspension. Secondly, Carbonite was an omni-directional vehicle, meaning that it was not required for them to perform large radial turns. Though, the Ackerman steering provided much more stability during navigating the rows, which might explain the lower penalty points compared to the winning and other teams.

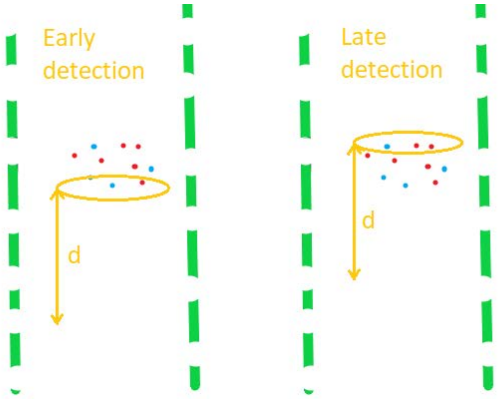
Task 2: The initial 90 degree turn at the end of the first row, the robot did not turn as much as expected and then the hard-coded distance (between rows) for the row search did not match with the expected one, when the robot once again turned 90 degrees in order to enter the following row, it was unable to find it correctly, and instead proceeded to drive on the maize plants between the two rows. At that time, the team pressed the stop button on the controller, pausing the mission, and attempted with a manual positional reset of the robot. Due to how the rules were for the competition, the robot had to be reset to the position where it was last functional, this meant that the robot had to be placed outside the entrance of the second row before the unsuccessful row entering took place. An unfortunate side effect of moving the robot back that position is that the mission plan had already progressed, resulting in the robot unable to move. This yielded a 9th place for task 2, since the overall distance covered was 14m, equivalent to the first row.

Task 3: The robot was able to detect eight of the nine weed patches. Seven of the eight detections were true positive, and one detection was a false positive. The two detection errors can be explained by looking at the position of the camera and the navigation of the robot.



The above figure shows the false positive detection. Even though the rows of maize were straight in this task, the robot changes its heading a little bit in the row due to uneven terrain or plants that bend inside the row because of wind. The webcam is mounted approximately 20 cm in front of the robot and looks straight down on the ground. The field of view of the camera is indicated by the grey square. In case of a sharp steering angle the camera does not see all golf tees and therefore enables the possibility of a false detection, which in the case of the above figure is detecting more blue than red tees. For the same reason the robot can miss an entire weed patch. The algorithm was setup in such a way that if the camera did not see at least three tees it does not signal a detection. This was to act sort of like a filter, since at times, depending on the lighting condition, the green leaves from the maize plants were detected as red tees. By having 7 true positives, 1 false positive, 1 false negative and without damaging any plants the overall points for this task were 42. This was enough to win 2nd place. The winning team of this task, Beteigeuze, had perfect overall detection and but with some plants damage. Though, due to the perfect detection they were able to get 48 points.

Task 4: At the competition, the golf tee pick-up tool was able to extract and collect 9 out of 30 golf tees. Furthermore, one tee was extracted but not collected. No penalty was given for the cleanliness of the tees, which means that very little soil was collected together with the tees. The result was below the expectations because the yield of the mechanism during the tests was approximately between 70 to 80%. The reason was that the during the competition a lot of soil was added to the rows of maize plants to adjust them, which resulted in a concave field. As explained in a previous figure during the design for the extractor, an uneven field is the worst-case scenario for the mechanism.



The above image shows two states of detection, early (left) and late detection (right) of the golf tees, in both cases the robot will move forward by a fixed distanced and activate the tool in different points which can result in excessive travelled distance. The excessive distance travelled outside of the golf tee patches was 0.65m which resulted in 0.65 penalty points. This distance depends on the accuracy of the golf tee detection. To improve robustness, a significant margin of distance was given between the point of detection and the point where the tool was lowered and put into operation (see figure 5.3). The use of a secondary rear mounted camera or a wider-angle camera would have been useful to solve this issue. Since the robot collected a significantly larger number of tees compared to the other teams, the overall points obtained for the task was 35.35, resulting in winning 1st place for task 4. The second-place team, Steketee Bullseye, obtained 9.54 points. One of the main contributing factors to the task win, was the overall robustness and repeatability of the solution. The solution had been extensively tested before and at the competition.

5 Conclusion

As mentioned in the previous section, the team managed to obtain a 2nd place in Basic Navigation, 2nd place in Selective Sensing and a 1st place in Soil-Engaged Weeding, which resulted in an overall 1st place making DTU Maizerunners the winner of the Field Robot Event 2018. A significant factor in the overall win was extensive testing, both before the competition but also at the competition. Compared to the other participating teams, our team tested by far the most. Every available minute the robot was actively being tested, modifying the task solutions to the new conditions and experiences obtained at the competition.

6 References

Danmarks Tekniske Universitet, Mobotware, Accessed 11/9/2018, http://www.aut.elektro.dtu.dk/research/aut_software/mobotware,

Danmarks Tekniske Universitet, SMR-CL, Accessed 11/9/2018, http://aut.elektro.dtu.dk/mobotware/doc/smrcl.pdf

Intel, NUC7iBNH technical specification, Accessed 11/9/2018, https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc7i3bnh.html

ERIC

Alex Fisher[1], Rhys Thomas[1], Megan Platt[1*]

[1)] *Harper Adams University, Engineering Department, United Kingdom*

[*)] *Instructor: Sam Wane, Supervisor: Dr Ianto Guy*

1 Introduction

The 16th Field Robot Event was held in Bernburg-Strenzfeld on the 12th to 14th June 2018, (Field Robot, 2018). The robotic event was run in conjunction with the DLG Feldtage, an annual agricultural show. The robotic event consisted of five individual tasks with varying difficulty:

- **Task 1: Basic Navigation:** The aim of this task was to navigate up a 75cm wide row, lined with maize plants, for 10 metres; turn in a 2m wide headland and then navigate down the next adjacent row. A total of 11 curved plant rows had to be completed in under 3 minutes.
- **Task 2: Advanced Navigation:** This task was similar to task 1 but a sequence of rows would be provided. For instance, down one row, left turn, miss 4 rows, down the 5th row and so on. The plant rows where similar to task 1 accept that they were straight rows.
- **Task 3: Selective Weeding:** The aim of this task was to demonstrate weed detection and identification whereby the robot would drive up and down rows similar to task 2, only this time with randomly placed weed patches. A total of 9 weed patches were present in just 3 rows. The weeds were represented by red and blue golf tees and the robot had to clearly identify which colour was in greater amount in the patch.
- **Task 4: Soil-Engaged Weeding:** The aim of this task was to identify the weed patches similar to task 3 accept that there was only 3 weed patches containing a single colour. The robot had to use a device to engage with the soil to remove the "weeds" but with minimal soil tillage.
- **Task 5: Free-Style:** A chance to show-off what the robot can really do. Creativity and fun is required for this task as well as an application-oriented performance. This task is unmarked.

The aim of this event is to demonstrate the abilities of sensing equipment and modern electronics to fulfil complicated tasks related to agricultural applications. The event therefore allows for autonomous behaviour to be explored in a reasonably safe environment, while showing the general public the future of agriculture.

This report discusses the mechanical elements of the robot, ERIC, (the vehicle itself and the robotic arm), as well as the sensing equipment and software used by the robot to operate

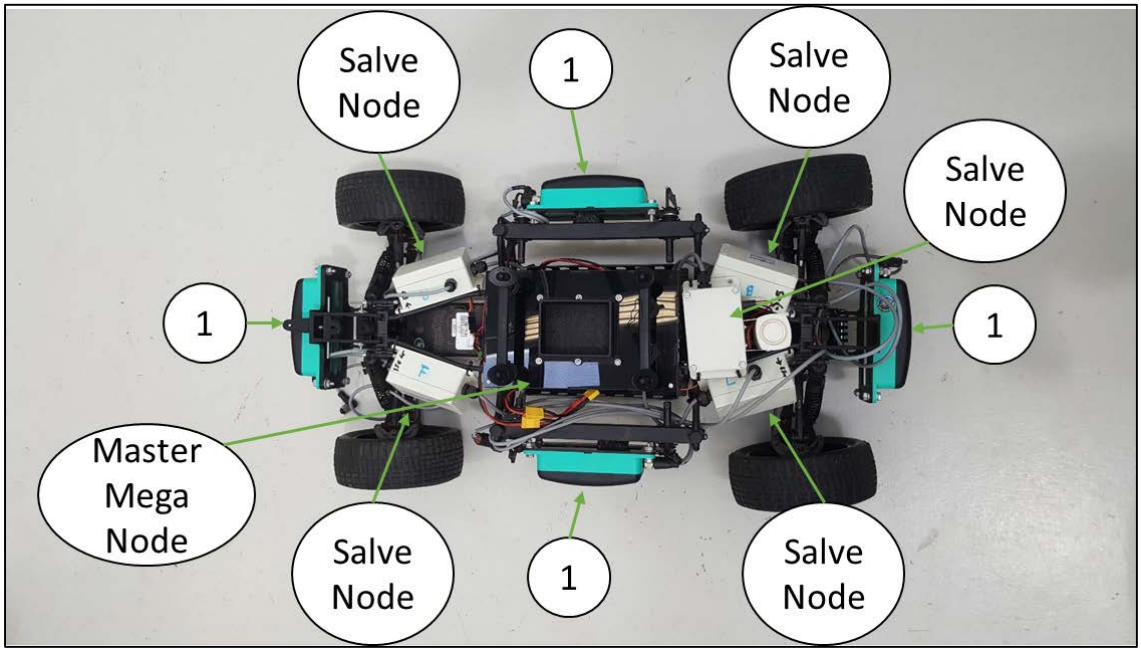
autonomously. It should be noted that the robot shares a lot of resemblance from the previous year's design, as the same hardware and vehicle was used. This year's focus has been to make the robot more efficient and faster with the team heavily focusing on the underlying architecture (coding) and the control strategy.

2 Vehicle and Implement

Throughout the design, a stripped down approach was taken in the interest of the given budget. Therefore, the robot adopted a functional appeal as it was designed for the task at hand and nothing more.

2.1 The Vehicle

The core of the robot was a ready-made radio controlled (RC) car, which provided a reliable and functional platform for the rest of the hardware to be mounted upon. Some modifications had been made previously to allow four-wheel steer and improved torque. Two 11.1V 3S 3500mAh batteries provided all of the power required; one supplied the high current components (drive motors, steering servos and robotic arm) whilst the other supplied the control and sensing components.



Source: Author's Own

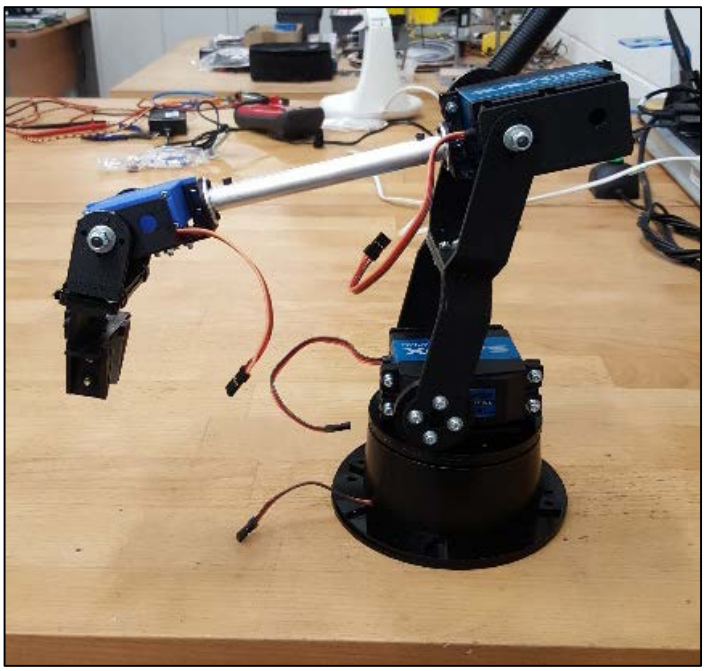
Figure 1: Whole Vehicle Layout

From Figure 1 the LIDAR sensors can be seen. Four of these were used, one on the front and back to guide the vehicle down the rows and two on the side to navigate in the headland. A digital imaging camera was mounted to the front (2) on the robotic arm which both detect weeds in task 3 and help navigate the arm in task 4. The main box on top of the vehicle housed the main microcontroller and the vehicle control node, with the individual sensing nodes

mounted within waterproof cases near their sensor. All parts added to the vehicle were designed, built and assembled to be easily accessible, serviceable and replaceable.

2.2 The Robot Arm

For Task 3 but mainly for task 4, a robotic arm was selected as the best and easiest solution to pick up the “weeds” out of the weed patch. The robotic arm was custom made at the university and is a 4 Degree of freedom arm. The arm is made from 3d printed parts and a couple of brought in components. The main way to move the arm is with 4 waterproof hobby grade servos and an additional servo to close the jaws. The arm is shown below in figure 2.



Source: Author’s Own

Figure 2: Robot arm

The robot arm mechanics consists of 4 pivot points as shown below in figure 3. The pivot points are the base, shoulder, elbow and wrist. The base was made of a platform supported by ball bearings for allowing the base to slew around. The shoulder, elbow and wrist can only move in the same directions as shown by figure 3 below. The main movement for these joints was from the power of the servo motors.

Source: Author's Own

Figure 3: Robot Arm and Camera Operating in Field Conditions

The camera was then mounted to just above the jaw to allow the camera to have the greatest field of view. It was observed during testing that the higher the camera can be the more uniform the objects were in the camera's view so the more likely it was to see them. The final mounted arm and camera are shown below in figure 4.

Source: Author's Own:

Figure 4: Robot Arm in Task 4 Mode Picking Up Golf Tees

2.3 Body Shell

As the robot had a functional & "naked" appeal it was decided that to appeal to sponsors a sort of marketing display was required on the vehicle. The team thought the best approach was to us an "off the shelf" RC car body shell as this would require little time and effort to fit and made the robot look very attractive. The shell was provided in clear PVC plastic and it was decided to keep it this way so to keep to the functional & "naked" theme of the vehicle and also allowing the internal working to still be admired. As the robot arm was needed to be fitted for task 3 and 4 it was decided to only run with the shell in task 1 and 2. This was due to the fact that most of the shell would have to be cut away to allow the arm to function and it would of ended up looking very unattractive. The completed fitting shell is shown in figure 5 along with our sponsor's stickers.

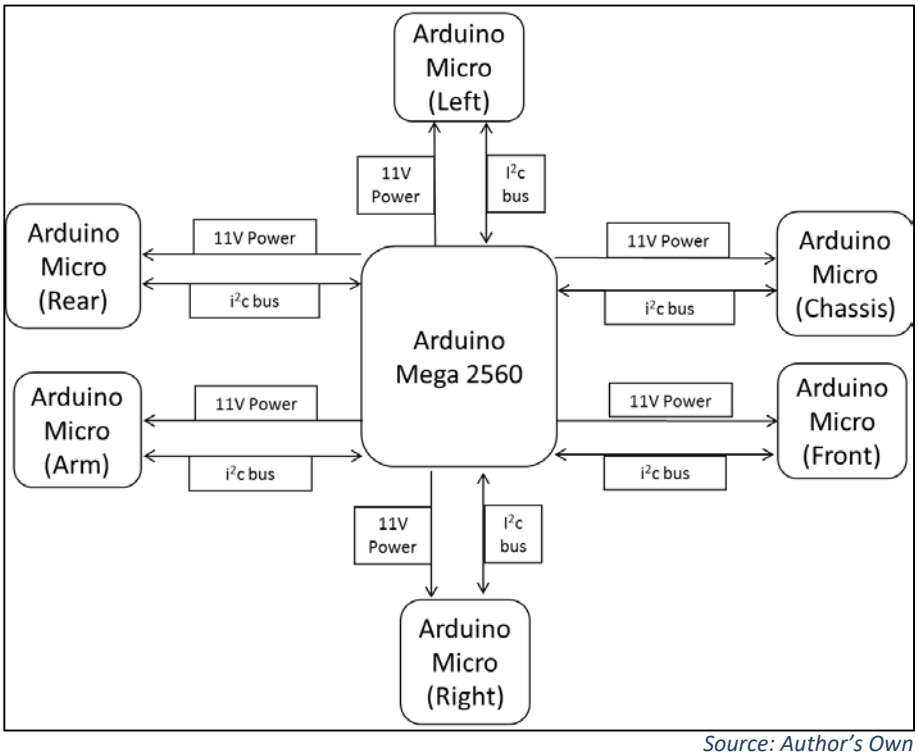
Source: Author's Own

Figure 5: Body shell Mounted to Robot with Sponsor Stickers on

3.0 Sensors and Hardware

3.1 Control Hardware

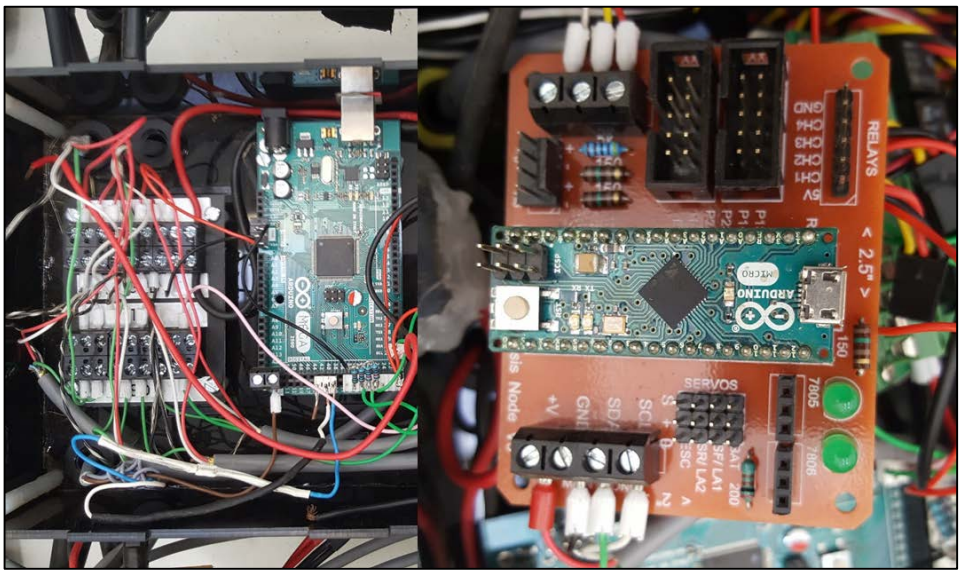
The sensors used are in a system of distributed control, with each Arduino Micro operating their respective sensors, and the Mega with the overall control, which communicated via an Inter-Integrated Circuit (i^2c) bus. This allowed each controller to take care of time critical sensing (such as reading of the encoder) and then pass data on request to the system master controller. This benefitted the application of the robot as it had a lower cost and lower software complexity that other options. Controller communication occurred over an i^2c bus with an Arduino Mega 2560 acting as the master and Arduino Micros acting as slaves. Figure 6 details the controllers used on the robot and their connections.



Source: Author's Own

Figure 6: Connections to Vehicle's components

An 11.1V power was provided to all on-board controllers (omitted from figure 7 for clarity). Figure 7 shows the Arduino Mega 2560 used as the system controller and the Arduino Micro used for each of the nodes.



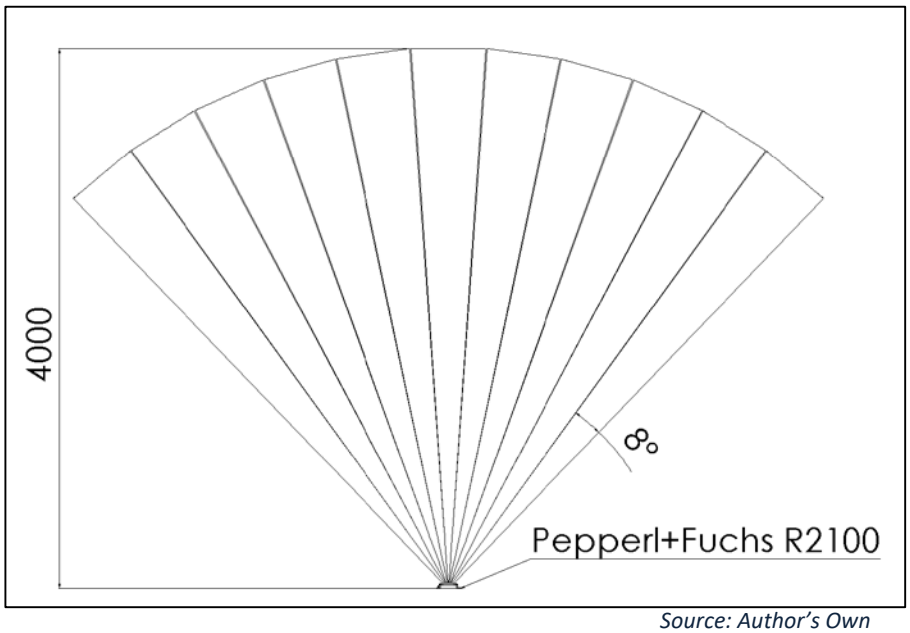
Source: Author's Own

Figure 7: Arduino Mega 2560 (left) and Arduino Micro Node (right)

The Mega 2560 was selected based on its program memory size and processing power, as it would be storing the program for all four tasks. Processing power was not as critical as first imagined as many time critical functions and processing-hungry functions (e.g. printing to an LCD screen) had been removed. Instead, it was important for the Micros, as the code was smaller the memory size required was not as great, and physical size took greater importance. The cost for both of these types of controllers was greatly less in comparison to an embedded computer.

3.2 Obstacle Detection

In order to determine the distance from an obstacle, four Pepperl+Fuchs 2-D LiDAR Sensors (OMD8000-R2100-R2-2V15) were used, (Pepperl+Fuchs, 2018), shows their mounting position. This sensor had been utilised by previous teams – hence its application was proven. The detection envelope can be seen in figure 8.



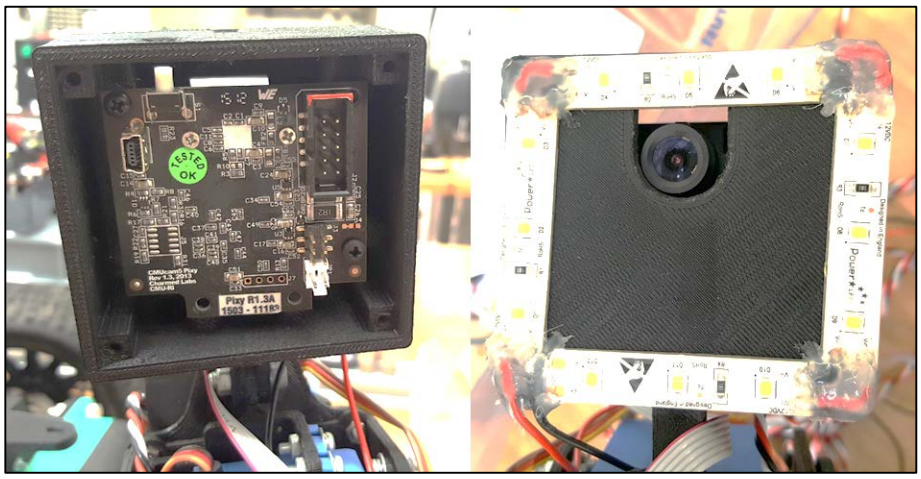
Source: Author's Own

Figure 8: Detection envelope of the Pepperl+Fuchs LIDAR R2100 sensor

Each sensor was interfaced with an Arduino Micro via a RS232-TTL transceiver, as shown in figure 9. The control hardware for each LED scanner was mounted onto a custom designed PCB. This was designed such that most components were socketed, allowing for quick replacement in the case of a malfunction. Each PCB was kept separate, allowing the system as a whole to be replicated or replaced.

3.3 Colour

A sensor capable of detecting colour was needed to compete in tasks 3 and 4. The previous years had used, and proven the ability of, Pixy CMUcam5s, which was mounted on the arm at the front of the vehicle. The camera has a resolution of 600x400 pixels and uses a colour filtering algorithm to detect programmable colours, (Charmed Labs, 2018). It relays the information via a serial protocol to an Arduino and processes the images at 50 frames per second sending only useful information such as an object size and position in the field of view (FOV) to the microcontroller, (Charmed Labs, 2018).



Source: Author's Own

Figure 9: Pixy Cam

Source: Author's Own

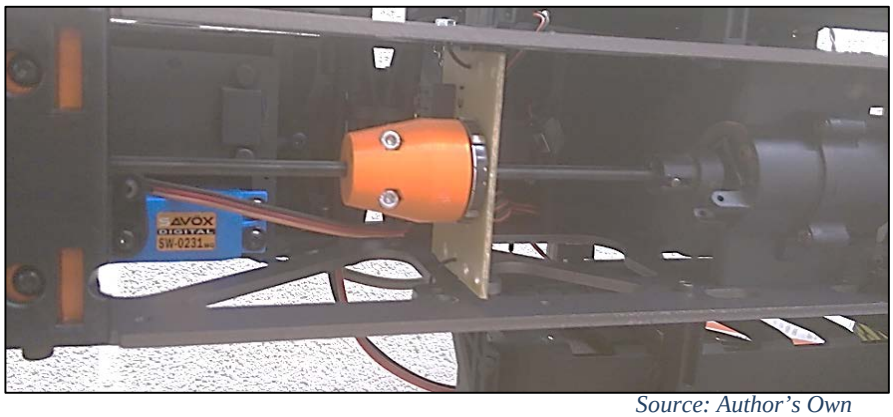
Figure 10: Pixy Cam in Viewing Position

As seen in figure 9 and 10, they were mounted to the arm at the front of the vehicle. The arm as then placed so that the camera had enough height to cover the whole of a row width allowing for any vehicle navigation errors.

3.4 Navigation Aide

An Alps 16 pulse incremental rotary encoder was used to measure the distance travelled by the vehicle. The encoder's was mounted directly to the drive-line and provided 16 pulses per

revolution, inclusive of revolution direction. Due to the gear ratios of the differential, this provided approximately one pulse per 10mm of vehicle travel. It was fitted onto a 3D printed custom mounting flange which clamped around the driveshaft, which is shown in figure 11.



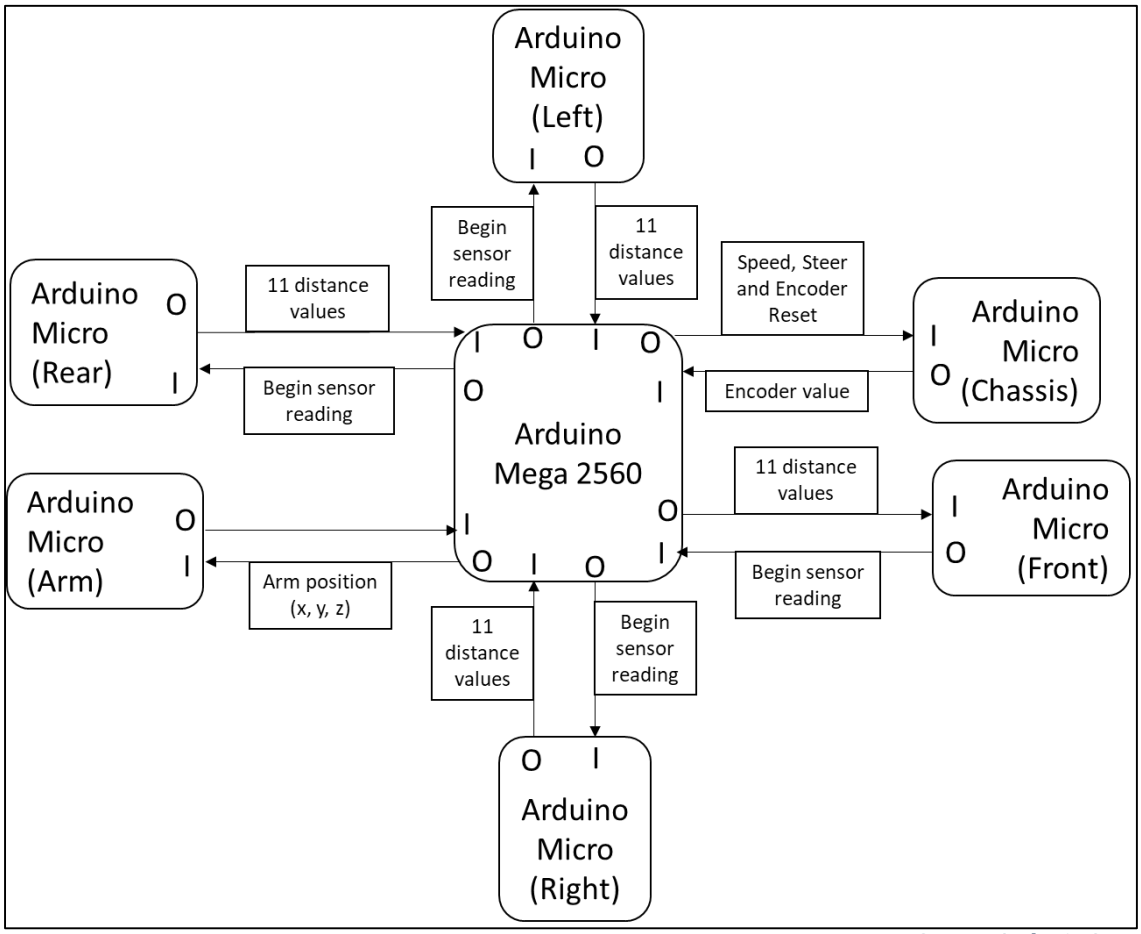
Source: Author's Own

Figure 1: Rotary Encoder Mounting and Position

4.0 Control Architecture

4.1 Overview System Structure

The previous year's strategy of using an Arduino Mega as the master computer, and having Arduino Micros as the slaves, communicating via i2c lines, was continued for this year. The Arduino Mega had unidirectional control over the Micros, therefore the Mega states when it requires the information gathered by the Micros, while the Micros operated their respective sensors, and provided the Mega with the information. The inputs and outputs for each of the microcontrollers are shown in figure 12.



Source: Author's Own

Figure 12: Communication layout of whole vehicle

The Mega code was separated into different tabs, for simplicity of coding. The "main" loop was comparatively short, as it called many other loops within it. Each task had its own tab, which had a singular loop, and was called from the main tab.

4.2 Overall Code Strategy

For tasks one and two, there were five nodes to be controlled by the Mega, one for each of the four sensors (front, left, right, back) and the chassis node, and six for tasks 3 and 4,

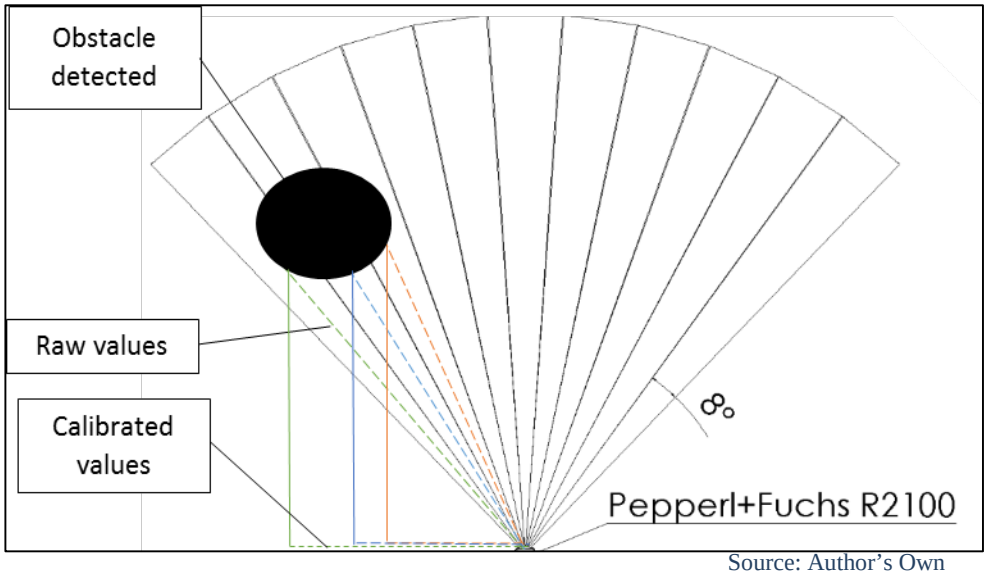
inclusive of the arm node. The main loop of the Mega operated the Micros with “read” and “write” signals, where write occurred at least 20ms before the node was read, except for the chassis node. The chassis node was only written at the end of the loop, after the speed and steer values had been assigned from the task control.

Before the chassis node was written, some emergency protocols were placed, such as the remote start/stop, and the emergency start/stop (dependant of direction and distance from either the front or rear sensor). This means whatever speed was dictated from the task control would be overwritten to stationary if an emergency stop was required. Only after this would the value be written to the chassis node, where the robots speed would be controlled.

4.3 Emergency Start/Stop

As stated before, one the final calculations done before writing to the chassis node is weather an emergency stop is required.

This can come from two different methods, using the front and rear sensor readings and direction of travel to dictate if a crash was imminent, and the start/stop remote. Three centre beams from each of the front and rear sensors are untampered with from row follow, as shown in figure 13.



Source: Author’s Own

Figure 13: Beam allocation of front and rear LIDAR sensors

A threshold value of 13cm was used. Therefore, if the robot was previously traveling forwards, and the one of the centre three beams on the front sensor had a value of less than 13, an emergency stop would occur.

The other method was the start/stop remote. The remote itself had four buttons, which activated four relay switches via radio waves. These relay switches were hardwired into the Arduino Mega, and if the stop switch was activated, it would incur an emergency stop.

The correlation between the four switches and the four relays can be given in three different ways. The method chosen for the robot was “flip-flop”, wherein buttons and relays 1 and 2 were linked, as were three and four. When button 1 is pressed relay 1 switches on, and Relay 2 switches off. Vice-versa for button 2. Relay 3 & 4 operate in the same manor with transmitter buttons 3 & 4. This was chosen, as there could be a start and a stop button, which would operate collectively.

4.4 Sensor Timings

On analysing the coding for the LIDAR sensors it was found that the sensor was being asked for data far too often. The data sheet for the sensor indicates that the request for data should not be less than every 20ms because it takes 20ms for the sensor to make its scan of the 11 beams. The code had been written such that data was being asked for randomly by the request from the Mega to receive new information, with no regard to timing. As control systems work much better with minimal delay between the reading of sensors and providing corrective action, the code has been modified such that data is requested from the LIDAR sensor by the Mega by writing data to the sensor nodes. On investigation it was found that the LIDAR sensors return the result within 6ms of the request for data to be sent, and so the Mega code has been changed to work on a loop time of 25ms. At the start of the loop, the sensor nodes are sent data to request the node sensor to request data from the LIDAR sensor. 10ms later (when the data will have been received from the sensor), the Mega triggers the reading of data from the sensor and then determines the corrective action and sends the required steering and speed demands to the Chassis sensor. After 25ms the loop starts again, by which time the LIDAR sensors will have completed another scan.

This made a very significant improvement to the robot with it row following and entering rows far more reliable and smooth.

4.5 Encoder Coding

It was found that only the negative edge of one of the encoder channels was being detected for counting distance travelled and direction. This has serious consequences if the encoder position is such that the encoder channel is toggling between high and low as counting will continue even if the vehicle is stationary. Encoders with two channels displaced by 90°have been designed to overcome this problem as one edge toggling causes the counter to just count up then down or visa-versa, providing that all edges are detected. For this reason, the code was modified to detect all edges as is the design intent for quadrature encoders. This has the added advantage of increasing the resolution of distance measured by a factor of four such that each encoder count was equivalent to 2.3mm of distance travelled. This would be needed to complete task 4.

5.0 Navigation Strategy

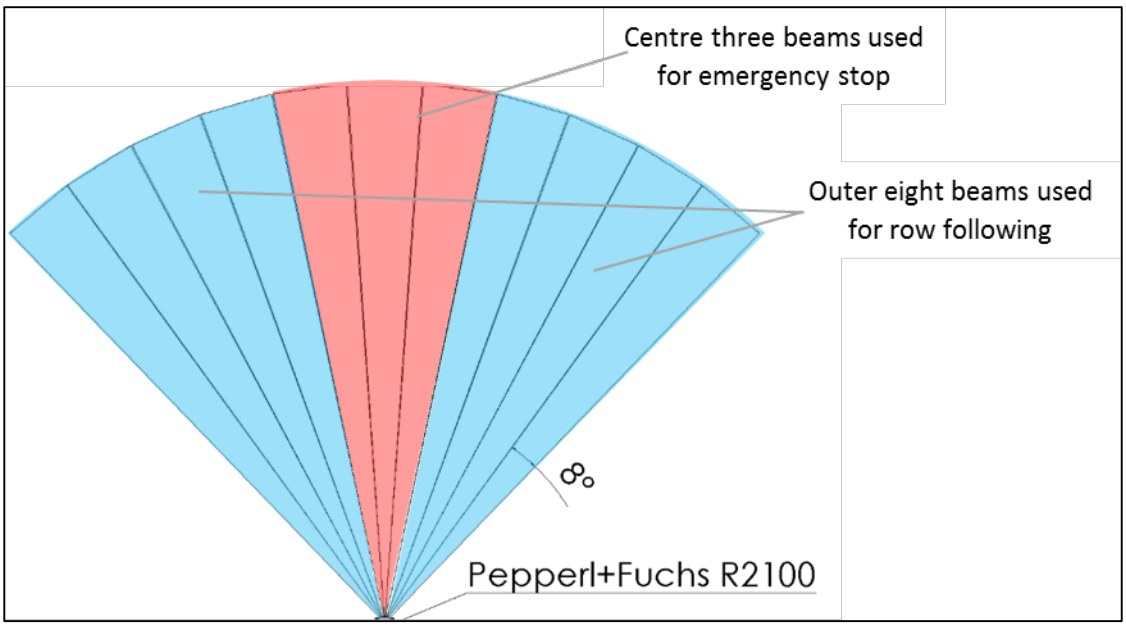
5.1 Task1 and Task 2

The navigation functions for tasks 1 and 2 all used the same layout: first go down a row (10 forwards, 10 reverse) then turn left or right, and skip 0-4 rows. These were labelled accordingly for example, FL0 for go down a row forwards, and go left, skipping 0 rows. As speed is a key aspect of the competition, it was found that by going down the next row in reverse made for a quicker headland turn.

Over 20 functions were written for the different turns required. However, this did not slow down the system, or take up unnecessary space, as only very few functions were called at any time.

5.1.1 Row Follow

Previously, both the perpendicular values from the side sensors and the lateral values from the front sensor were used to calculate the steer values for following down a row forwards. However, after much testing, it was found that they would tend to "argue" with each other, and show highly different values, even after they have been calibrated. Therefore, it was decided to use only the front sensor for forwards, and rear sensor for backwards.



Source: Author's Own

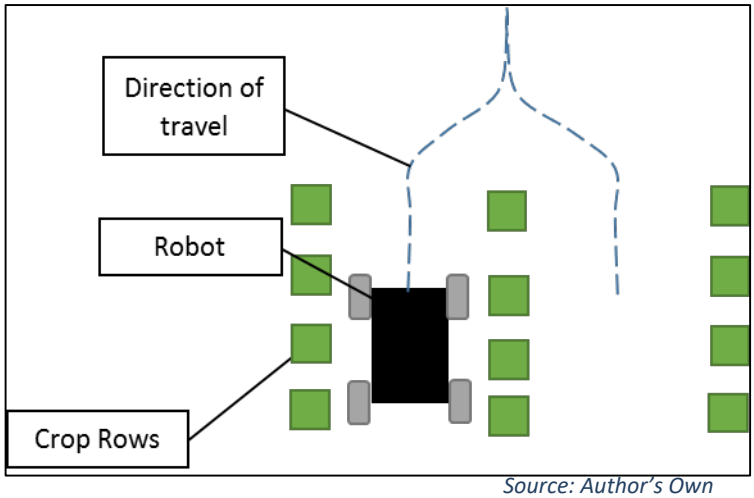
Figure 14: Obstacle Detection Diagram

The calibrated values use the cosine of their angle to the centre beam and is multiplied by raw values as shown in figure 14. This meant if the robot saw a vertical line (i.e. a crop row) the calibrated values would output the same value throughout. This row follow method used the outer 8 beams of the 11 beams from the sensor. The lowest value from the left four and the right four were calculated, and from there, a proportional steer value was given.

5.1.2 Headland Turn

Some changes were made from the previous year, such as no longer using compass readings for turnings, instead relying on encoder counts. This is because in the majority of the testing places (soil -hall, the back of the engineering innovation centre) were not compatible with the compass, due to a high metal environment, and would distort the readings.

As speed is a key aspect of the competition, it was found that by going down the next row in reverse made for a quicker headland turn. The path for this is demonstrated in figure 15.



Source: Author's Own

Figure 15: Headland Turn into Next Row

Other turns that do not go into the next row (required for task 2) used the side sensors to wall follow the ends of the crop rows down the headland, for a given amount of encoder counts, before entering the required row. It was ensured the robot stopped with room to spare, so that it wasn't "blindly" entering a row. The front sensor can view a row from at least 0.5m outside it, and was capable of guiding the robot in with the usual row follow code.

5.2 Task 3

After many discussions, the team decided the best method for tasks three and four was the robotic arm approach. The arm could be used to hold the pixy cam in a steady position for task 3, and could pick up the weeds in task 4.

The pixy cam is capable of labelling areas of similar colours (known as blocks) as different signatures. Two signatures were written for the two colours of weeds, signature one was red, and signature two was blue. The pixy cam could also give the total number of blocks, as well as the signature label for each of them, all in a matrix. The calculations were done by adding together all the signatures, and taking away the total number of blocks. This gives the number of blue weeds. The reds were them calculated by total number of blocks minus the blue weeds.

A problem with the pixy camera is that it is highly sensitive to changes in light, and a change in the lighting conditions, or the background colour could result in it being unable to pick-out

the signatures correctly. Therefore, lighting was attached to the pixy cam, so that in the event of cloud cover, the lighting would remain more constant. This is shown in figure 10

5.3 Task 4

As well as labelling the signatures from the colours, the pixy cam is also capable of giving two dimensional Cartesian co-ordinates (x,y) of the different signatures. These can then be calibrated, along with a z value (height from the ground), to the robotic arm to pick up the weeds.

It was the team's belief that this approach would result in a very low amount of soil held by the weeds and minimal soil tillage. However, it is a time-consuming exercise, but the time limit was given as 5 minutes, which the team hoped would be long enough. It was also hoped the team would be in contest for the best design prize.

6.0 Problems Faced Pre- Competition

This section of the report will be looking at the problems faced as the project progressed and how they were overcome.

6.1 I^2C bus Issues

After considerable reliability testing, the robot was found to glitch out being stuck in random parts of the program due to waiting for data. It was found after further investigation that the I^2C bus would hang due to noise or crash out completely. So to rectify the issues with noise, capacitors were added to the dc motor to minimise noise generation at source, the robot was re-wired with shielded wires and EMF shielding sheets placed in all the boxes.

After further testing the issue still remained. So it was found that the Arduino I^2C or Wire library had a lot of inbuilt faults that weren't fixable by our limited coding knowledge. As a result, it got recommended that we use a Watchdog Timer to reset the I^2C bus allowing the robot to function.

A Watchdog timer is an external circuit to the processor that can detect and trigger a processor reset (and/or another event such as an I^2C bus rest) if necessary. At the start of every program loop the Microcontroller checks in with the watchdog timer and effectively resets the timer. As we have timed our loop speeds we knew that anything more than 50ms could be classed as an error or fault. Like a bomb, the watchdog timer is set to count down and if it times out at 50ms, it resets the I^2C bus as this is where the microcontroller would hang or become unresponsive. This was achieved by using an interrupt function which operates much like someone poking a stick in a rotating bicycle wheel, the microcontroller will stop all functions to perform the reset function. But as long as the program was still looping, it will continue to check in with the watchdog to reset the timer and it would not interrupt the program.

This watchdog timer for the I^2C bus reset was found to cure must of our big underlying issues with system crashes and inaccuracies.

6.2 Data Skew

When variables that are larger in size than the platform being used are being transferred from one level of code another level of code, consideration should be made of data skew. For example, the ReadEncoder routine in the chassis node updates a 16 bit variable, which is then transferred to the Mega in an interrupt routine. As the node has an 8 bit processor, it is possible for the interrupt routine to read the 16 bit value part of the way through it being updated in the ReadEncoder routine:

ReadEncoder is changing the 16 bit value from 0x00ff to 0x0100 (+1)
Least Significant Byte (LSB) is changed from 0xff to 0x00
Interrupt occurs and reads 16 bit value as 0x0000
ReadEncoder Routine changes Most Significant Byte (MSB) to 0x01

The interrupt routine has occurred part of the way through the update of the value in the ReadEncoder routine and read an incorrect value (Date skew)

This can be avoided by different methods:

1. **Double buffering:** Data is transferred from one level to another through a two buffer mechanism, one buffer being active and the other being inactive. A flag is used to indicate which buffer is active. The level passing data to the other level updates the non-active buffer and then switches the flag to indicate that this is now the active buffer. The level reading the data only ever takes data from the active buffer. This ensures that data is read from a buffer that is never being updated.
2. **Disabling interrupts:** If it is possible to do so, interrupts can be disabled during the updating of a variable such that it is not possible for the interrupt routine to read the data while it is being updated.

Method 1 has been implemented in the Chassis node for the 16 bit Encoder count variable. Once this had been done, it was found that there was a significant improvement in the robot reliability. Previously the robot would have odd runs with no explanation.

7.0 Problems during Contest

The robot runs fantastically well on the straight rows of task 2 but didn't perform very well in any of the tasks. Our test day was filled with a lot of problem solving mainly speed controller issues and remote control issues. But most importantly the encoder values are completely set up to perform turn manoeuvres up to a 5 row jump, this was where most of our time was spent on Monday due to us playing catch up because of the issues mentioned previously.

7.1 Task 1

At the event, the robot completed 45m of 110m in the curvy rows of task 1. The robot was found to not navigate the curvy rows very well mainly due to the un-tuned gain and limit values that govern the steering reaction to sensor distance values. We tried to increase the limit values and reduce the gain for the speed we were running at but due to limited time

would could not complete this tuning properly. This was mainly due to the poorly grown curvy rows in the test field. The encoder values for the control strategy for task 1 are set up for a perfect middle exit which obviously didn't happen.

We would recommend that after a completed turn the robot should reverse straight to analyse the end of the row to allow for the row following program to run effectively. Also the speed of the robot needs to be increased further or the systems components improved to allow for full speed running all the time as the speed was limited due to the encoder speed maxing out.

7.2 Task 2

For simplicity it was decided to code all the required turns under functions so in the task 2 competition the functions could be called in the require sequence as there was a 20-minute time constraint to code the robots. It was established on the day of the event that the rules were explained in more detail which did not suit our original strategy of sequence coding. This was because if the robot did not complete a turn it would then have to be reset before the turn it miss complete which then would make our sequence incorrect. If this would have been clearly stated before the contest that strategy could have been adjusted so that a revert back to last function but could have been set up. But due to time constraints on finding out this information it was an unrealistic time constraint to code in time for the event.

7.3 Task 3

The rules for tasks three and four were not released until around 7 weeks before the competition. While this would give the team enough time to make the appropriate decisions and implement them, many of the "problems faced pre-competition" happened during these 7 weeks, as well as the bookings and planning for traveling to the competition. This took up the vast majority of the team's time and efforts and resulted in very little time and energy spent on tasks 3 and 4.

While much of the coding and mechanics for task 3 had been completed prior to the competition, it had not been fully tested. Additionally, the waterproofing for the pixy cam and arm was not prioritized during the short time spent on tasks three and four. The weather appeared to be taking a turn for the worst, and it was a unanimous decision by the team to forego entering, so not to risk damaging the equipment for next year for an untested piece of software. Furthermore, it was highly likely for the robot's abilities to cause negative points from entering the tasks.

7.4 Task 4

As with task 3, the majority of the testing for task 4 had not been completed, and again, to reduce the risk of damaging equipment and causing negative points, it was decided to forego entering in task 4.

8.0 Sponsorship

The team would like to take the opportunity to thank our sponsors, the Douglas Bomford Trust, for their assistance in travel and accommodation to and from the event. And thank Sygenta and Pepperl & Fuchs for enabling us to upgrade the robot for this year's event.

9.0 Conclusion

This report has explained the robot's functions with justifications for hardware, software/coding, sensors selection and the mechanics of the robot. The team placed 10th in task 1, 9th in task 2 and did not compete in task 3, or 4. The team placed 13th overall out of 15 mainly due to the fact we were one of a few teams not to compete in the last task. The team hopes that the future recommendations given will provide any future team from Harper Adams University the building blocks to build this simplified system up into a reliable and competitive machine.

10.0 References

Charmed Labs, 2018. Pixy (CMUcam5). [Online] Available at: http://charmedlabs.com/default/pixy-cmucam5/ [Accessed 26 June 2018].

Field Robot Event, 2018. Field Robot Event: Home. [Online] Available at: http://www.fieldrobot.com/event/ [Accessed 05 April 2018].

Pepperl+Fuchs, 2018. 2-D LiDAR Sensor OMD8000-R2100-R2-2V15. [Online] Available at: https://www.pepperlfuchs.com/great_britain/en/classid_53.htm?view=productdetails&prodid=62235 [Accessed 05 April 2018].

FARMBEAST

Peter Bernad[1], Aljaž Zajc[2], Rok Friš[2], Jernej Mlinarič[2], Jurij Rakun[3*]

[1)] *University of Maribor, Faculty of Natural Sciences and Mathematics, Slovenia*
[2)] *University of Maribor, Faculty of Electrical Engineering and Computer Science, Slovenia*
[3)] *University of Maribor, Faculty of Agriculture and Life Sciences, Chair of Biosystems Engineering, Slovenia*

[*)] *Instructor and Supervisor*

1 Introduction

The use of robotics in agriculture is unavoidable. This is not just a far fetched future scenario, but a present reality with it is full swing not from now. The world population is expected to grow to 9.8 billion by the year 2050 for which around 70 % more food will have to be produced to fulfill the demands of rapidly growing population [1][2]. One of the necessary production steps will be the utilization of robotic in agriculture, since the robots are faster, more accurate, they cannot get tired and can also work in conditions unsuitable for human workers. This will optimize the food production process, minimize its expenses and have less negative impacts to the environment [3].

An example of such possible robots are the robots developed by Naio Technologies [4], Rowbot [5], Wall-ye [6] and other. Some of them are already commercially available, already useful in the fields and produce good results. Such robots can already weed, hoe, seed crops and even cut the twigs in vineyards in a successful and autonomous manner. So, the farmer has no need to do these repetitive tasks and can concentrate on being more productive in other fields.

The robots mentioned in previous paragraph are not very different from the one described in the following sections. But this is not the first version of this robot. The development of a field robot is conducted in form of a student project that dates back to 2009. Since then the student team went through many iterations and tests. The first version of the robot was based on a 4 wheeled RC platform that was driven by an internal combustion engine and controlled by an energy hungry x86 computer. Additional sensors were then added to this platform, such as ultrasonic sensors and a Bayer coded camera to sense the environment the robot is in. Back then a specific code was developed in a Linux operating system that captured all the reading and send them to a remote computer for data analysis developed in Labview and Matlab. From them we have upgrade the robot and through the years, completely change the working principle, along with its software and hardware. The new robot is now driven by 4 individually controlled motors, controlled by an embedded distributed computer system, with 2 industrial cameras and a LIDAR sensor mounted on the robot. The software is now implemented as part of the ROS system for easier implementation of algorithms and better portability to another system. The following chapters describe all this in detail.

2 Mechanics and Power

The robot is designed for driving between the two crop lines, which dictates its dimensions. One of such requirements is around 750 mm of space between the rows, so the robot has to be small in order to be able to turn around on the spot, which also gives it better maneuverability on the field. The chassis itself is made of aluminium and that makes robot much lighter.

The inside of robot is divided in three levels. The lowest is for the the battery. By putting the battery as low as possible we put the barycenter closer to the ground, which makes the robot more stable. In middle level we placed all four motor drivers and power supply components, such as DC-DC converters. On third, top level, we mounted other electronics, such as an extensional board, Raspberry pi, NUC and PoE managed switch. We then covered all three levels with a plastic case, which was custom designed and 3D printed. On the case itself we installed some interactive appliances, such as touch screen OLED LCD, a couple of buttons and two cameras. At the back side of the robot, we installed a connector to connect external appliances, such as a sprinkler and a laser system. A CAD design of the FarmBeast robot is depicted on Figure 1.

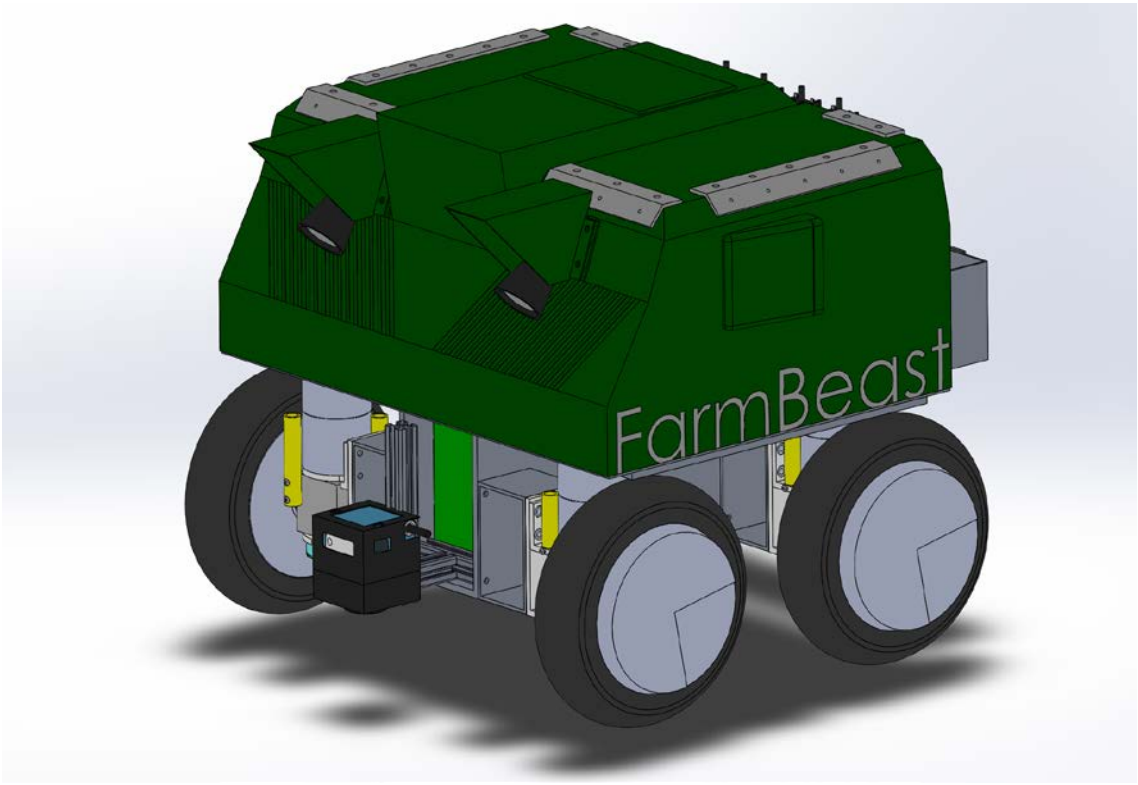


Figure 1: CAD design of a FarmBeast robot.

The robot is powered by four in-wheel (BLDC) electric motors that provide 200 W of power per wheel. Each wheel is attached to a steering mechanism which can rotate the wheel for a 180°. This configuration allows each wheel to rotate at different speeds and steer in any direction independently. Wheels are equipped with incremental encoders that allow the motor controller to regulate the power output to achieve a constant rotational speed regardless of the terrain. Similarly to the wheels, the steering mechanism in equipped with an absolute encoder. This allows the electronics to align all the wheels to a pre-programmed angle at any moment even after a reset.

The robot is powered by a large capacity $LiFePO_4$ battery with a nominal voltage of 25.6 Volts. It was designed to allow the robot to operate in idle for approximately 10 hours and 2 hours while driving. Battery is placed in the middle of the chassis which protects it from impacts.

Robots chassis is made out of 6 mm aluminium plates bend in required shapes. A large plate is bend in such a way that it houses the battery and forms a horizontal mounting area for other components. It has four cut out holes for wheels steering mechanism motors and additional two for carrying handles. Four smaller rectangular aluminium profiles connect the wheels to the chassis which houses the battery. CAD design of the robot's chassis, the wheels and the battery is depicted on Figure 2.

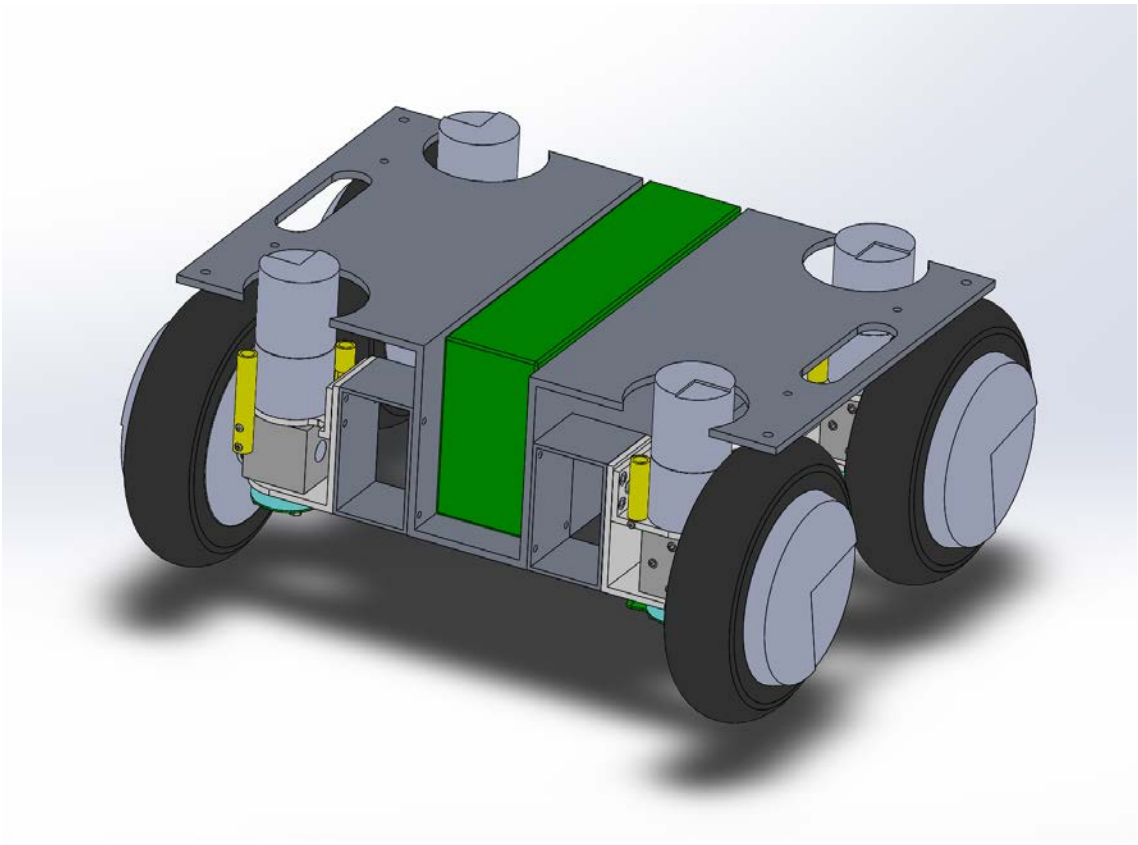


Figure 2: CAD design of the robot's chassis, drivetrain and battery.

As an expansion to our robot we have designed a unique spraying unit. It is used to precisely target weeds, so it activates the nozzles only when a targeted weed is underneath, following similar process such as described in [5]. This way we can reduce the usage of plant protection products (PPP) and apply them when necessary. The system uses a micro diaphragm high pressure pump which operates at 24 Volts. It has a built in pressure switch that automatically turns the pump off when the output pressure reaches 8 bar, so there is no need to control the pressure by a microcontroller. Between the pump and the nozzles there are currently installed two solenoid valves, each connected to two nozzles. This system is designed to include two additional valves, so each nozzle can be controlled by its own valve for maximum accuracy and minimum usage of PPP. Flued is stored in a specially designed tank which fills up most of the remaining space around other components. Its holding capacity is approximately 1.5 liters. The holding capacity is small, but it lasts a long time if weeds are selectively targeted, so ti is enough for testing purposes. The tank was made by 3D printing four parts, which were connected with screws and quick seal glue. The whole spraying system is shown on Figure 3.

Figure 3: A custom build spraying unit.

The whole system is placed in a metal case which attaches to the robot with 3D printed carriers and pins. Pins are designed to allow quick attachment and detachment of the unit which takes approximately 30 seconds. Power supply lines and valve control lines are connected to the robot via a single 7 pin connector. Two pins are used for pump positive and ground terminal and remaining five for valve control and another ground.

3 Controller Architecture

3.1 Computer and other Hardware

The robot is built on a distributed computer system, including a high level and low level computer, both running a Linux operating system. The first one is a Raspberry Pi 3, an energy

efficient device used for controlling basic robot operations. It has a 1.2 GHz 64 bit CPU, 1 GB of RAM and build-in hardware for wireless connection via Wi-Fi and Bluetooth. The second computer, Intel NUC7i7BNH, is way more powerful build around the Intel I7 processor, reaching the speed up to 4 GHz and has 8 GB RAM of memory. It's down side are the power requirements as it consumes a lot more electrical energy. This is the reason it is only used when we need more power for processing data for specific tasks, such as computer vision algorithms.

3.2 Software and strategy

TASK 1

In the first task, the robot has to navigate autonomously between the two curved rows of maize plants [8]. In this task the robot is switching between the two steps. First, it has to navigate between the two parallel rows, then it has to make a turn into the next one. The two steps are repeated until the robot drives 150 m or the time of 3 minutes runs out.

Step 1

For navigation through the row, we use data from the SICK TIM310 LIDAR sensor. So in the first step the algorithm is calculating an optimal path/turn in each "window", that is generated each time we get a new set of data from the sensor. The sensor generates a collection of distance readings from the surrounding in front of the robot in 270 degrees angle. For each degree, the sensor measures the distance to the nearest obstacle. From those points the algorithm generates two lines that correspond to the collection of points on each side of the robot. The approach is presented on Figure 4.

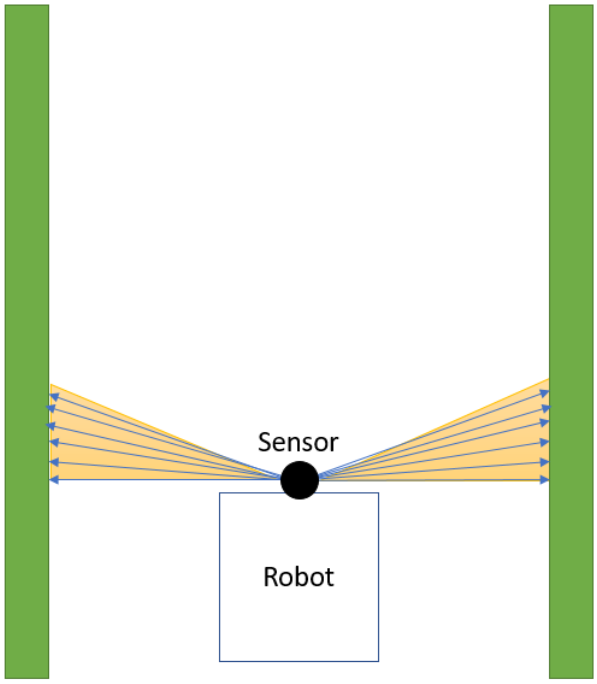


Figure 4: Collecting data from the LIDAR on each side of the robot.

The measured points create two lines, representing part of each side of the row that is in front of the robot. The goal is to calculate the direction we need to go to and not to drive over the rows of the maize plants. For that we need to define some variables first, which are described in the lower picture:

$$x_{sum} - sum\ of\ all\ the\ distances\ taken\ for\ the\ line,$$
$$x_{square\ sum} - sum\ of\ all\ the\ squared\ distances\ taken\ for\ the\ line,$$
$$y_{sum} - sum\ of\ all\ the\ angles\ taken\ for\ the\ line\ (in\ degrees),$$
$$y_{square\ sum} - sum\ of\ all\ the\ squared\ angles\ taken\ for\ the\ line\ (in\ degrees),$$
$$yx_{sum} - sum\ of\ the\ products\ of\ the\ angle\ and\ distance\ of\ each\ point.$$

With those variables we can first calculate the slope and y intercept of each line as shown by the following equations:

$$k = \frac{number\ of\ points * yx_{sum} - x_{sum} * y_{sum}}{number\ of\ elements * x_{square\ sum} - x_{sum}^2}$$

$$n = \frac{x_{square\ sum} * y_{sum} - x_{sum} * yx_{sum}}{number\ of\ elements * x_{square\ sum} - x_{sum}^2}$$

With the slopes and y intercepts we get the interception, where the two lines meet (the slope and y intercept of the left line is marked with a 1, the slope and y intercept of the right line are marked with a 2):

$$x_L = \frac{n_2 - n_1}{k_1 - k_2}$$

$$y_L = k_1 * x_L + n_1$$

This point defines in which direction the robot has to steer. The goal is to keep the intercept point always at the middle, so the robot doesn't touch the maize plants.

When the sensor / algorithm does not detect any points in front of the robot, for instance when it reaches the end of the row, the robot signals the end of the first step and starts the second step.

Step 2

In step 2 the robot makes a turn into the next row, that is the space between the crop lines. It has to drive out of the current row and make a 90 degrees turn towards the adjacent row that the robot has not driven in yet. After that it drives forward for 75 cm and makes another 90 degrees turn into the new row.

To calculate the turn, it uses the data from the compass, which limits the turn speed because of the data refresh rate. For measuring distance travelled out of the current row and into the next one, the readings from the odometry values of each wheel are used. This allows us to have minimal drift error, since each wheel is controlled individually.

Once the robot makes the last 90 degrees turn, it signals the end of the second step and switches back to the first step.

Task 2

In this task the robot must navigate autonomously through the field [9]. The rules for entering the field are the same as in Task 1, but the terrain is more challenging then in Task 1. At some parts of the field plants might be missing (small gaps) and the robot needs to adapt to the changes. The robots also have to follow a certain predefined path pattern across the field. S means start, L means left, R means right and F finish. For example, 2L means the robot turns left and goes to the second row. 4R means robot turns right and goes into fourth row. Our program reads a text file which has this values entered. These values can be changed if there is a demand to do so. Example of enter values: S - 3L - 2R - 1R - 4L- F. There are some small stones placed on the field and the robot must be able to climb over these obstacles or turn on the spot and go back.

For this task we modified the algorithm for driving between the lines of corn and for the rotation used in the first task. The difference is the missing corns at some parts of the route and the rows in this task are straight. So we adapted the algorithm to align the robot to the line of corn that is not missing, and this way it is possible for the robot to continue its way. Another exception is the length the robot has to drive in the headland when it performs the turning maneuver. This part is solved by the odometry form the wheels.

Task 3

In the third task the robot has to detect blue and red colored golf tees [10]. Again, the algorithm is cycling through two steps. In the first step the robot uses the already described algorithm for navigation through the rows. In addition, the other algorithm looks for golf tees and signals accordingly with a beep, if it detects them. The second step is the same as the second step from the first task. That is the reason are only describing the camera detection part here.

Step 1

Beside the calculation of the optimal path as described in task 1, the robot is also checking the data from the camera in each frame. The data in each frame of the video is in a form of a RGB matrix. Since we do not want to check all three colour planes and combine the colours for each pixel, the data is transformed into the HSV representation. In this case we only need to check one plane for colour values. So the algorithm thresholds the original image two times. The first time, it only detects pixels, and the second time, it detects only blue. Then it compares, if there are enough blue or red pixels in order to prevent false detection. The process is simple, yet effective. Once it detects enough pixels with each colour, it compares the number of detected pixels to each other. The robot then beeps once if there are more red pixels and twice if there are more blue pixels. After the robot reaches the end of the row, step 2 is initialized (refer to Task 1).

Task 4

In this task the robot must navigate autonomously through the field [11]. It has to navigate through straight rows as in task 2. But the length from start to finish is only three rows, and there is no specific path to follow, so the robot just has to turn on the headland and return to adjacent row. The robot removes patches of red golf tees placed in the field with his rakes. Three patches are placed in each row. The rakes were 3D printed and modelled by our team and are depicted in Figure 5. It only uses them when the two cameras sense presence of the patches in short distance. When this happens robot stops in the distance of approximately half a meter and activates two stepper motors that start lowering the rakes. The robot drives forward and tries to pick up as many tees as possible. Then, approximately half a meter from the patch, it activates his stepper motors again and pulls up the rakes. Same procedure is applied until it reaches the finish line. The algorithm for driving between lines of corn is the same as in previous tasks.

Figure 5: FarmBeast robot with mounted rakes.

3.3 Sensors

The sensoric part of the robot includes a LIDAR sensor (SICK TIM310) which is used for navigating between the lines of crops. The LIDAR can sense distances between 4 cm up to 4 m in angle of 270°. To estimate rotation an IMU (Phidgets spatial) is used, which is installed in the middle of robot as far as possible from all electronic and high voltage cables, but still in centre of the robot. Because some tasks required colour detection, the robot is equipped with two industrial cameras (The imaging source DFK 23GP031) which are installed on the top of the robot. These two cameras capture photos in real time and send it to the NUC, where it processes the data in order to detect wanted coloros/objects.

4 Results and Discussion of Machine Performance

Machine performance was in general good, if we exclude the failure of a sensor chip on one wheel. This almost prevented our team to compete and made us work late into the night to develop an algorithm to compensate to the missing data. We could not solve this completely and the robot was competing with limited maneuverability.

TASK 1

Due to limited maneuverability of the robot moved in slowed down mode and reached the final 4th place at the end with one point missing to 3rd place.

TASK 2

In this task the robot had some problems with the orientation so it did not completely follow the designated pattern. Otherwise, in general, it performed well.

TASK 3

In third task the robot detected the plots successfully, but with some double detections due to slowed down speed which was not accounted for and was caused by limited maneuverability. The robot was ranked 4th in this task.

TASK 4

The presented rake system proved working but not to be the perfect choice for the fourth task. When tested it was used on flat ground, and worked well. But the task 4 was performed on uneven level that was not compensated by the robot. Although the robot was ranked 5th in this task.

Overall

Although we encountered problems at this years FRE the team reached the final 4th place in overall ranking!

Freestyle

We presented an idea of the approach to threat the weeds by using different approaches; chemical and thermal. As this proved to be a more challenging tasks than previously thought we still have to work on it and will be fully operational by the next FRE.

5 Conclusion

The concept of having an autonomously driving robot on the field is already in practice. With building and developing Farmbeast we concluded that it is not that far from a working field robot. Farmbeast can already drive autonomously with success and operate with some attachments that can be used on the terrain.

One of the biggest disadvantages so far for now is the speed the field robot can reach and still be precise enough to fulfil its task. That depends on both hardware and software as we need to get the data as fast as possible and calculate it fast enough to react to the terrain. All this in safe and efficient way. Through time, we can compromise that problem, but the budget will always limit the quality of sensors needed to operate and the sensors dictates the processing speed.

In the end, we can say that FarmBeast is indeed a good prototype and a beginning to create a fully functional field robot. Hopefully it will grow in to a product that would be a part of everyday equipment on the field, simplifying the work and maximising the production.

References

[1] How to Feed the World in 2050. Food and Agriculture Organization of the United Nations. [Obtained on 22.09.2018]. Available at: http://www.fao.org/fileadmin/templates/wsfs/docs/expert_paper/How_to_Feed_the_World_in_2050.pdf

[2] Britt J.H., Cushman R. A., Dechow C.D., Dobson H., Humbolt P. and others (2018). Invited review: Learning from the future-A vision for dairy farms and cows in 2067. Journal of dairy science, 101, 3722-3741

[3] Slaughter D.C., Giles D.K. and Downey D. (2008) Autonomous robotic weed control systems: A review. Computers and electronics in agriculture, 61, 63-78

[4] Naio Technologies, https://www.naio-technologies.com/

[5] Rowbot, https://www.rowbot.com/

[6] Wall-Ye, http://wall-ye.com/

[7] Vogt W. Super-targeted sprayer getting closer to market. *Farm Industry News*. (23.8.2017). [Obtained on 21.9.2018]. Available at: https://www.farmindustrynews.com/technology/super-targeted-sprayer-getting-closer-market

[8] Field Robot Event, Task 1. [Obtained on 22.9.2018]. Available at: http://www.fieldrobot.com/event/index.php/tasks/task-1/

[9] Field Robot Event, Task 2. [Obtained on 22.9.2018]. Available at: http://www.fieldrobot.com/event/index.php/tasks/task-2/

[10] Field Robot Event, Task 3. [Obtained on 22.9.2018]. Available at: http://www.fieldrobot.com/event/index.php/tasks/task-3/

[11] Field Robot Event, Task 4. [Obtained on 22.9.2018]. Available at: http://www.fieldrobot.com/event/index.php/tasks/task-4/

Sponsors

A big thank you goes to our sponsors that help to make this robot a reality.

Public Scholarship, Development, Disability and Maintenance Fund of the Republic of Slovenia: for partial funding of the project.

KORITNIK s.p.: for cutting and bending the aluminium plates used for the chassis.

SICK d.o.o.: for partal educational discount on the SICK TIM310 LIDAR sensor.

SMT d.o.o.: for Velodyne VLP-16 multichannel LIDAR that is currently being tested on the robot and will be used in the future.

HELIOS - FREDT

David Bernzen[1], Tobias Lamping[1], Steffen Lohmann[1] Christian Schaub[1], Enrico Schleef[1], Julius Steinmatz[1], Constantin Ruhe[1]

[1)] *TU Braunschweig, Institute of Mobile Machines and Commercial Vehicles, Germany*

1 Introduction

The Field Robot Event Design Team (FREDT) was founded in 2005 at the Institute of Mobile Machines and Commercial Vehicles – TU Braunschweig.

Since 2006 our team participate in the annual Field Robot Event. Our 15 Team members are from the areas of mechanical engineering, electronic engineering and computer science.

Helios has been self-developed by our team in the year 2007. The robust mechanical consruction has a very feasible design for mastering the challenges of the Field Robot Event.

2 Mechanics and Power

The durable Helios drive train realises a permanent four-wheel-drive. Helios is based on an aluminium ladder frame with two identical axles. The damped beam axle construction can easily adapt itself to very uneven ground while mainting a good ground clearance. Four identical wheels with AS-pattern allow good power transmission while driving under difficult ground conditions.

Figure 1: Helios with 2018 Sensor configuration

The mechanical power of 250 Watt, provided by the electrical motor (Dunkermotoren BG75X25CI), is spread by three differentials (1x transfer gearbox, 2x Axle).

The gearbox allows a maximum speed of 10 km/h. An Ackermann steering concept at both axles is implemented. This concept combines a small turning radius of 75 cm and smooth movement at the field headland with large vehicle stability when driving straight between the crop rows.

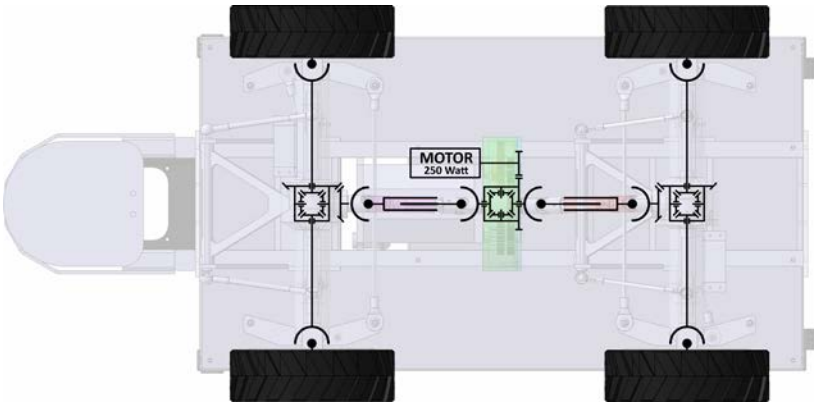


Figure 2: drivetrain of Helios

The vehicle electrical system is based on a nickel–metal hydride battery with 4.5 Ah and 24 V. In the 2018 configuration, a single LIDAR (Sick LMS100) and a single camera (Allied Vision Prosilica) are mounted to the adaptable robot platform.

For Task 4 we developed a collector module. A rotating rake with two tine rows pick up the weed and collects it in a container. The height of the rake is adjustable. Two runners on the outer edges of the module and a vertical adaptable suspension allows a good ground guidance. The usable wide is 330 mm.

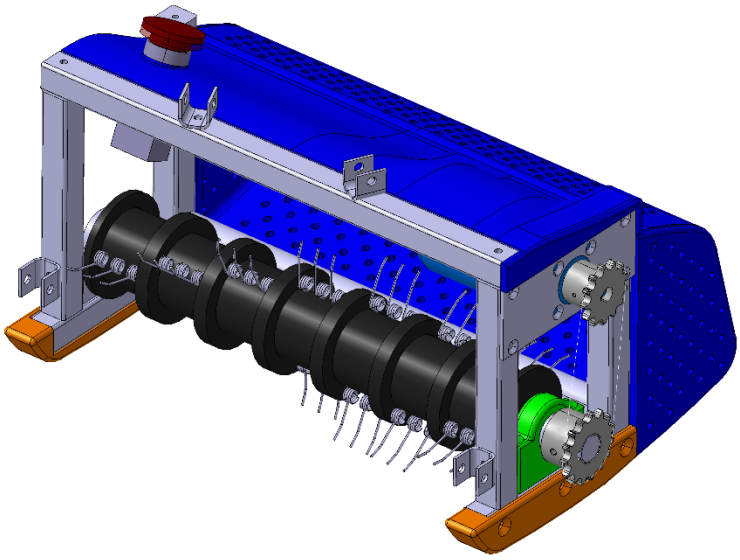


Figure 3: collector module

3 Controller Architecture

The structure and components as well as interfaces and communication means of the electronic controller.

3.1 Computer and other Hardware

- Computer
  - Gigabyte GB-BXI7-4770R
    - 16GB DDR3 SDRAM
    - Intel Core i7-4700R 3.90 GHz
- Laser Scanner
  - SICK TiM571-2050101
  - SICK LMS100-10000
  - (SICK TIM310-1030000S01)
- Router
  - TP-Link TL-WR841N
- Power Supply
  - 24V Nickel-Metal Hydride
- Remote Control
  - Futaba T7C
- Camera
  - Allied Vision Prosilica GC 650C

3.2 Software and strategy

We are using ROS, the robotic operating system, an open source environment for controlling robotic platforms. Its speciality is the heavy usage of the observer design pattern, which, beside other techniques, decouples the need of thread and inter-process-communication management for the developer.

Task 1 & 2

For the overall navigation, but especially for the first two tasks we implemented the ROS Navigation, a 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. The goals were created from a premade global planner, which was not adapted to that special way of navigation at this time. This is one important thing to do for the next event.

The advanced navigation in Task 2 was therefore not possible for our robot because of the mentioned global Planner.

To navigate inside the row, we used the normal front wheel steering powered by servos and for the turn at the end, Helios has a four-wheel control.

Task 3 & 4

For the obstacle detection Tasks our robot used a front facing camera in order to detect the golf tees. The detection was accomplished using OpenCV and by looking at the average colour values of detected tees. Taking the ratio between distinct RGB colour values, we were able to distinguish between the red and blue ones. A speaker on top of the robot produced the signal accordingly to the scanned tees.

Task 5

We did not participate in Task 5 due to several defects at the power supply. We will show our concept in the next event in 2019.

3.3 Sensors

The most important robot sensor is the Sick LMS100 2D laser scanner, with a maximum range of 20m and 270° observation angle with 0.5° angular resolution at 50 Hz scanning frequency. It is permanently installed at the front of the vehicle and provides a precise capture of the near and far environment. For this event, it was installed upside down to be able to correctly scan the small crops.

Additionally, there is a camera (prosilica GC 650C) installed above the laser scanner to identify the small coloured obstacles inside the rows and pick them up with the picker attached at the backside from the robot.

4 Results and Discussion of Machine Performance

A mayor damage to the central electrical distributor on the first day of the field robot event limited our on-field-testing time of the software. After fixing this problem the hardware and software functioned as expected in Task 3. The image recognition works perfectly under all light conditions and is easily adaptable for upcoming tasks.

Due to the electrical error the weed removing unit (Task 4) can´t be lifted which results in a large accumulation of soil.

The next generation of the central electrical distributor is already under development. In addition, a spare part system for all mechanical and electrical parts will be implemented.

5 Conclusion

The flexibility of Helios allows our Team an easy adaptation of the robot to every upcoming new task in on- and off-road environments. After over ten years Helios will be upgraded to comply with all upcoming tasks. The new Software which was implement in the season 2018 is the first step in this project.

6 References

Our website

http://www.fredt.de/

TAFR

Tim Kambič[1], Žiga Brinšek[1], Gal Pavlin[1], Iza Burnik[1], Pavel Remic-Weiss[1], Martin Turk[2], Anrej Šenk[3], Janez Cimerman[1*]

[1)] *Faculty of Electrical Engineering, University of Ljubljana, Slovenia*
[2)] *Faculty of Computer and Information Science*
[3)] *FH Joanneum, Graz, Austria*
[*)] *Team leader*

1 Introduction

One of the disciplines that is on the verge of technological leap is agriculture. We can already see real development on technologies from smart surveillance with drones to small ground autonomous robots working in fleets. All these have recently been made possible due to rise of open-source technology and software, cheaper sensors and actuators and the need to automatize work on farms.

Project TAFR (http://tafr.si) was started to advance the knowledge and awareness of automation technologies in farming among students and public. This robot was designed to tackle real world farming scenarios: driving in between rows of corn, check for weeds and remove them (all autonomously). Although weeds were represented with clear markers (red/blue golf tees) the possibility of the automatization of work on fields was demonstrated with great success.

This report briefly describes the technical aspects of the project, as well as problems and solutions faced when designing the robot for farming.

2 Mechanics and Power

The main guideline for building the whole robot was simplicity. The robot was built with minimal, simple parts, which are easy to replace and/or modify. This gave us a real advantage when testing and correcting all components on the go.

Robot's chassis is constructed from 40mm extruded aluminum profiles. The mounts for motors and tires were all custom milled for this robot as were the Plexi plates on the side of the robot for dust protection. The tires are mounted directly on the motors and connected directly to the motor's axis to achieve extreme simplicity and rigidity of the design. Each wheel is actuated by 125W DC motor.

The robot is powered from 24V 10Ah LiPo batteries stored at the bottom of the robot. All onboard electronics (except motors) can also be powered from a 12V connector at the back of the robot. Motors are speed controlled by RoboClaw motor driver from Pololu. We also installed fuse board (made ourselves) for the protection from high currents and peripheral board (also made ourselves) with which we control all the additional electronics (for example: mechanism for tillage). The central computer is Intel NUC which has i5 processor and 8GB of RAM. Other parts of electronics also include various voltage step-downs, rotary encoders on each motor, SICK Tim LIDAR and an IMU. We have two safety switches at the back of the robot. One is used to cut power only to motor drivers and the other cuts power to the whole robot and is used regularly as ON/OFF button of the robot.

3 Controller Architecture

All the computing is done on a single Intel NUC. The main sensor we used for navigation is lidar from Sick (10-meter range, 0.33° resolution). For turning on the spot we used IMU from Bosch (BNO055) and for all the computer vision tasks the Intel Realsense ZR300 was used.

3.1 Software and strategy

For following the rows of corn, we used the following algorithm: first we performed clustering on lidar scan data, which returned us approximately 1 cluster per plant. Than the closest clusters to the robot were found and the distance to that clusters were used as reference for PID controller used to calculate angular velocity of the robot. For linear velocity the same distance was used but this time with only proportional controller.

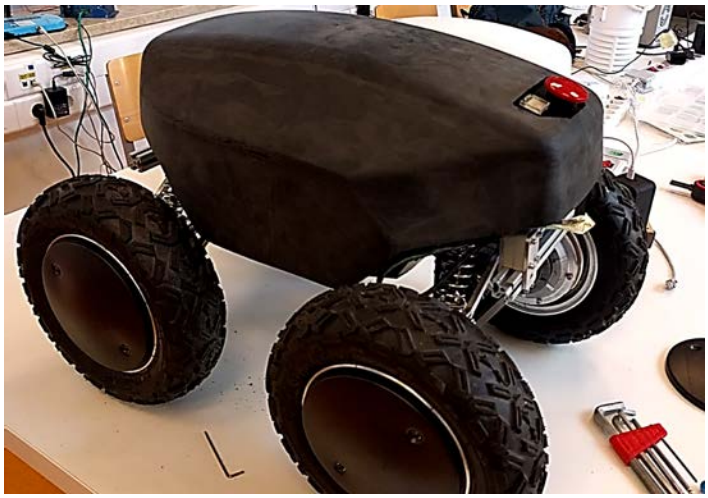
When the end of the row was detected the robot drove straight for 0.7 meters based on odometry from encoders positioned on motors. At that point the robot turned for 90 degrees in the desired direction. The turn was controlled with data from onboard IMU sensor. After the turn the robot used the same clustering as in rows to follow the end of the rows. For counting rows, the odometry information was used. When enough rows were counted the robot again turned 90 degrees to face the plants. To drive in the row the same algorithm as for row following was used, but with different parameters for PID controller. For tasks 3 and 4 basic color segmentation based on histograms was used to detect “weeds”. Detections were filtered by size to remove noise.

4 Results and Discussion of Machine Performance

The software for autonomous driving worked as expected during the competition. For some still unknown reason (maybe the wrong configuration was loaded on the robot) the part of the software used for computer vision (task 3&4) did not operate nominally.

Our current robot has no suspension which proved to be quite a nuisance when driving despite having enormous wheels. Some problems also occurred due to less than ideal network connection between robot and laptops (for developing and debugging).

Due to above mechanical and hardware problems we are building new improved robot that was unfortunately not quite ready for FRE2018.



5 Conclusion

After two years of developing autonomous farming technologies we can conclude that the problem isn't as trivial as it seems. All the unattended problems in hardware affects the development in software and vice versa. The problem also resides in being in uncontrolled outdoor environment. It is also much more difficult to find the right conditions and space for outdoor testing. All in all, with superb technology all that problems can be overcome and solutions for autonomous farming made available to the public.

6 Partners and Future Plans

We could not build this robot without the help of our partners:

Zavod404, Epilog d.o.o., RLS, Laboratory for telecommunications-FE-UL and Mestna Občina Ljubljana.

With their help we can continue to improve our system and implement our ideas and future goals, such as more complex autonomous navigation, based on stereo cameras or 3D LIDAR. Since we will soon have two fully functional robots they can work together on a field and execute tasks more efficiently. One of our next goals is also creating a charging station which will, with use of GPS, automate charging the robots.

VOLTAN

Armando Reyes Amador [1], Luis Gerardo Ruiz González [1] , Norberto Cuaupantecatl Garrido [2], Noé Velázquez López [3*]

[1)] Ph. D. Student, Postgraduate in Agricultural Engineering and Integral Use of Water, Chapingo Autonomous University, Mexico.
[2)] Ms. Eng. Student, Postgraduate in Agricultural Engineering and Integral Use of Water, Chapingo Autonomous University, Mexico.
[3)] Researcher and Professor at the Irrigation Department, Chapingo Autonomous University, Mexico.
nvelazquez@taurus.chapingo.mx

[*)] Instructor and Supervisor

1 Introduction

Population growth, urbanization, uneven distribution of land, shrinking farms and the continued impoverishment of third world farmers have contributed to reducing traditional production in critical areas. One of the basic needs of the human being is food and due to these changes the social and agricultural sector creates the need to find new ways to increase their productivity. Parallel to achieve appropriate conditions to give life and generate the reproduction of plants of any purpose, whether edible (such as fruit and vegetables), or ornate in a greenhouse. According to Sammons et al., (2005) optimal physicochemical conditions of temperature, carbon dioxide and humidity (among others) are needed. This leads to workers being exposed to risk conditions, which may affect their health (Acaccia et al., 2003), due to exposure to chemical products over extended periods of time. Another important fact is the use of pesticides that are too intense to penetrate latex clothing, and even rubber gloves in a half-hour of work (Sammons et al., 2005).

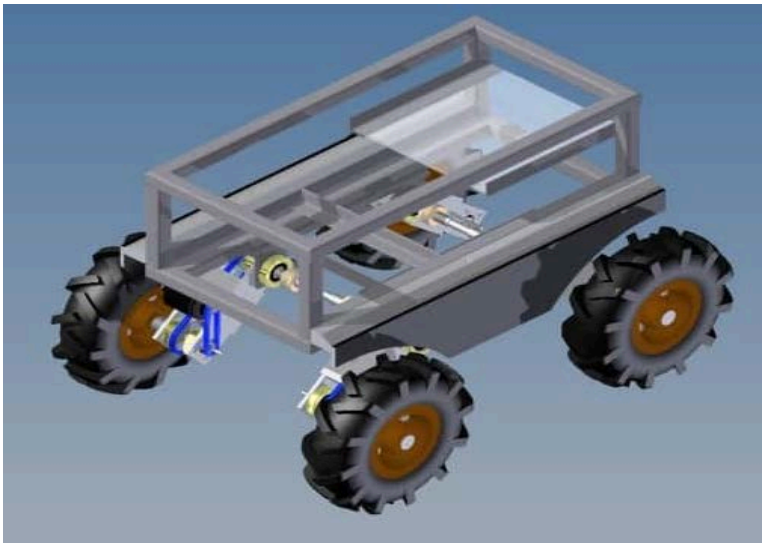


Figure 1: Voltan frame structure

In this work, an unmanned light vehicle that can move in spaces for agriculture was designed, built and evaluated. In order to control the vehicle and an autonomous navigation system was developed using OpenCV and Visual Studio libraries. As for the mechanical design of the robot a 3D CAD software was used.

2 Mechanics and Power

Manufacturing and selection of components

The main parameters for choosing motors and designing the main parts of the robot were, maximum velocity (1.6 m/s) and a maximum load (50 kg). This allowed for the selection of motor and designing the chassis of the robot.

Tires

Based on technical and safety requirements we selected the mobile components of the vehicle. As a result, agricultural type tires of 26 cm diameter were selected (Figure 2).

Figure 2: Tires used in the vehicle

Power Support Arms

To have enough clearance of the vehicle four arms where designed. The arms were manufacture with aluminum to keep the vehicle structure weight as light as possible (figure 3). These arms were used to transmit power from the motor to the axle of the tires. The power transmission was done using a chain and a sprocket and two motor were used. The vehicle was designed to be 4WD.

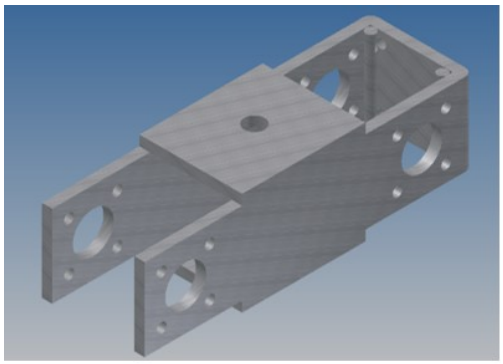



Figure 3: Arms, bush and support of the tires

Drivers and motors

A power wheels toys vehicles motors (Figure 3) were selected due to its convenient characteristics, such as price, torque and energy consumption (Table 1).

Table 1 Motor features

Voltage	**12 V**
Angular Speed	**139 rpm**
Power	**490 W**
Current consumption	**2-70 A**
Maximum load	**59 kg.**
Torque	**33.5 Nm**

Figure 4: Electrical motors used in the vehicle

The selected motors current consumption is high ranging from 2 A to 70 A in stop torque, that is when they are presented with a load with enough torque to stop the motor from turning. At its best performance, maximum efficiency consumes a current of around 10 A, this was the main parameter for selecting the drivers. The selected model is shown in figure 5.

Figure 5: Drive model IBT-3 50 A/5V-15V

Battery

The selected battery features are described below.

Table 2 Battery features

Voltage: 12V
Lithium Cranking Amps: 120
Lead-Acid Replacement Range: 7-9(Ah)
Case Dimensions: 4.21” (107mm) length x 2.2” (56mm) width x 3.35” (85mm) height
Operating Temp: 40 - 140° (F)
Weight: 1.2 lbs
Max Charge Rate: 10ª

Figure 6: Batteries used in the vehicle

Frame (structure)

All the instrumentations and controllers were mounted into the main structure (Figure 6), which was built with aluminum. The frame was assamble with screws and nuts, to allow more practical disassembly in case it is required for future modifications. All the frame was covered whit acrylic to protect the electronic components from dust and water.

Figure 7: Mainframe (structure) of the vehicle

Main features of our robot are listed below.

- Frame
- Power unit
- Electrical systems
- Transmission systems
- Tires
- Attachments

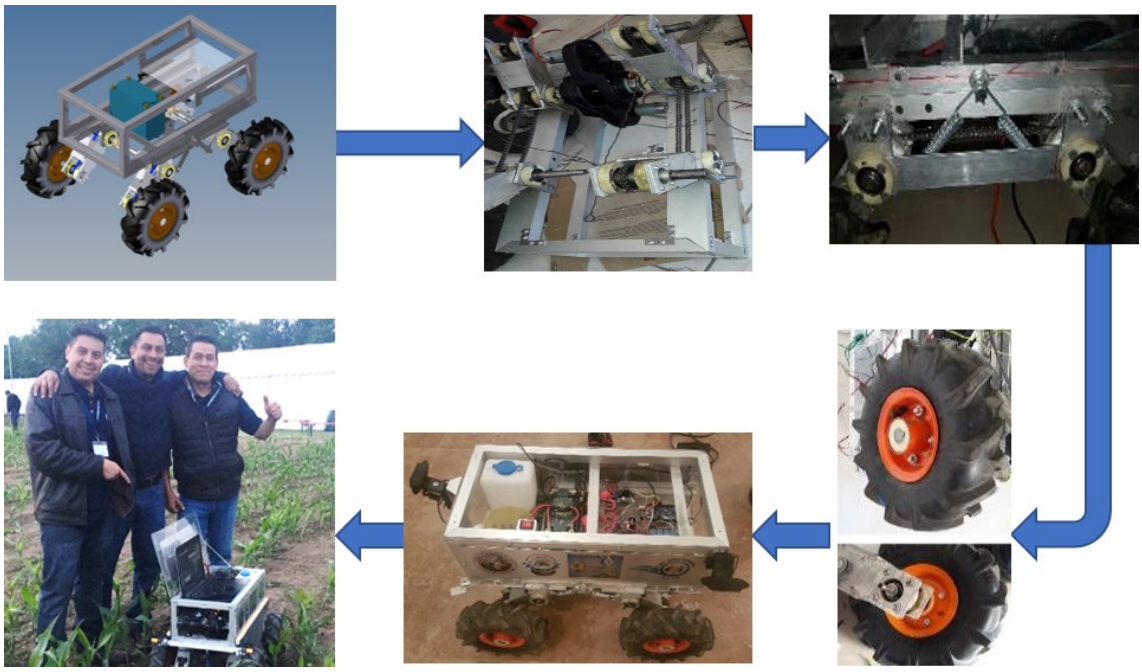


Figure 8: Robot construction process

3 Controller Architecture

Figure 9: Control remote in the vehicle

Control of the robot. Vehicle control was developed using the C ++ programming language and the Open CV libraries. We worked on each sensor and actuator independently, after getting the proper operation, every element was integrated into two Arduinos.

The control of the robot can be done manually using a Sony dual shock 4 joystick (Figure 9), which is intuitive, ergonomic and easy to get or replace, or automatically with our developed algorithm and computer vision system.

In the developing of this robot we are focusing on three tasks: crop monitoring, seed sowing, and spraying. The control algorithm is shown in figure 10.

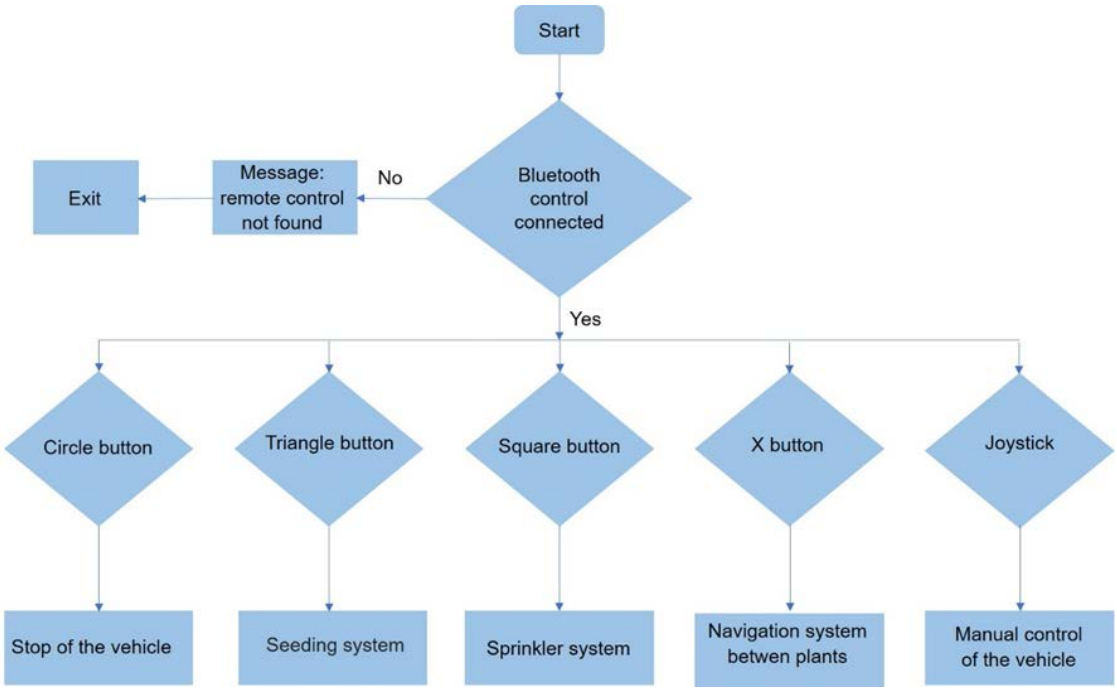


Figure 10: Flow diagram of the main control of the robot

The algorithm was developed to start manually, therefore the program detects whatever the Bluetooth joystick is connected or not, in case the control is not connected the program displays a message and closes. On the other hand, if there is communication with the joystick, the program waits for the buttons to be pressed.

The left lever is used for robot movements; when moving upwards the vehicle moves forward; when the lever is moved downwards the vehicle moves backward, the movements to the right and left generate a displacement in those directions with a wide radius of rotation, that is, an open turn. To make turns on its own axis, the R2 and L2 buttons are used. R2 button generates a turn to the right and the L2 button a turn to the left. The circle button is used to stop the vehicle completely, until a movement button is activated.

When the vehicle is moving in manual mode, we can activate actuators for spraying and sowing seeds or fertilizer. The squared button is used for the sprinkler and the triangle is used for seeds. With the X button we activate the autonomous system of navigation between plants, this system allows the vehicle to identify the midpoint between crops lines to move autonomously, while the camera detects the plants and the computer calculates the trajectory. To detect the end of the row we are using ultrasonic sensors, which detect the presence of plants on the sides of the vehicle, once they do not detect anything for a certain time, the vehicle makes the turn and advances to the next line. This can be used to make applications autonomously.

Artificial Vision System:

The programmed artificial vision system allows the vehicle to navigate autonomously between plants, this was done by implementing a type of control known a visual servo, which uses artificial vision to determine the movements of motors.

The algorithm consists of detecting the two rows of maize that are aside the vehicle. Then we calculated the center of mass of these two rows which happened to be the exact center of the row. Finally, through serial communication the main computer sends signals to the Arduino for motor control.

In Figure 11 it is shown the operation of the control system for autonomous navigation. The system calculates the coordinate of the center of mass and compares it with established values. These values determine if the vehicle must turn to the right, to the left or continue straight. Each action is sent it to the Arduino trough serial communication and a single character is used for each one. This is very important to not saturate the Arduino with all the information for each coordinate that is being calculate, but only when it required to do an action.

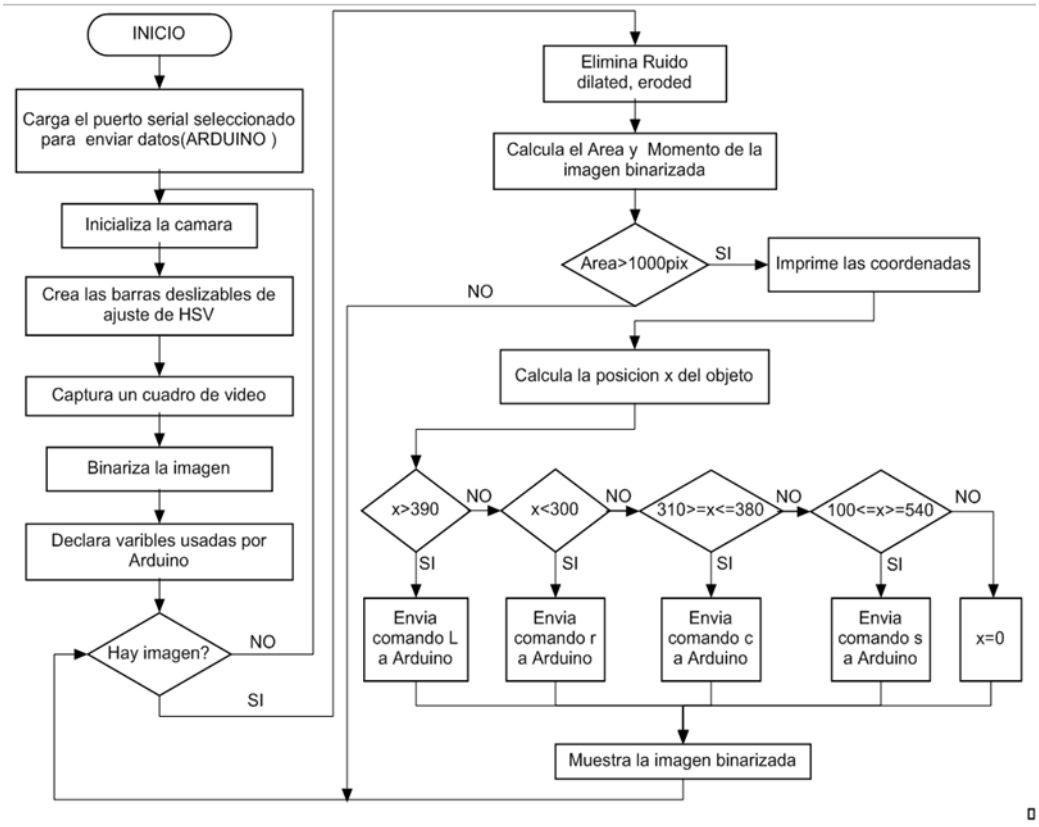


Figure 11: Flow diagram of the servo visual control implemented

3.1 Computer and other Hardware

We are using a Dell notebook with intel core I7 processor, 16 GB RAM and 250GB Hard disk. Two Arduinos Mega 2560, two ultrasonic sensors HC- SR04 compatible with Arduino, an incremental encoder, an IMU to measure the turning direction.

3.2 Software and strategy

We used Microsoft's Visual Studio 2013 and the OpenCV library in his version 3.0. and 3.1 for the first task. We developed a software that adjust automatically to sun light changes and detects the green color of the crop. The advantages offered are many, such as saving CPU resources, fast real-time response, low cost, excellent autonomous navigation between crop lines; however, it is limited by the presence of obstacles, which cannot be detected yet. Using this software, we can detect the two sides of the row where the robot will navigate (Figure 12). Once the two rows are detected the center of mass is calculated, this coordinate happen to be the exact center of the crop line. When the crop line is curve as in first task we just must calibrate left and right velocities for each side of the vehicle. We used the same software for task two combined with our mounted ultrasonic sensor to count the crop lines. For turning we program an Inertial Measurement Unit (IMU), which we tested and worked well.

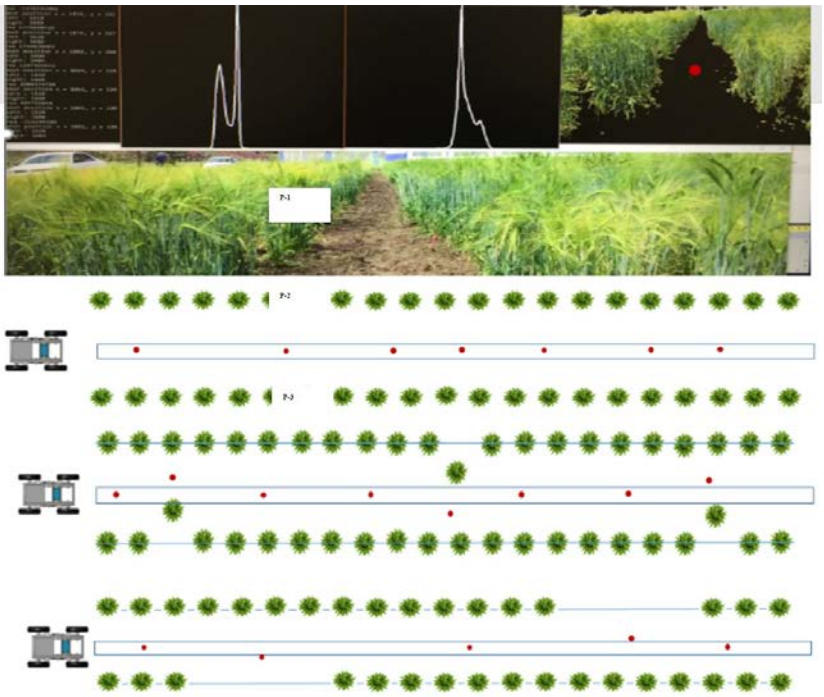


Figure 12: Information displayed by the servo visual control software.

For task 4 and for we used LINUX and ROS as the operating system. For navigations we used the same algorithm as task one and two. For the detection of the weeds we include and algorithm that detects only the red color and counting the red color objects we were able to detect if there were more red ties or more blue one. We used the same strategy for task 5.

3.3. Sensors

Encoder EE-SX1103

As a sensitive element of the vehicle, it was decided to use the optical encoder.

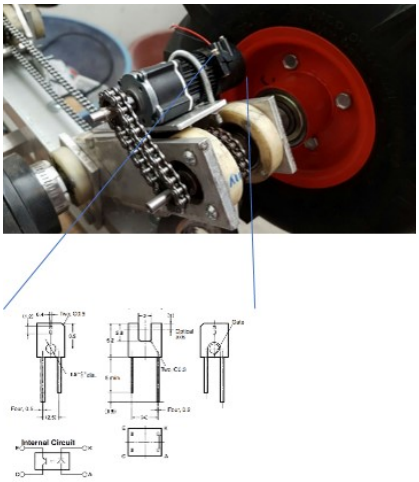


Figure 13: Encoder used to control the velocity in the vehicle.

Ultrasonic sensor.

The ultrasonic sensor (figure 13) was connected to the vehicle. This sensor as mention above was used to detect the end of the row and also to count the crop lines.

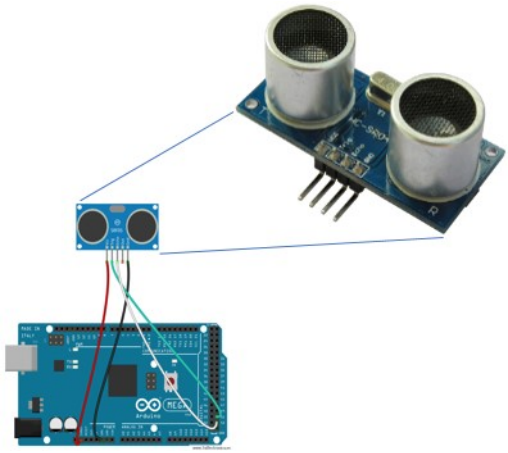


Figure 14: HC-SR04 ultrasonic sensor installed in the vehicle

MPU-9250

Figure 15: MPU-9250

MPU-9250 (Figure 14) is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. Hence, the MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor™ (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU6515. With its dedicated I2C sensor bus, the MPU-9250 directly provides complete 9-axis MotionFusion™ output.

This sensor was used mainly to measure the turning angle of the vehicle at the end of the crop line.

2 Results and Discussion of Machine Performance

We have great performance of the navigating system between the crop rows, however our ultrasonic sensor did not detect the end of the row, therefore we were not able to initialize the turning routine and go into the next line. Same thing happened in task two. At the end we only could made one line for each task, however, even our robot was very wide, the autonomous navigation was great.

As for tasks 4 and 5, we changed to UBUNTU 14.04 operating system and ROS. We did some test, and everything was working well. However, we did not realize that the difference between Windows OS and UBUNTU OS was the number of information we were sending to the Arduino. For UBUNTU OS and ROS every single coordinate that was being calculated was send it to the Arduino and then it had to decide whether to turn or to continue straight. However, the information send it was to many and the Arduino was not able to handle it on real time. Therefore, the reaction time was very slow, and the robot was out of trajectory. As

a result of these we were not able to navigate accurately into the crop lines and could not make it to the first patch of weeds. However, in the tests we performed, we verified that the algorithm for detecting the ties was working well.

3 Conclusion

This event is always a great opportunity to learn from other teams and test our prototypes. In this occasion we realize that we must improve our instrumentation, specifically for the detection of the end of the row. We considered that best option would be to use the same Sick LIDAR sensor that most team have or to look for other solutions that can keep our robot as cheap as possible. We are already working on that.

We also need to improve the implementation of ROS for the control of our robot and we realize all the advantages this software has. For our team was a great honor to participate for the second time in the Field Robot Event 2018 and represent Mexico and Chapingo Autonomous University.

4 References

Acaccia, G. M.; Michelini, R. C.; Molfino, R. M. and Razzoli, R. P. 2003. Mobile robots in greenhouse cultivation: inspection and treatment of plants. ASER 2003, 1st International Workshop on Advances in Services Robotics, Bardolino, Italia

Sammons, P. J.; Furukawa, T. and Bulgin, A. 2005. Autonomous Pesticide Spraying Robot for use in a Greenhouse. Australasian Conference on Robotics and Automation. Sydney, Australia. p. 1-9.

Team Support

All expenses to participate in the FRE 2018 were cover by Chapingo Autonomous University. Until now we don’t have any sponsor, however we received the CLASS foundation support last year (2017).

Sparrow

Grischka Ulrich[1], Nils Lüling[1], Michael Steppich[1], Lucas Jahn[1], Robin Löffler[1], Georg Feyrer[1], Matthias Schlötterer[1], David Reiser[1]*,

[1)] *University of Hohenheim, Institute of Agricultural Engineering, Stuttgart, Germany,*

[*)] *Instructor and Supervisor*

1 Introduction

As efficiency and time management becomes more and more important, robots start to take over in many areas of our daily life. Automatic vacuums and lawn mowers are just two of the most common examples of this development. Furthermore, the trend of automotive vehicles has become a huge research domain. In agriculture this progress was established. Through the robot fleet, “MARS” Fendt started research in planting corn with autonomous robot units (Fendt, 2017). To promote research with farm robots, the field robot event is hosted every year to motivate students from all over Europe to develop new ideas and solutions for autonomous agricultural machines. The attendance of competing teams is increasing, which might help to develop the farming machines of the future. Small autonomous agriculture robots could help to overcome problems like soil compaction or the shortage of skilled employees. To maximize output and minimize input, it is very essential to work site-specific. With a swarm of robots, this could be realized easily.

For this year’s event, the student team of Hohenheim improved the robot “Hefty”, which was invented last year. The improved robot version was called “Sparrow”. The authors’ work included mechanical, but also software related upgrades. This was necessary to be able to fulfil the given tasks by the Field robot event 2018.

2 Power and Mechanics (Hardware)

As basic vehicle, a small 4-wheel autonomous robot with differential steering was used. The size of the basic robot platform is 500 x 400 x 150 mm (see Figure 1). The weight of the robot is around 15 kg and it is equipped with four motors with a total power of 240 W. The used motors use a metal gear system of 1:50 what results in a speed of 200 rpm by 1.17 Nm per motor. The motors were directly connected to the wheels of the robot (see Figure 1). The motors at each side were connected with a belt system to ensure, that both motors at one side can pull with full force at one wheel. This ensured, that at least two motors push the whole robot forward. Maximum driving speed with the attached wheel diameter was 0.8 m/s and the maximum static motor torque was 4.68 Nm.

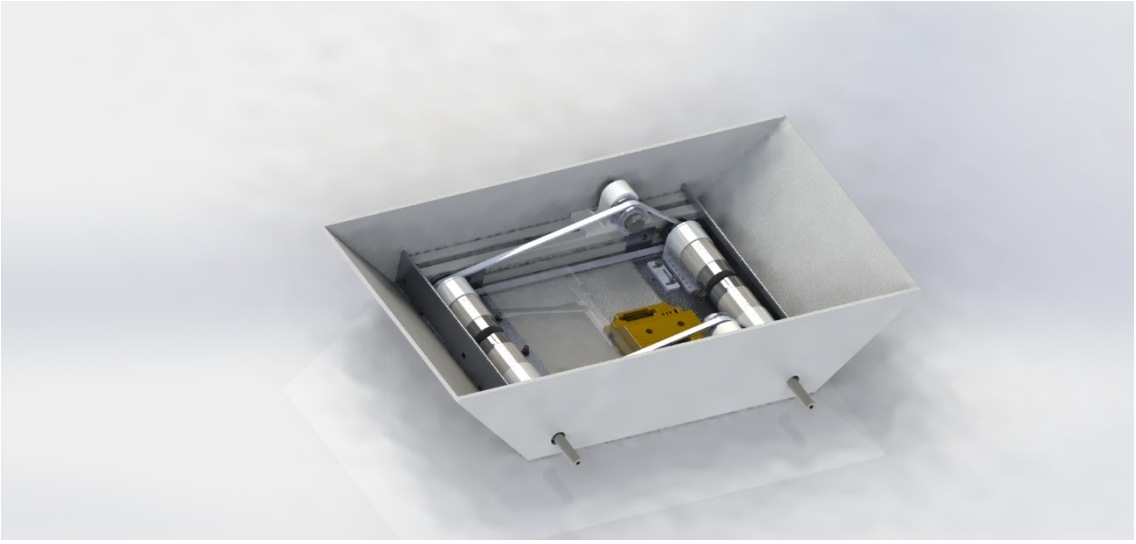


Figure 1: Mechanics of the motor belt system used for the robot sparrow

For cooling the whole robot system, two computer fans were attached to the top, giving the option to work also under harsh conditions like in direct sunlight and hot weather conditions. The air got sucked through the robot, by using an aperture at the front of the robot. This helped circulating fresh air inside the chassis. To avoid dust to enter, an additional air filter was attached. To optimize the cooling capacity of last year performance, the position of the two fans on top of the robot was changed and placed rather to the end of the vehicle. The air, which was sucked in at the front, flowed through the whole robot and cooled all components such as the motors and microcontroller. Another obvious change was the rearrangement of the control buttons to the back of the robot “Sparrow”. This was necessary to ease the lifting of the cover and avoid cable kinking. As a result of all these changes, the cover had to be rebuilt (see Figure 2).

Figure 2: Rearrangement of control buttons and fans

Inside the robot, the open laying belts, which connected the front and the rear drive were problematic, because they could damage wires and cables. This problem was easy to solve by two Plexiglas covers placed above the belts. Another issue was caused by the tension rollers the drive belts were mounted on. The new material and the proper bearing allowed a correct guideline and tension.

The general power to move the machine was supplied by a 14,8 V LiPo battery. Any additional power requirements such as for the microcontroller or the camera were provided by an 11.1 V LiPo battery (1000 mAh battery). All the power supplies were equipped with battery keepers to supervise charge status and avoid deep discharging. The whole wiring inside the robot has been structured and the complex electrical connection of all components was simplified. Apart from these major changes a few smaller things had been optimized. This, for example, included to cut off the cooling rip on the back of the tablet and the rearrangement of the microcontroller due to lack of space.

3 Sensors and controllers

The robot was equipped with different kind of sensors. Each motor was connected to an encoder (Pololu, Las Vegas, USA) with a 64 tick's resolution per motor rotation what corresponded to 3200 counts per rotation of each wheel. For estimating the initial state of the robot, an Inertial Measurement Unit (IMU) was fixed to the frame of the robot. The used IMU was a VN-100 (VectorNav, Dallas, USA). For sensing the environment, one light detection and ranging (LIDAR) laser scanner was mounted at the middle of the robot at a height of 0.3 m above the ground level. It was mounted that the complete sight of 270 degree of the laser is usable. The used sensor was the TiM 571 2D-LIDAR (SICK, Waldkirch, Germany). The LIDAR was attached to a variable mounting platform, able to tilt and change the height of the sensor. The motor encoders, as well as the IMU and the LIDAR from Sick were still used in the same way as last year. Only the IMU was probably placed and fixed at the bottom of the chassis.

A tablet, connected to all necessary devices with a USB 3.0 Port, controlled the whole robot. The Tablet had an integrated battery with a runtime of 8 hours. It is using an Atom x5 1.44 GHz processor, 4 GB RAM and a 64 eMMC high speed storage. The touchscreen works with 1280x 800 pixels (10").

As motor controller the Roboteq SDC2130 (Roboteq, Scottsdale, USA) was used. This dual channel motor controller is able to power up to 30 V and 2x 20 A maximum, sufficient for the used motors. The motors at each side were connected to each other, so that they turned always in the same direction. In addition, the encoders were attached directly to the motor controller and the update signals were sent to the control tablet. For the remote control and direct user interaction a standard X-Box Joystick Logitech F710 (Logitech, Lausanne, Switzerland) was connected to the computer with a Bluetooth dongle. For activating the magnetic valves and a siren, four USB driven relays were used. The IMU used a RS232 protocol for communication, while the laser scanners worked with TCP/IP over Ethernet with an attached adapter.

The tablet runed with Ubuntu Mate 16.04., including ROS as basic software structure for programming, visualization and user interface. For remote control a hotspot was integrated, enabling to control the computer tablet with SSH and remote desktop connection via WLAN.

For competing at the field robot event, the most important task of the robot was the autonomous row following of the system. To set up this structure, a mode changer was implemented (Blackmore et al., 2002). This included one mode for navigation inside the row and one mode for headland turning. It was possible to overwrite the autonomous mode with a user joystick input and to separate the program code to different tasks. In general, three modes were used:

- User Mode
- Headland Turn
- In Row Navigation

For each mode, the motor controller subscribed to a separate speed message, provided by the joystick, the laser scanner or the odometry with a point-to-point navigation. The ROS Middleware was used to set up the programs of the robot (Quigley et al., 2009). The whole programming was performed in using C/C++ and Python packages and nodes.

4 Field Robot Event 2018

The event took place during the DLG -Feldtage from 12th to 14th June 2018 at Bernburg - Strenzfeld in Sachsen- Anhalt, Germany. The prepared fields were in good shape and kept in this way by the staff members, who for example replaced damaged plants or removed the tracks of larger and heavier robots. The weather conditions were steady, partly cloudy to quite sunny with no heavy rain and temperatures about 20 °C. All rules, regulations and the tasks were communicated to all competitors in advance of the event.

4.1 Task 1 “Basic Navigation”

The navigation through a maze row was the first and very essential task. In order to do that the robot had to detect the rows and switch into the next one fast enough, to complete most of the field within the given time. As an additional difficulty, the rows weren’t seeded straight but curved. The navigation in the rows was realized by the LIDAR sensor, which scanned its environment and provided relevant position information to navigate the vehicle with a PID regulator. To turn at the end, goal points were estimated which the robot tried to follow.
For detection of the rows, the special object oriented structure of ROS was used to define a flexible and adjustable program, able to perform in different agricultural environments. The starting point for the navigation algorithm was the raw data of the used LiDAR sensors. The aim was to define a goal point in the middle of the row, what

was used to guide the robot for navigation. To reach this goal, several filter needed to be applied, to gain a position out of the data (see Figure. 3).

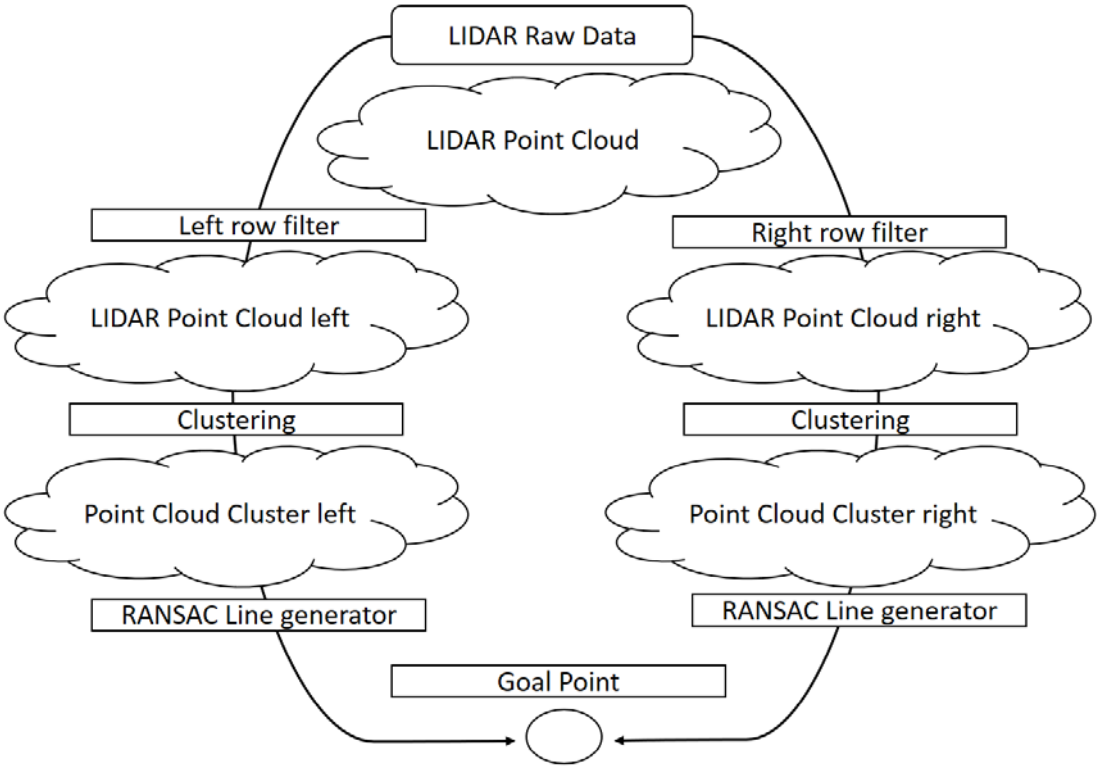


Figure 3: Description of the row filtering algorithm used for the Field Robot Event 2018.

As the LIDAR Data was received in polar coordinates, they were first converted to a Cartesian coordinate system. To get rid of all unnecessary points, a box filter was applied, removing all points with a higher distance to the robot. So it could be assured, that just the two rows directly in front of the robot were detected with the sensor. Now the resulting points were separated in two different point clouds, one corresponding to the left row and one-point cloud for the right row of the robot. Each of this point clouds were filtered, to remove noise out of the sensor data, before searching for the biggest cluster inside this selected area. The best fitting cluster was used to estimate the row on each side. Out of the two separated point clouds, the RANSAC row detection algorithm could be applied to each row separate. Out of the difference of both lines, a navigation point was estimated, taking the orientation and the position of the lines into account. Figure 4 is showing the point cloud of the front laser in combination with the base coordinate system of the robot (base_link). The next goal point (row point) coordinate system was defined out of the two filtered point clouds and the resulting lines estimated by the RANSAC line detector.

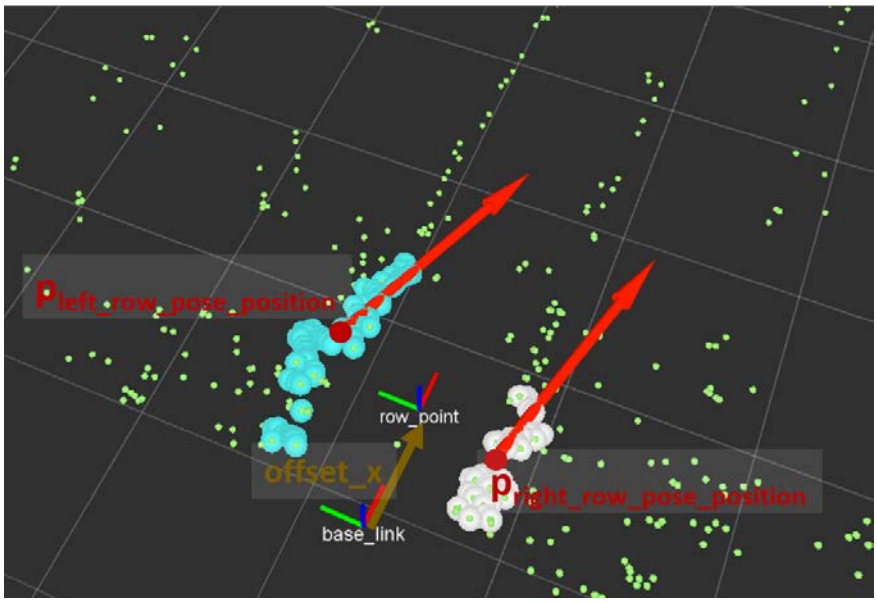


Figure 4: Description of the rows and the resulting row point. The base_link coordinate system corresponds to the robot center.

The change for the mode from row navigation to headland turning was performed when no points in a distance around 1.5 m ahead of the robot were detected.
The headland turning used goal points to estimate the next position for the robot and a point follower addressed the necessary speed signal. The turning was performed with three goal points. As soon as the last goal point was reached, the mode changed again to row navigation. For evaluation of the algorithm, the sensor outputs and the filtered data could be shown in real-time in the RVIZ visualization tool (see Figure 4).

As soon as the headland was detected, odometry was restored and resolved by the use of wheel encoders and the IMU data to move to the next row. The headland turns were defined by a list of row points, which could be changed for every single turn. Therefore, the same program for task 1 and 2 could be used. The only changes were made in waypoint lists before the start of task 2.

It contained mode changes to enable the navigation with different modes such as “Row Follower”, “point follower” or “Joystick”. Only location relevant parameters as for example driving speed or turning angle were adjusted. The driving speed within the row was higher compared to the other teams and the steering worked mostly accurate, so barely any maze plants were destroyed. Even though the main part of the program wasn’t modified, the turning movement at the field end was a problem. During the contest, Sparrow wasn’t able to make the second turn and got stuck in a program error. Even manual interference couldn’t solve this problem quick enough until the given tree minutes were up. We assume that the problem was caused by the lack of power for turning. The engines might not have been powerful enough to steer the robot to the estimated coordinates under all ground conditions. This lead to the fact that, the

machine's turning angle sometimes was too large and in other cases to small. Therefore the correct settings of the headland turning parameters were hard to achieve.

4.2 Task 2 "Advanced Navigation"

This task simulated more realistic field conditions, by relatively straight rows with some gaps within them. This afforded a higher accuracy in guidance to follow the rows because the robot had to decide whether it is already at the rows' end or not. In addition, the switching of rows had to be done along with a certain pattern that was given upfront the competition's start. This year's pattern was: S-2R-3L-1R-2R-4R-F. The points for turning had to be adjusted and were noted down in text files.

Sparrow did the first turn without problems, even though at the second one (3L) it had issues and turned into the wrong direction at first. We couldn´t figure out why this problem occurred. Setting it back to the last correct position and restart the program cost much time. Nevertheless, it worked with the second try and Sparrow successfully continued to follow its path. The 1R rotation caused trouble because Sparrow didn´t detect the row and by setting it back to the last position the time was up.

4.3 Task 3 "Selective Sensing"

The goal of this task was to detect weeds, which were represented by red and blue golf tees within the maze rows. Nine "weed patches" were distributed throughout the course with different combinations regarding the number of red and blue tees in one patch. Those weed patches sized 25 x 25 cm with a total number of 10 tees. If the weed patch contained more red golf tees, the robot should give one acoustic signal, in the case of more blue tees two signals were required. The camera to detect the tees was mounted on top. An additional program to analyze the captured pictures was created and ran simultaneously next to the row follower program. Two of the three weed patches in row number one were detected correctly. At the end of the field issues with the turning occurred again so that unfortunately we had to quit the task.

4.4 Task 4 "Soil-engaged weeding"

Assignment four was aimed at the attempt to mechanically reduce weeds with some sort of tillage tool (see Figure 5). The weeds were simulated by red golf tees, that were placed randomly in a square spot of about 25 x 25 cm between the maize rows. Due to energy saving, this tools activation had to be in the range of the weed patch, with a tolerance of 10 cm. A brush with a collecting bucket was constructed and installed at the robot's front. This container had plenty of holes to lose as much as possible of the collected dirt. Logically these holes were fairly narrow in order not to lose the picked-up golf tees. The whole construction could be lifted by two servo motors. The source code

of task three was adjusted. First, it lowered the pin collector, then it started the motors of the brush. We lowered the driving speed before the task because we wanted the robot to drive slowly over the pins and expected a better collection rate. It turned out that this was a crucial mistake as we lost the necessary driving torque in order to go forward continuously. By setting the robot back on track we must have made a prohibited move so that we were disqualified. This was really upsetting, due to the fact that no other team's collection rate was as high as ours. Our robot picked 9 out of 10 pins at the first weed patch.

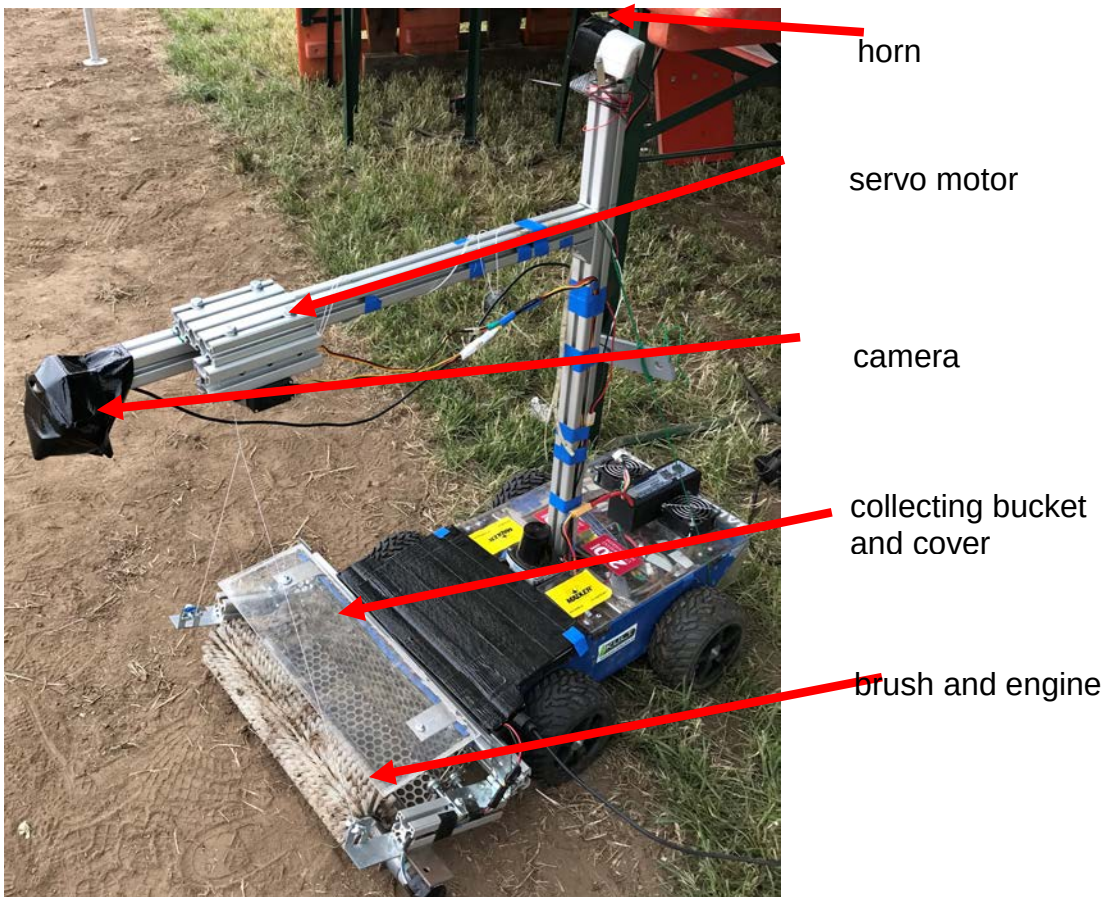


Figure 5: Robot set up for task number 4

4.5 Task 5 "Freestyle"

Here creativity and fun ideas were required to convince the jury of a useful freestyle function the robot would perform. As we didn't have any time left during preparation and were short of any convincing ideas, we decided as a team not to participate in that contest.

5 Conclusion

At the 2018's Field robot event, the University of Hohenheim competed with an upgraded version of last year's robot "Hefty", named "Sparrow". The team was able to participate in four out of five tasks and achieved not perfect but satisfying results.

Even though the team was mostly pleased with the robot's performance, several weaknesses have been detected. During the event, it often seemed like the robot was short of power. A couple times it looked like the engine power of 300 W was not sufficient. Especially at the headland turning and on the uneven ground this was obvious to see and the robot got stuck easily. The small chassis clearance of just 4 cm could be enhanced by bigger wheels, which would require additional power as well.

Furthermore, a few issues with the weak WiFi occurred. The used receiver worked on a 2,4 GHz bandwidth. With this device, it was not possible to transfer data in real time. The connection was laggy and the signal was delayed. This problem was enhanced by the processing power and RAM of the tablet. With its configuration, it was tough to run several programs, such as image processing and data evaluation simultaneously to ensure a steady performance of the vehicle. A 5 GHz WiFi system, which allows a long-distance connection, and a more powerful computer for data processing would be an improvement.

Program modifications should be made as well. After the robot detected the end of the maize field, it stopped for several seconds until it started to move into the next row. As for all the tasks, there are just three minutes available to fulfil it, this is an unnecessary loss of time.

Even with the describe flaws, the robot of the University of Hohenheim contributed well at the 2018's Field Robot Event. We are looking forward to fixing these issues and keep improving the machine to be able to achieve better results next year.

All the projects that were shown at the field robot event, are equipped with modern technology and afford a great understanding to use them. If this knowledge is extended through the next couple years, it could lead into major innovations and changes on future farms.

However, this year was a great success for our team, we looking forward to compete next year again and using the gained knowledge for better results.

6 References

Blackmore, S., Fountas, S., Have, H., 2002. A proposed system architecture to enable behavioural control of an autonomous tractor., in: Zhang, Q. (Ed.), Automation Technology for Off-Road Equipment. 2950 Niles Road, St. Joseph, MI 49085-9659, USA, ASAE, pp. 13–23.

Fendt, 2017. MARS: Robotiksystem zur Aussaat und exakten Dokumentation, Pressebeitrag, Verügbar unter: https://www.fendt.com/at/fendt-mars.html (Stand: 11.06.2018).

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Mg, A., 2009. ROS: an open-source Robot Operating System, in: ICRA Workshop on Open Source Software. p. 5.

Acknowledgments

We want to thank Balluff GmbH, Mädler GmbH, Robert Bosch GmbH and the University of Hohenheim for sponsoring this project. Without their consulting and financial aid, this team would not have been able to participate in the contest.

We give thanks to all those, who contributed to the organization and the jury of the 2018's field robot event. We owe them a fair and interesting contest.

Abbreviations

DLG	Deutsche Landwirtschaftsgesellschaft
IMU	Inertial Measurement Unit
LiPo-Battery	Lithium- Polymer Battery
MARS	Mobile Agricultural Robot Swarms
ROS	Robot Operating System

LAZER MAIZER

Matti Siponen[1], Albert Georgs[1], Timo Mauranen[1], Elias Ala-Kaila[2], Timo Birnkraut[2], Benjamin Nikolov[2], Timo Oksanen[1*]

[1)] *Aalto University, Department of Electrical Engineering and Automation, Finland*
[2)] *Aalto University, Department of Engineering Design and Production, Finland*

[*)] *Instructor and Supervisor*

1 Introduction

The goal of this project was to design and build an autonomous agricultural field robot from scratch, implementing all of the software on our own. The project started out as a joint project between the course ELEC-E8004 Project Work for Aalto University's Masters's programme at the School of Electrical Engineering and MEC-E5002 Mechatronics Project in the Master's programme in Mechanical Engineering. The purpose of this project was to create an autonomous robot. The latter part of the project consisted of preparations and participation to the annual Field Robot Event, this year held in Bernburg-Strenzfeld, Germany [1]. At this stage, the main functionalities of the robot were dictated by the rules and regulations of the competition [2].

The work was roughly divided into hardware and software, but all group members got to familiarize with both segments. The project started in January 2018, the first months were focused on design and teaching, after which the implementation phase started, i.e. the construction and ordering of the parts, assembly, and software development.

After this the focus was switched to competition task specific design and part manufacturing, software development and algorithm tuning. There were several important milestones during the project, most important of which were the Mechatronics Circus, the Final Gala of the Project Work course and the competition. In retrospect, although we didn't manage to reach all the goals set at the beginning of the course, the main objectives were completed and we had something to show for our work at all of these main events.

This report describes the different stages of the project. In chapters 1 and 2 the mechanics and hardware are described; chapter 3 focuses on the software followed by discussion and conclusions in chapter 4.

In conclusion, this project has been demanding and time consuming, but also very interesting and teaching. It is safe to say that all of us involved have gained valuable knowledge in our coming lines of work and also gotten a broader perspective on project handling and team work in practise.

2 Mechanics and Power

This chapter discusses the components of the robot and how the mechanics work. Only a few parts like the cut-offs and the batteries are used from the robot built on the same project in 2017. The most important parts of the robot are listed in table 1.

Table 1: Main components used in the chassis and drivetrain.

Component	**Count**	**Purpose**	**Features**
Electronics box	3	Protect the electronics from the environment	Waterproof and dustproof
Handle	1	Ease of transport	From side to side
LED mast	1	Mount for LEDs	1150 mm from the ground
Axle	2	Direct motor power to wheels	380mm wide, differential
Steering servo	3	Turn the wheels left/right; lifts berry grabbers up and down	35kg/0.15 (7.4V) torque
Wheels	4	Produce linear movement from motor movement	Heavy-duty, off-road
DC motor	2	Provide the driving torque for the wheels	Brushless, 200 W
Gearbox	2	Tune motor rotation speed and torque for wheels	38:1 ratio
Driving belt, pulleys	2	Deliver driving power to the axles	1:1 ratio

Due to a tight schedule, it was important that as many off-the-shelf components as possible would be used. 2 millimeters and 4 millimeters thick sheet metal parts, such as bumpers and the main part of the equal force support system, were ordered from a company called Laserle, which is one of the sponsors of the team. The chassis of the robot as well as the maintenance stand were built from aluminium extrusions that were ordered ready cut to the right lengths and just needed to be assembled.

2.1 Axle Modules

Axle modules are needed for steering and for transmitting the power of the motors to the wheels of the robot. It is very time consuming to design and manufacture steering and driving axles. For this reason, off-the-shelf axles were chosen to be used. The drivetrain was designed to have two driving and steering differential axles, which enables the robot to have small turning radius, and to do “crab-walking” (diagonal shift without changing yaw). The two axle modules are identical, and mechanically the front and back of the robot are identical, so it performs similarly driving either way.

For driving the robot, each axle module has one TrackStar 21.5T "Out Law" Sensored Brushless Motor. A Banebots Gearbox with a ratio of 38:1 is mounted straight to the motor shaft. For power transmission, a belt drive is used because it was possible to realize easily with off-the-shelf components and it allowed to make the axle module compact by putting the motor-gearbox assembly straight above the drive shaft of the axle. The axle has a gear ratio of 2,53:1, and the gear ratio in the belt drive is 1:1. The maximum rotational speed of the motor is about 20 000 rpm and the diameter of the wheels is 205 mm, so the theoretical maximum speed of the robot is about 2 m/s. A 3D model of the axle module can be seen in figure 1. A 3D printed cover protects the belt drive from dirt and dust, but in the picture it is hidden to show the belt drive.

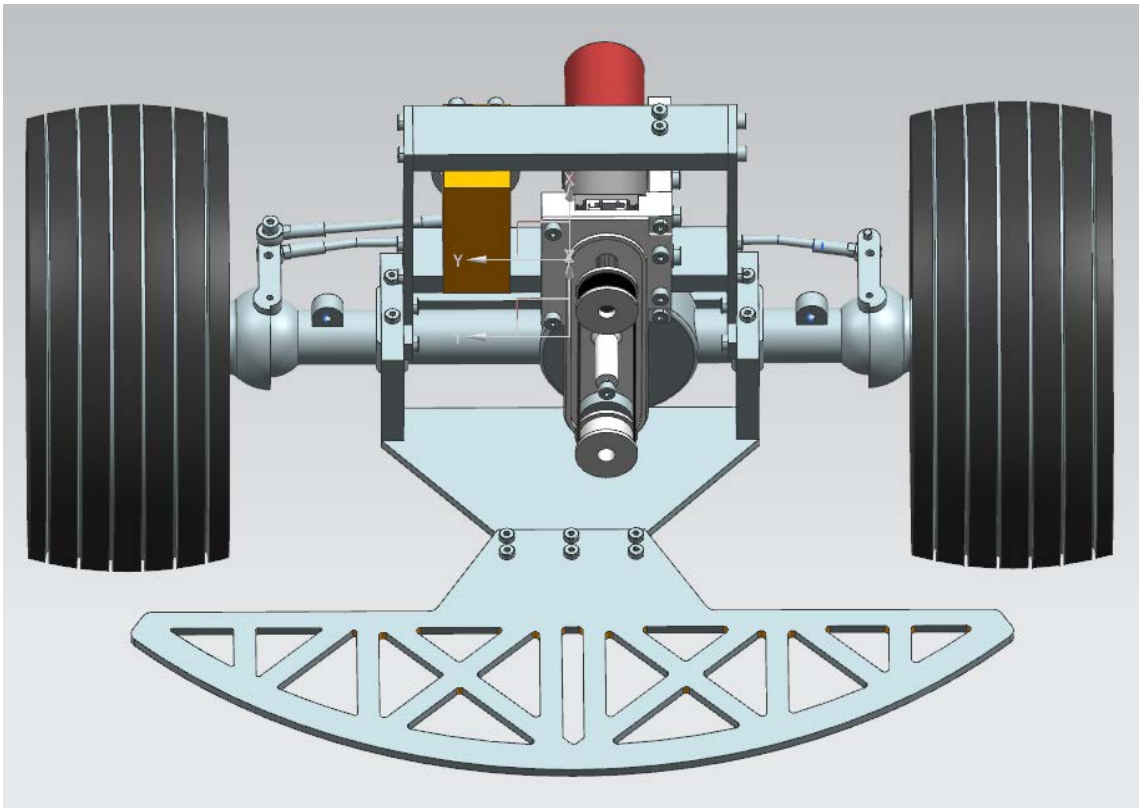


Figure 1: Front view of the axle module

Steering was realized by mounting a Savöx SV-0235MG digital servo next to the gearbox. Aluminium servo horns were used, and connection to the axle knuckles was done with M4 size threaded rods and ball joints. The maximum steering angle is about 35 degrees, the restrictive factor being the shape of the axle knuckles. The steering configuration can be seen in figure 2.

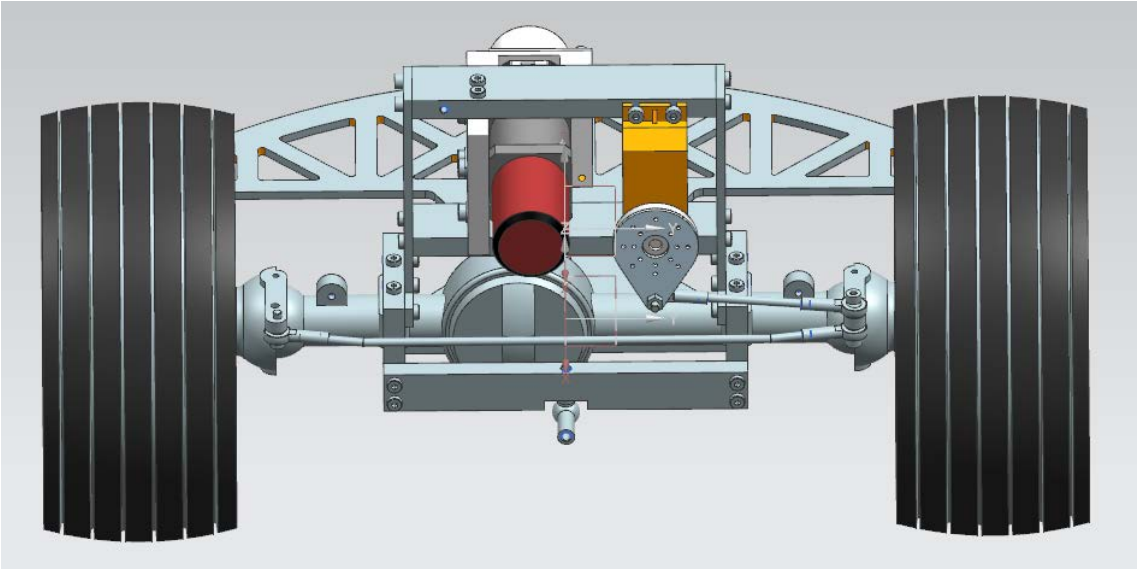


Figure 2: Back view of the axle module.

2.2 Equal Force Support System

The robot includes an equal force support system, displayed in figure 3. In case of an obstacle pushing one or more wheels off the ground, the support system compensates and pushes the wheels back to the ground to provide maximal traction and thus, torque. In the robot built on the same project in 2017, the system was realized using multiple triangular parts that had multiple degrees of freedom [8]. This year, the system was designed to be much simpler while still achieving the same result.

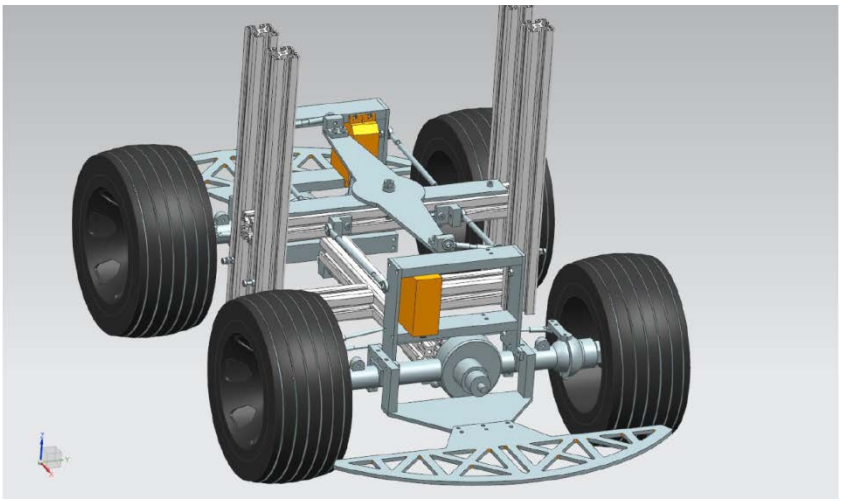


Figure 3: Equal force support system of the robot

The main part of the equal force support system is a plate that is pivoted in the middle with a bolt and two thrust bearings. Two rods and four ball joints connect the plate to the axle modules. When one axle module tilts, the equal force support system causes

the other axle module to tilt in the opposite direction, which is shown in figure 4. Rods connecting the axle modules to the chassis keep them in upright position. The operating principle of the system is shown in figure 4.

Figure 4: Equal force support system of the robot

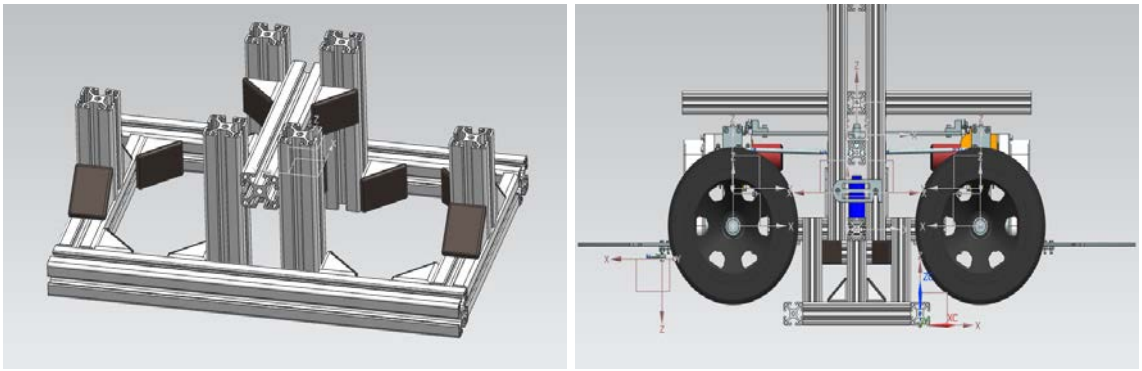
2.3 Local UI Mounting

Besides the autonomous mode, the robot is also equipped with a physical user interface (UI), which allows the user to control the robot without a remote controller and display important system status information. To simplify the case and ensure the waterproofness off-the-shelf boxes are used. Holes for the functional buttons and LEDs were drilled and a rectangular hole for the LCD was filed. The LCD and the LEDs show the current status of the robot. The buttons and the screen are connected to an Arduino board and allow controlling the different autonomous states of the robot. The UI is located on the edge of one of the robot's cover boxes, shown in figure 5, for easy access even during robot movement.

Figure 5: Local user interface [7]

The buttons in the local UI include buttons for powering up and shutting down the individual circuits for motion and computing, a toggle button for reversing, starting and stopping a navigational sequence, buttons for backwards and forwards, an emergency stop button and one extra button for an action that may be needed. The LEDs in the local UI box include individual power indicators for the circuits, battery charge warning LEDs for both circuits, an “alive” LED to indicate nominal functioning, and an extra LED for indicating something needed. Additional LEDs were mounted on the robot mast to signal reverse on/off, the turning direction for the upcoming turn, and an “action-LED” indicating with different signals the current action the robot is taking. The LCD displays the direction for a following turn, and the current status of the robot navigation.

2.4 Maintenance Stand



Figures 6 & 7: The maintenance stand and the robot on it

A maintenance stand is needed to allow easy maintenance and repair of the robot. Its main requirements are simple: it should be stable when the robot is on it, and it should lift the wheels of the robot off the ground. The stand was designed to be assembled from similar aluminium extrusions as are used in the chassis of the robot. As can be seen from figures 6 and 7, the final design is very simple but in the competition it was found to be very useful.

2.5 Weeding Trailer

The trailer is designed for tasks 3 & 4, Selective Sensing and Soil-Engaged Weeding. The main part of the trailer was built out of similar aluminium extrusions as the frame of the robot and the maintenance stand, so the stability is quite high. The goal for Selective Sensing was to detect how many blue and red tees were at different patches in the soil. There were different combinations regarding the number of the red and the blue tees. Initially JeVois smart cameras were supposed to be attached on the weeding trailer, but as turning with the trailer was very difficult, the cameras were attached to the front bumper of the robot. This way it wasn’t necessary to use the trailer in task 3.

For the task Soil-Engaged Weeding the goal was to collect small golf tees, which were stuck in the ground. It was required to avoid collecting soil. To reach this goal two berry collectors were attached to the weeding trailer. They were connected with a rope to a servo motor, which was used to lift them up and down. Although we were unable to

detect any weed patches during our attempt at task 4, we were able to detect a weed patch and pick up a few tees with the berry pickers in our tests.

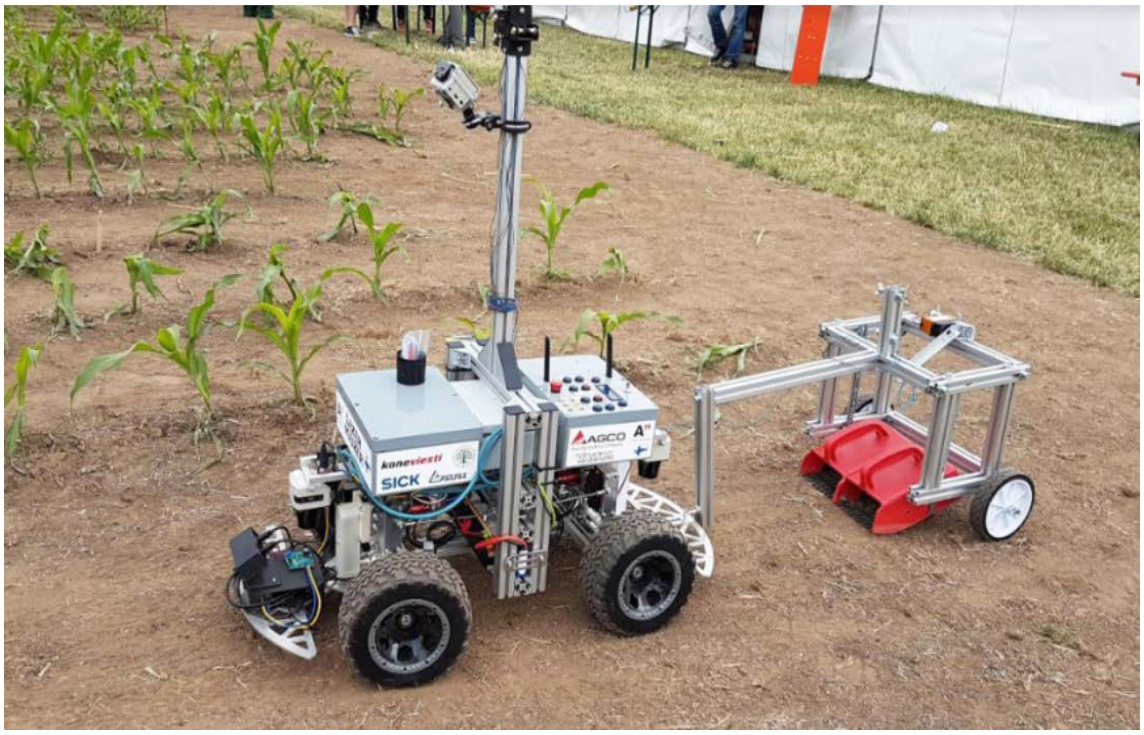


Figure 8: Weeding trailer attached to the robot [7]

3 Hardware

The basis of the hardware was the power distribution, which had to be efficient and reliable in order to provide the different components with constant power so that we wouldn't experience any sudden loss of connections due to lacking power supply. For this we needed one power distribution board and two Arduino shields, i.e. printed circuit boards that we designed and manufactured ourselves in order to fit our needs. The figure below presents a high-level connection diagram of the electronics design of our robot with these three clearly marked in their correct locations. The connections are marked with the same colour as the corresponding wires used in practice.

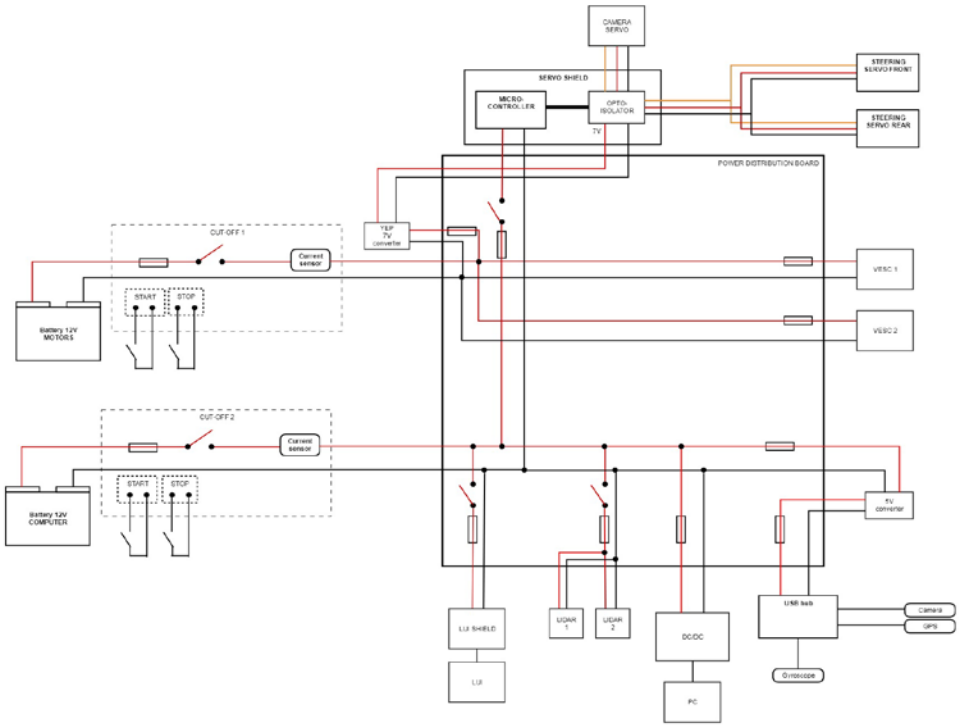
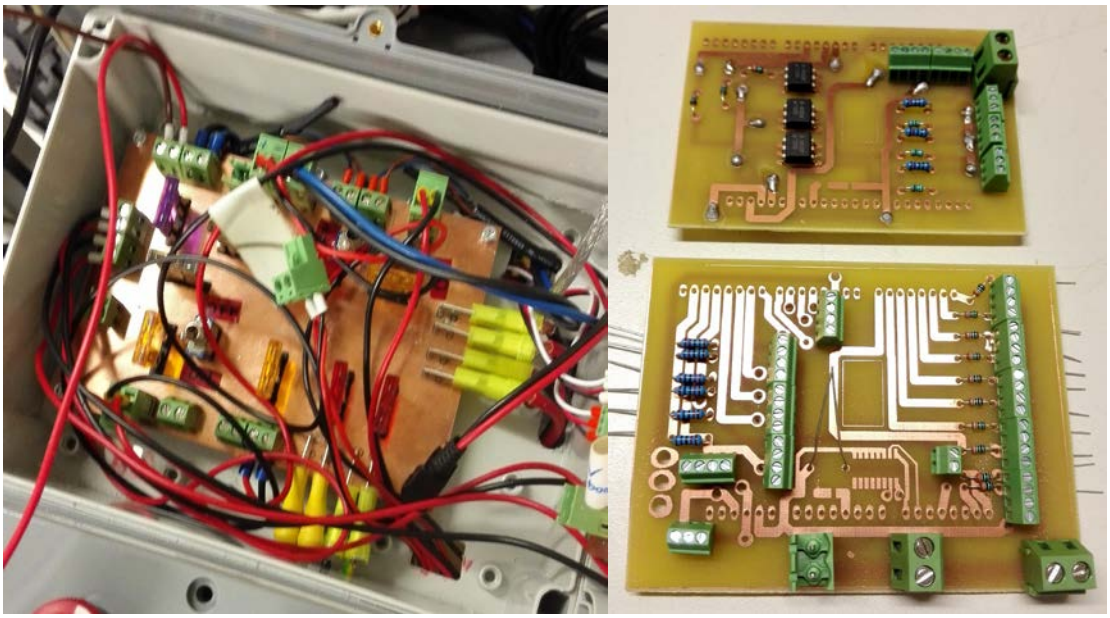


Figure 9: Electronics diagram

3.1 Printed Circuit Boards

For the power supply and control signals to be transmitted correctly to the different actuators we had to manufacture three printed circuit boards (PCBs) - one main power distribution board and two Arduino shields, one for each Arduino we used. The PDB's purpose was to divide the power from the batteries to all the components in the robot and take care of the conversion to lower voltages when needed. In short, we used 12V input voltage in two main circuits, a computer circuit for the computer and Arduinos and a motor circuit for all the other components. The idea of having two separate circuits was to avoid any problems with the NUC rebooting during voltage drops, i.e. when the motors are stopped. On the PDB we used fuses of different sizes to protect the wires and components from excessive currents in case of short circuits or other, to the components, fatal misbehaviour in the electronics. Figures 10 - 14 show the schematics for the PCBs used and the manufactured PCBs with all the components attached.



Figures 10 & 11: Power distribution board and Arduino shields

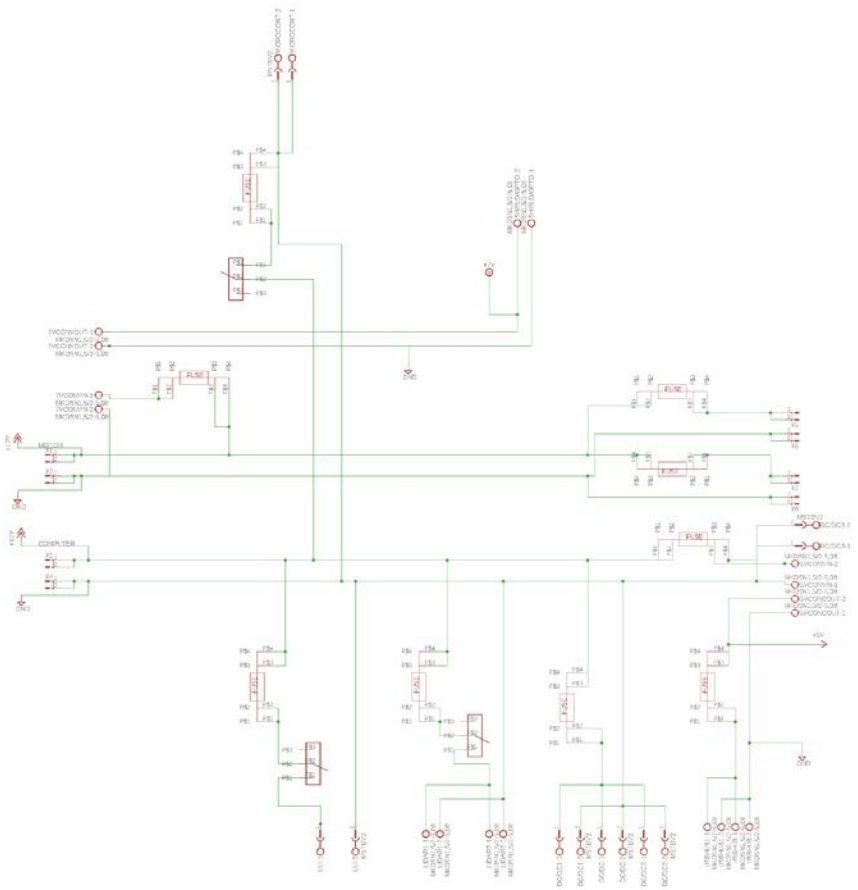


Figure 12: Schematic over the PDB

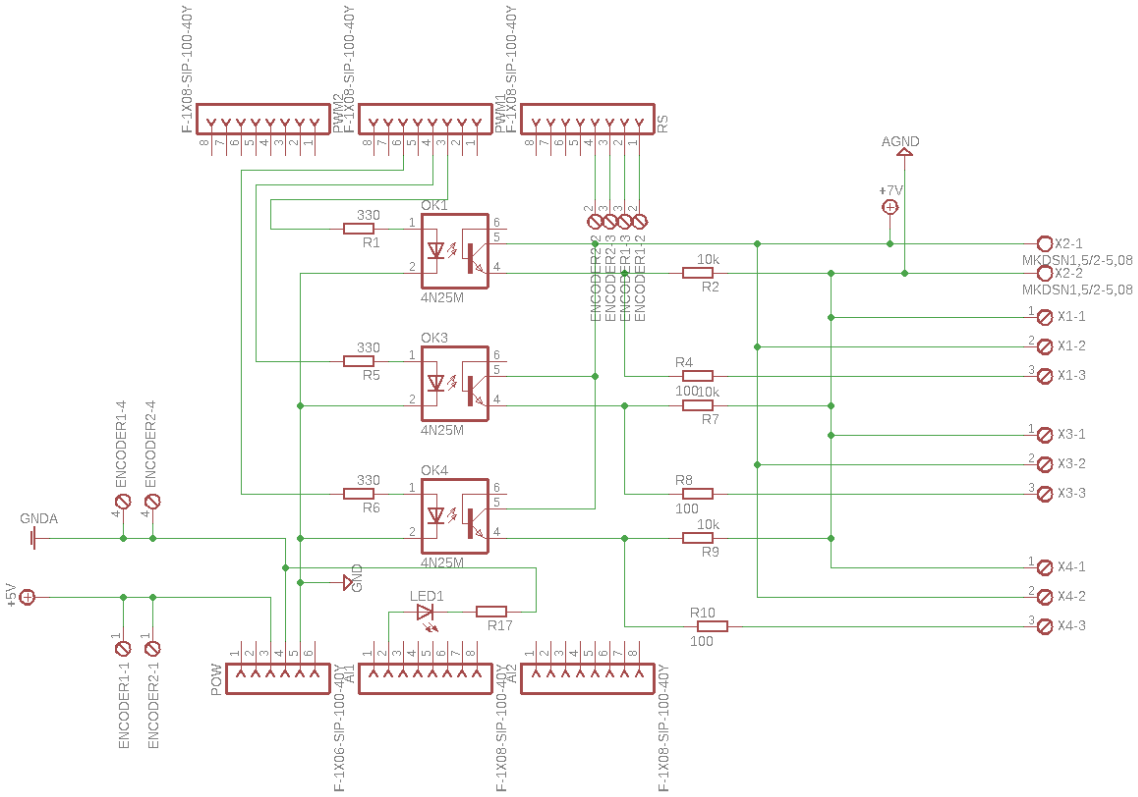


Figure 13: Schematic over the servo Arduino shield

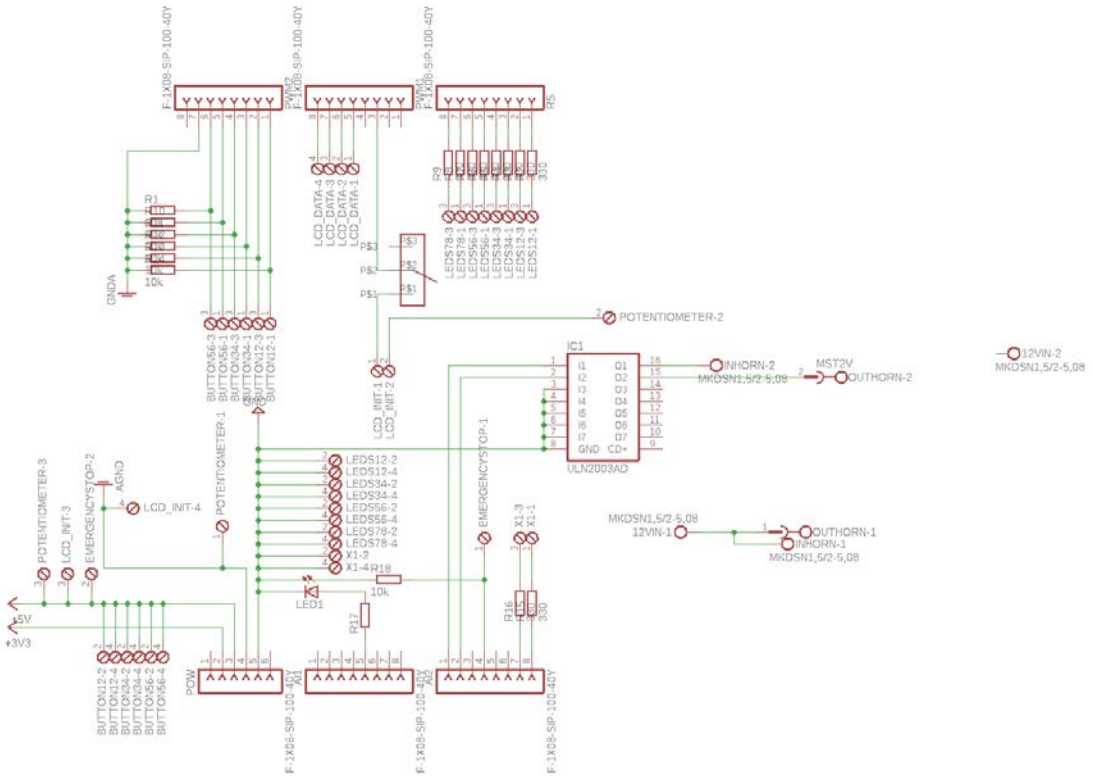


Figure 14: Schematic over the LUI Arduino shield

3.2 NUC Mini PC

The robot had only one computer on board. It's Intel's NUC with i7 processor. The previous robots had usually 2 computers but this one had enough processing power to do all the processing and run our main program without problems. The NUC ran Windows CE OS so we could run all our windows applications straight on it not needing to convert the files in any kind other than build our code and add the new version of the executable to the NUC.

3.3 VESCs

The VESC (Vedder's Electronic Speed Controller) is an open source project so all the software and hardware documentation is available to everyone [3]. ESC is an electronic circuit that controls and regulates the speed of an electric motor. It was originally built for things like electric skateboards but it suited our project well. The two VESCs controlled the motors by RPM. 1000 Was the minimum RPM that worked for us. We used higher RPM for tasks that didn't require camera vision and smaller RPM for the other tasks. The RPM was also higher in in-row sequence than in headland sequence.

The VESCs were powered by our batteries through the power distribution board. It was also connected to our PC through the USB hub and of course to the Brushless DC electric (BLDC) motor itself. By using the BLDC tool, we set up some maximum and minimum limits for the voltages, currents, RPMs and temperatures of batteries, motors and metal-oxide-semiconductor field-effect transistors (MOSFETs). We still had some problems

with the VESCs. Mainly the computer not being able to find the VESCs at the start of the run and sometimes the VESCs turned off during the tests and the robot stopped moving. After cleaning up the code we got mostly rid of those problems.

3.4 Sensors

Lazer Maizer had four different types of sensors: two laser scanners, a spatial board, two cut-offs and two smart cameras. These sensors were used for autonomous navigation, weed detection and monitoring the robot's batteries.

3.4.1 Laser Scanners

Lazer Maizer had two SICK TiM561 laser scanners, which can be seen on figure 15 below [4]. One of the laser scanners was mounted on the front and the other on the rear of the robot. The distance between the laser scanners was approximately 0.665 metres and with 270 degree this allowed us to scan almost everything around the robot. The laser scanners were placed approximately 20 centimeters from the ground. The use of two laser scanners and the symmetric design of the robot could've also allowed us to reverse the robot's directions on software level instead of turning 180 degrees, but unfortunately we never had sufficient time to make use of this design feature.

Figure 15: Sick TiM561 laser scanner [4].

3.4.2 Gyroscope

Lazer Maizer's had a PhidgetSpatial 3/3/3 Basic spatial board, which can be seen on figure 16 below [5]. This device was connected to the NUC Mini PC via USB and featured 3-axis accelerometer, gyroscope and compass, although in practise only its gyroscope's yaw axis was ever used in the navigation algorithm. The gyroscope's angular rate reading of yaw axis had quite a significant non-constant error that would quickly build up into an error of few degrees in the calculated heading. This error was reduced by subtracting a constant multiplied by passed time from the calculated heading. This solution wasn't perfect, but it was good enough.

Figure 16: PhidgetSpatial 3/3/3 Basic spatial board [5].

3.4.3 Cut-offs

The cut-offs were used for measuring the voltage and current of the batteries. They cut off the power if the voltage of the batteries drops too low. This protects the batteries from damage which is caused by being exposed to low voltages. The green, orange and red LEDs indicated the state of the batteries which is read from the cut-offs using USB RS-485 connections and serial communication. The cut-offs we used were old ones (designed and manufactured in 2015). They were also equipped with screw terminals designated for power buttons (ON and OFF separately) so that we could control the power supply to the robot and save the batteries. The figures below demonstrate the design (top and bottom layer) with the components and connectors included.

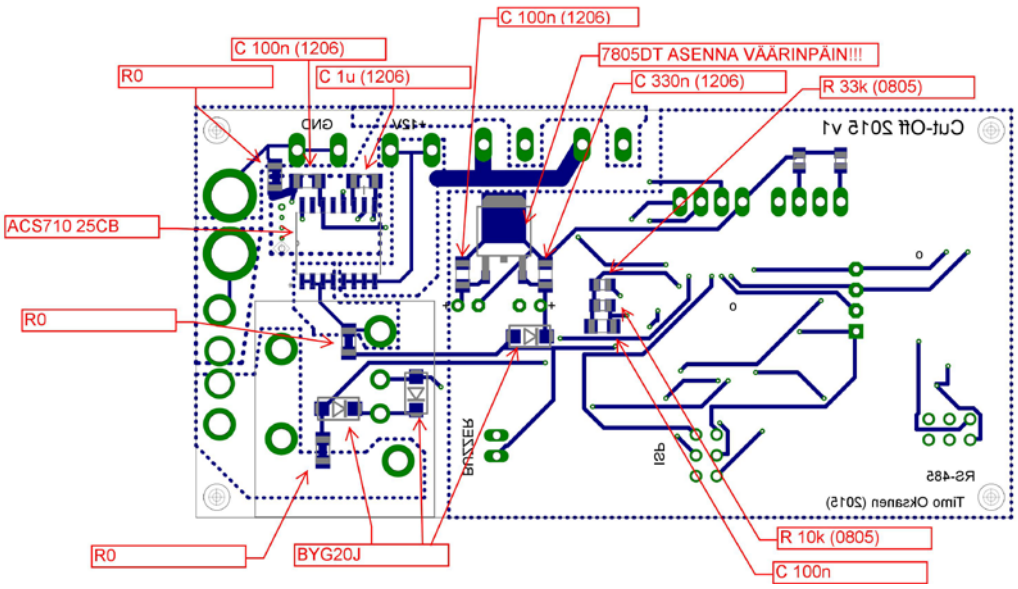


Figure 17: First part of the Cut-off 2015 board design

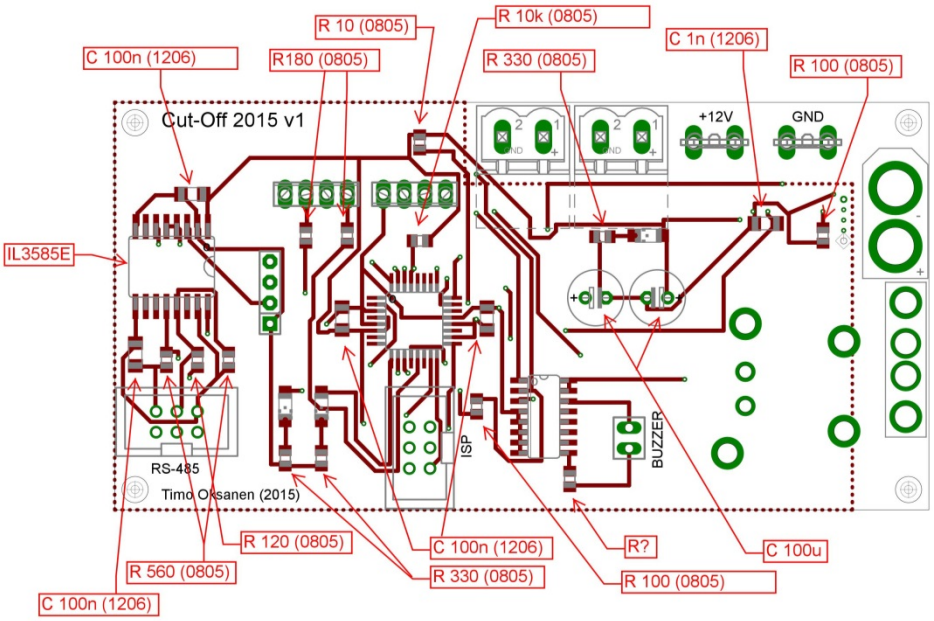


Figure 18: Second part Cut-off 2015 board design

3.4.4 Cameras

We used 2 JeVois Smart Machine Vision Cameras [6]. JeVois is an open-source machine vision ready camera as seen in figure 19. JeVois made it easy to add machine vision to our project by embedding image capture, vision processing, and delivery of results into the tiny camera itself. Because JeVois contains many different machine vision modules, we just chose the correct module and changed the parameters a bit. We used ObjectTracker module which tracks blue objects and counts the number of objects found by default. The parameters that were changed were the range of hue, saturation and value for the HSV window so we could detect blue objects with one camera and red objects with another camera. We could have also changed color balance, gain and exposure to get more reliable colors. Camera vision algorithms will be discussed later on in 4.7 Selective Sensing Algorithm.

Figure 19: JeVois Smart Machine Vision Camera [6].

4 Software

4.1 Toolchain

The complexity of the software solution and the fact that we had to be able to control the robot remotely required a comprehensive toolchain at the base of the software development. The main program, which was constructed in VisualStudio was built and executed on the NUC PC. It contained calls to all the other segments of the code, the logic and algorithms, which were executed as Simulink and MatLab models, the Arduino code, which was running on the Arduinos on board the robot and the Remote User Interface (RUI), which was running on a remote desktop. We did not use any ROS to implement this toolchain, all the function calls were executed in the main program.

From the user point of view, the toolchain ran from the RUI to the code segments running on the robot. The RUI sent commands to the main program on the NUC, from where the correct scripts were executed. This was still in the VisualStudio environment. The main program called the required Simulink models, which were built and executed as C++ code in VisualStudio in parallel to the main program. The RUI had supervisory access to the execution of the code at all times, meaning it could launch interruptions to pause or stop the main program. Data was also sent back to the RUI from the main program at consistent intervals to allow monitoring of the status of the different components and the state of the robot inside the control loop. Depending on at which state the control loop was, the main program sent and read messages to and from the Arduinos running their own code on board the robot. This could be seen as the final extension of the toolchain in a sense that the main program was the brains calling the different functions and the end effectors the last components furthest away from the user in the toolchain.

4.2 Communication

Unlike during the previous years, our team used mostly USB communication. There was no CAN hub. Basically all the devices except the laser scanners were connected to the NUC PC through the USB hubs. We also communicated using RS-485 serial communication. UDP and TCP/IP communication was also used. Figure 20 below shows which technologies were used with each device.

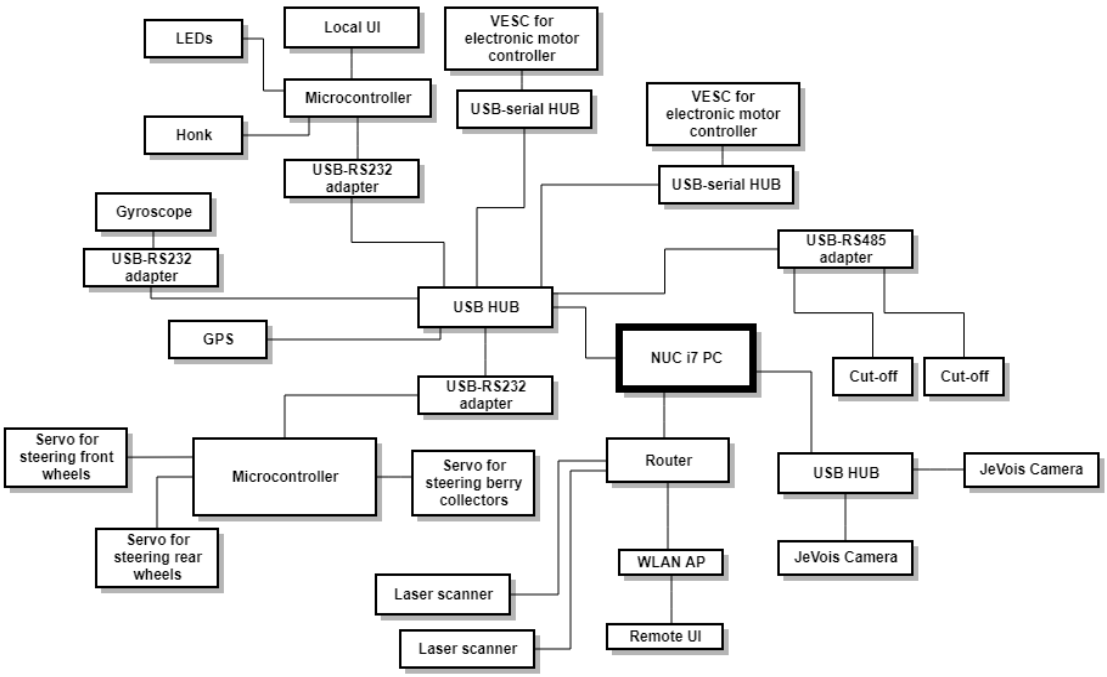


Figure 20: The communication diagram

4.3 Simulink Architecture

The Simulink model consists of three subsystem blocks: Pre-calibration, AI and Post-calibration. Pre-calibration subsystem converts inputs from sensors to desired formats. For example, degrees are converted into radians, polar coordinates are converted into cartesian coordinates and so on. The AI subsystem is the brain of Lazer Maizer. It contains the Stateflow charts and Matlab function blocks for navigation, weed patch detection and weeding trailer control. The AI subsystem also processes inputs from both local and remote user interfaces. Post-calibration subsystem converts the outputs of the AI subsystem into desired formats such as RPM for VESCs and radians to degrees for steering servos. The three subsystem blocks were implemented as a Simulink library. This allowed the same AI subsystem to be used in both the Simulink model used for code generation and the simulator.

The stateflow chart for navigation was implemented such that it could be used for all four tasks without modifications between tasks. The chart consists of initialization state, stop state and four navigation states with their substates. The initialization state sets the initial values for local parameters. In case the robot had to start a task away from the maize rows, the initialization state would drive the robot forward for a set duration. It is also possible to configure to the initialization state to skip some of the navigation states to allow testing of a specific state.

The four navigation states are inrow navigation, exit row, headland navigation and exit headland. The exit states consist of multiple substates such driving forward, turning and reversing. Headland navigation and exit headland states have alternative versions where

the robot uses crab walking to move on the headland and reverses its front and rear ends when exiting headland. These states will be described in more detail in Navigation section.

The stop state is entered based on inputs from LUI and RUI. During the stop state, the current navigation state of the robot can be changed from LUI. If the robot's current navigation state was changed before exiting the stop state, the new navigation state will be started from the beginning, otherwise the robot will resume from the point it was stopped. For example, if the robot is in middle of making a 90 degree turn when it is stopped, it will remember how much of that turn was completed and turn only that much after resuming. However, due to the use of substates, the robot would briefly enter each substates and execute their entry action once before moving on to the next substate. Effectively this means that if the robot is stopped while turning, it will briefly reverse and drive forward before finishing the turn. This behaviour could have been corrected by giving each substate a unique identifier to make it possible to return to the correct substate after stopping, but implementing this was not considered a high-priority as the robot rarely required stopping during exit row or exit headland navigation states.

4.4 Kinematics Modeling

The kinematics of a vehicle with two front wheels and two rear wheels can be approximated by considering the kinematics of a vehicle with one front wheel and one rear wheel, in other words a bicycle. Since Lazer Maizer is capable of four-wheel steering, the angles of both sets of wheels need to be considered separately instead of setting one of the angles constant. The maximum steering angles for both front and rear wheels were measured to be approximately 20 degrees left and right. The current steering angles couldn't be measured directly, but they could be approximated based on commands given to steering servos. However, the steering servos were unable to turn the wheels into precisely correct angles when attempting to drive straight. This was a major problem during indoor testing, but less notable in outdoor tests and the competition.

The kinematic equations were derived by first considering intuitive cases such as both sets of wheels having the same or the opposite angles and then generalizing the findings to all combinations of different angles. The parameters of the kinematic equations are the heading of the robot θ, the difference of angle of the front wheels from the heading of the robot α_f, the difference of angle of the rear wheels from the heading of the robot α_r, the velocity of the front wheels v_f, the velocity of the rear wheels v_f and the distance between the axes l. The heading of the robot is considered to be zero when the robot is parallel to positive y-axis. Turning right is considered positive and turning left is considered negative. With these parameters, it is possible to calculate the changes in the heading and the location of the robot with the following equations:

$$\dot{x} = \sin\theta \frac{v_f \cos\alpha_f + v_r \cos\alpha_r}{2} + \cos\theta \frac{v_f \sin\alpha_f + v_r \sin\alpha_r}{2}$$

$$\dot{y} = \cos\theta \frac{v_f \cos\alpha_f + v_r \cos\alpha_r}{2} - \sin\theta \frac{v_f \sin\alpha_f + v_r \sin\alpha_r}{2}$$

$$\dot{\theta} = \frac{\left(\tan\alpha_f - \tan\alpha_r\right) \frac{v_f \cos\alpha_f + v_r \cos\alpha_r}{2}}{l}$$

These equations form the forward kinematics of the robot. They define how wheel velocities and steering angles affect the heading and the location of the robot. To control the robot, it is necessary to know which velocities and steering angles provide the desired change in the heading and the location of the robot. These equations would be the inverse kinematics of the robot. While it would be possible to solve the inverse kinematics equations analytically, a numerical solution was deemed sufficient.

The first step in defining the inverse kinematics equations was to generate a large set of data using the forward kinematics equations with constant velocities and random steering angles. The second step was to assume that the inverse kinematics would be polynomial and the variables of the polynomial would be change in heading $\Delta\theta$ and change in lateral direction Δx and their multiplications. The third and the final step was to apply least squares estimation to solve coefficients of the polynomial. This yielded the following polynomials:

$$a_f = \Delta x + 0.176(\Delta x)^3 + 0.336\Delta\theta - 0.246\,(\Delta x)^2\Delta\theta + 0.057\,\Delta x(\Delta\theta)^2$$

$$a_r = \Delta x + 0.176(\Delta x)^3 - 0.336\Delta\theta + 0.246\,(\Delta x)^2\Delta\theta + 0.057\,\Delta x(\Delta\theta)^2$$

Terms with sufficiently small coefficients were omitted from these equations. It can be seen from these equations, that lateral movement is generated by turning the front and rear wheels in the same direction and turning is generated by turning the wheels in opposite directions. These inverse kinematics equations were used in the navigation algorithm to correct lateral and angular errors.

4.5 Positioning Algorithm

The purpose of the positioning algorithm is to calculate the robot's lateral distance from an imaginary centerline between two maize rows and the robot's heading with relation to the same centerline. These values are used in navigation algorithms and are referred to as lateral error and angular error from now on. The positioning algorithm uses data from laser scanners as its input.

The positioning algorithm is based on the idea that when the robot's angular error is close to zero, the data points from the laser scanners should form vertical groups of points and the lateral distance between these groups should be approximately 0.75 m. When a histogram is formed of the x-coordinates of the data points, the variance of the

histogram will be high as some bins have significantly more cases than other bins have. Therefore, a histogram with high variance implies low angular error. Due to the repetitive pattern of the maize rows, it is possible to apply modulo operation on the x-coordinates of the data points to merge the vertical groups.

To find the robot's current angular error the data points from laser scanners are rotated multiple times to find the angle with the histogram with the highest variance. This angle will be considered the angular error. The positioning algorithm tries angles from -45 degrees to 45 degrees with 1.25 degree resolution. The lateral error is chosen by calculating the center of mass of the histogram with the highest variance. The histograms have 90 bins, which means that the lateral error is calculated with approximately 1 cm resolution.

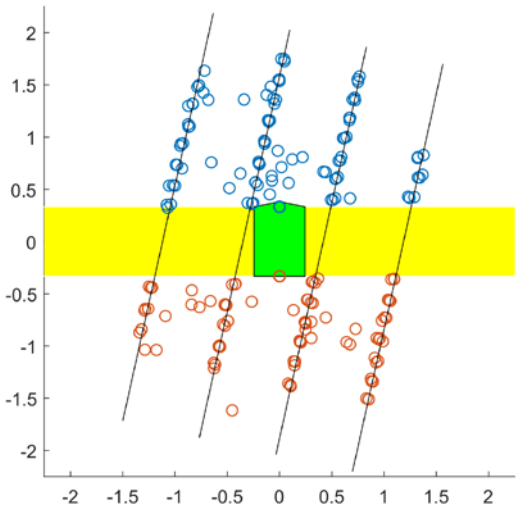


Figure 21: Output of positioning algorithm on simulated data.

In figure 21 above, the positioning algorithm has been tested with simulated laser scanner data. The robot is represented by the green rectangle and data from front and rear laser scanners is represented with blue and red circles respectively. The "blind zone" of the simulated laser scanners has been marked with yellow. This area is significantly larger than the blind zone of real laser scanners with 270-degree scanning area. The black lines have been drawn based on the angular error and lateral error provided by the positioning algorithm. The correct angular and lateral errors in this test were 12.6 degrees and -2.5 cm and the positioning algorithms outputs were 12.5 degrees and -2.5 cm.

The main version of the positioning algorithm uses data from both laser scanners, but there are variations of the algorithm that only use data from front or rear laser scanner. In headland navigation, the positioning algorithm is modified to filter away the data points on the opposite side of the maize rows and try angles from 0 degrees to 180 degrees with 2.5 degree resolution. The headland version of the positioning algorithm

was also extended to estimate the lateral distance from the ends of maize rows. The idea was to make another histogram, but this time with y-coordinates of the data points with x-coordinates close to the previously calculated lateral error. The lateral distance from the ends of maize rows was chosen by finding the first histogram bin where the number of cases was twice or more the mean of the histogram.

Unfortunately, the extended algorithm was very inconsistent. In figure 22 below, the extended algorithm is used on two different sets of simulated laser scanner data with the same heading and location of the robot. The lateral distance from the ends of maize rows is represented by a green line. The extended algorithm is successful on the first data set but it fails on the second data set.

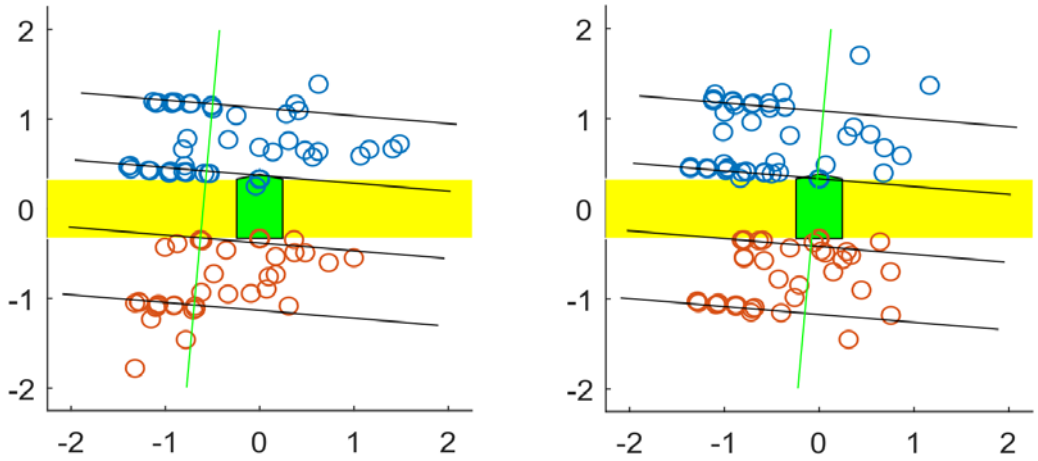


Figure 22: Output of the positioning algorithm on simulated data in headland.

Various parameters of the extended algorithm, such as the number of bins, the maximum distance from calculated lateral error and the threshold for accepting a bin as the output, were tuned to improve the algorithm, but it remained unreliable. Ultimately, the calculated lateral distance from the ends of maize rows was not used in navigation algorithms.

4.6 Navigation Algorithm

Lazer Maizer's navigation algorithm consists of four states executed in a loop. In addition to those states there are states for initialization and stopping. There was also an extra state for driving straight forward before entering the main loop. This state would have been used if the robot had to start a task with maize rows outside of its front laser scanner's range.

The inputs of the navigation algorithm are data from laser scanners in cartesian coordinates, yaw rotation in radians from gyroscope and commands related to stopping, resuming and manually changing the current state from RUI and LUI. The robot did not use odometry. Instead, tunable timers were used when the robot had to something for a predetermined distance, such as reversing a little after turning 90 degrees.

The main outputs of the navigation algorithm were angles for steering servos and setpoints for velocities, although there were always same for both front and rear wheels and could have been merged into one output. Additional outputs of the navigation algorithm include the current state of the algorithm, stop indicator, turning direction, current index in the given path and various debug outputs.

4.6.1 Inrow Navigation

During inrow navigation the robot is driving forward at constant velocity while correcting angular and lateral errors by adjusting the steering servos. These errors are calculated by using the positioning algorithm and used as inputs for a PID-controller. The outputs of the PID-controller are used as the variables of the inverse kinematics equations and their results are used as the setpoints for steering servos. Additionally there was a feed forward gain parameter for allowing the angular error to directly affect the setpoint for steering servos, but this was not used in the competition.

Figure 23 below shows the angular and lateral errors calculated by the positioning algorithm during the first row in task 2 in competition. The angular error is kept less 15 degrees while the lateral error is less than 10 centimetres at all times during the first row. Both of the errors have some fluctuation, which suggests that the parameters of the PID-controller were not optimal.

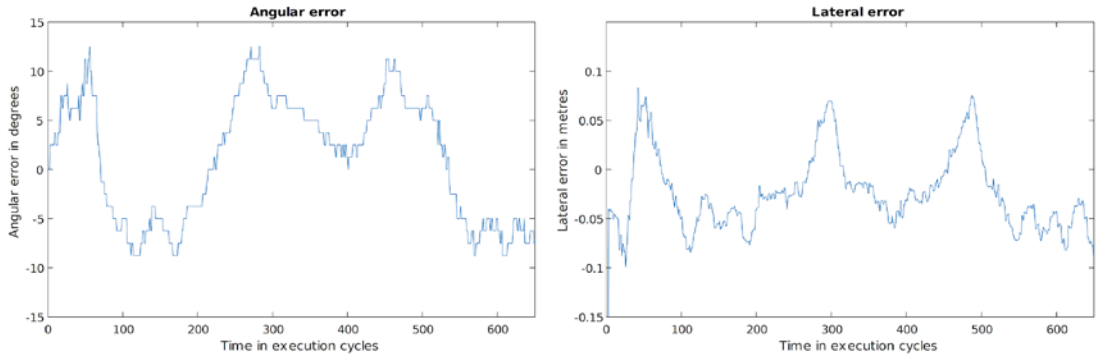


Figure 23: The outputs of the positioning algorithm during the first row in task 2.

In pre-calibration, scans that exceeded the chosen maximum range for laser scanners were set to “zero”. These zeros had their x-coordinate set to zero while the y-coordinate was set to plus half of the robot’s length for front laser scanner and minus half of the robot’s length for rear laser scanners. This allowed the number of zeros for each of the laser scanners be counted separately.

The number of zeros in front laser scanner was used for detecting ends of rows. The number of scans from each of the laser scanners was 811 and once there were more than 775 zeros, the navigation algorithm navigation transitioned from inrow navigation to exit row turning sequence. Figure 24 below shows the number of zeros in front laser scanner during the first row of task 1 in the competition. The row end is detected at 926

and it can be clearly seen that the number of zeros is steadily increasing prior to that moment.

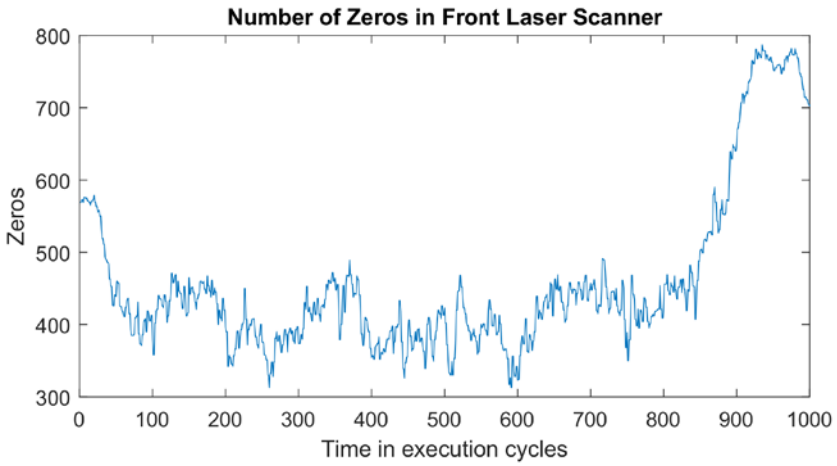


Figure 24: Zero counting during the first row in task 1.

4.6.2 Turning Sequences

Lazer Maizer had different turning sequences for exiting rows and exiting headland. These sequences consisted of multiple substates executed sequentially. This caused undesired behaviour if the robot was stopped during this sequence. In retrospective, this could have been fixed by implementing the turning sequences within a single state and using a local variable to remember the current substate.

The turning sequence for exiting a row and entering headland is described in the following list:

1. Reverse a little while adjusting the heading similarly to inrow navigation.
2. Drive forward to compensate for the distance reversed.
3. Turn left or right until the yaw rotation of the gyroscope has changed by 90 degrees.
4. Reverse a little to compensate for the turning radius.

This turning sequence was very accurate and positioned the robot's center point approximately on the centerline of the exited row with the robot heading in perpendicular direction from the centerline, but a lot of time was wasted reversing instead of moving forward.

The first step of the sequence was added to make sure that the robot's heading would be correct before turning, but the logged data from the competition shows that the robot's angular error was only few degrees when the end of the row was detected. Figure 25 below shows the current angular error calculated by the positioning algorithm prior to the first turn in task 1 of the competition. The end of the row was detected at 926 and the figure suggests that reversing the robot to adjust the heading wouldn't have

been necessary. However, as we had limited time to test our algorithms on the practise field, we decided to keep the first step of the turning sequence.

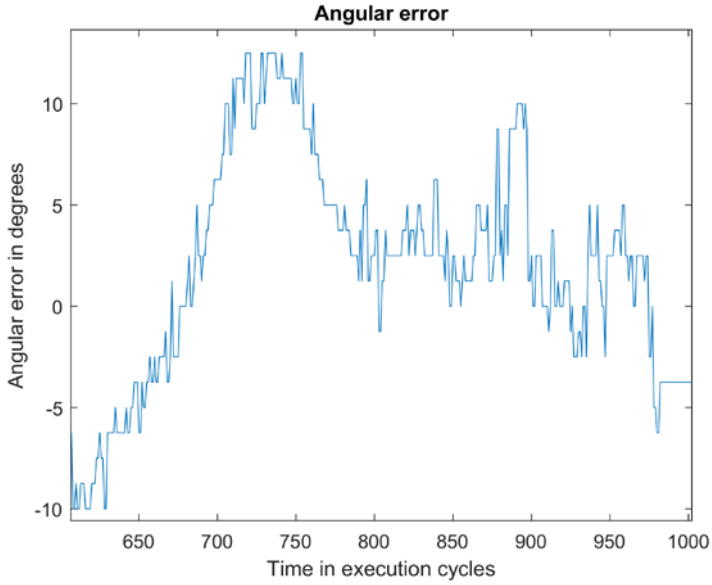


Figure 25: Angular error prior to first turn in task 1.

The turning sequence for exiting headland and entering a row is described in the following list:

1. Reverse to make room for the turn
2. Turn left or right until the yaw rotation of the gyroscope has changed by 90 degrees.
3. Reverse a little to give the robot some space for adjusting angular and lateral errors.

Due to the robot's large minimum turning radius, it was necessary to reverse to make room for the turn. However, it is debatable whether or not the third step was really necessary. While the step made it easier for the robot to enter a new row without damaging any crops it also wasted time by reversing instead of driving forward. With more time to do tests we could have either adjusted the duration of reversing or removed it completely.

4.6.3 Headland Navigation and Row Counting

Similarly, to inrow navigation, the algorithm for headland navigation would adjust the robot's heading using a PID-controller. Adjusting the robot's lateral distance from the maize rows with another PID-controller was also implemented but not used due to the inconsistencies of the extended positioning algorithm.

Figure 26 below show the current angular error calculated by the positioning algorithm during headland navigation in task 2 of the competition. Here the robot has turned left and is skipping two rows to enter the third row. The PID-controller is able to keep the angular error below 20 degrees.

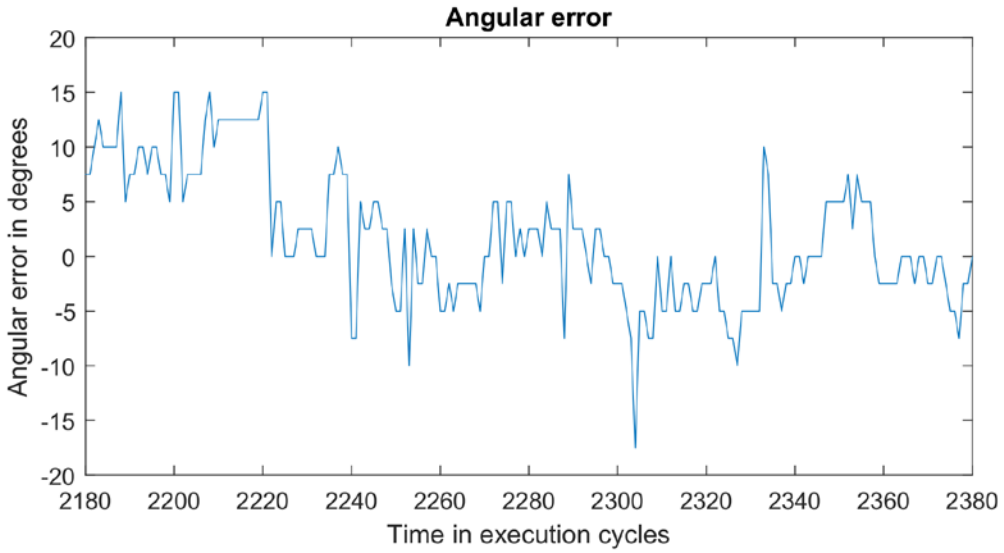


Figure 26: Angular error during headland navigation in task 2.

Since the positioning algorithm used during headland navigation was mostly same as the one used during inrow navigation, it was possible to use the lateral error from the centerline to implement a row counting algorithm. This value acted similarly to a sawtooth wave, it would rise when approaching a maize row and fall sharply after the row had been passed. However, the sawtooth wave was very noisy and had to be filtered to be reliable. A simple filter that outputted a new mean every five steps and the previous mean otherwise was deemed sufficient.

The row counting algorithm would now increment its row counter whenever a peak was detected. Four conditions were chosen to detect the peaks of a sawtooth wave:

1. The previous mean should be sufficiently high
2. The current mean should be sufficiently low
3. The difference between these means should be sufficiently high
4. A fixed amount of time has to have passed since a peak was last detected

In our initials tests, only the first two conditions were used count rows. They were soon found unreliable on their own as the robot could accidently count the same row twice if it corrected its heading immediately after incrementing the row counter. A forced delay after a peak was implemented to avoid counting the same row twice. Even after that, finding universal parameters for the first two conditions turned out to be difficult. The third condition was added to allow making the first two conditions less strict. With these four conditions in use the row counting algorithm worked reliably.

Figure 27 below shows the input and the output of the row counting algorithm during task 2 of the competition. This is the same situation as in figure 24. The row counting algorithm is able to use the lateral error to count passed rows. The minimum value for previous mean was 0.3, the maximum value for current mean was 0.5 and the the minimum difference between the means was 0.1.

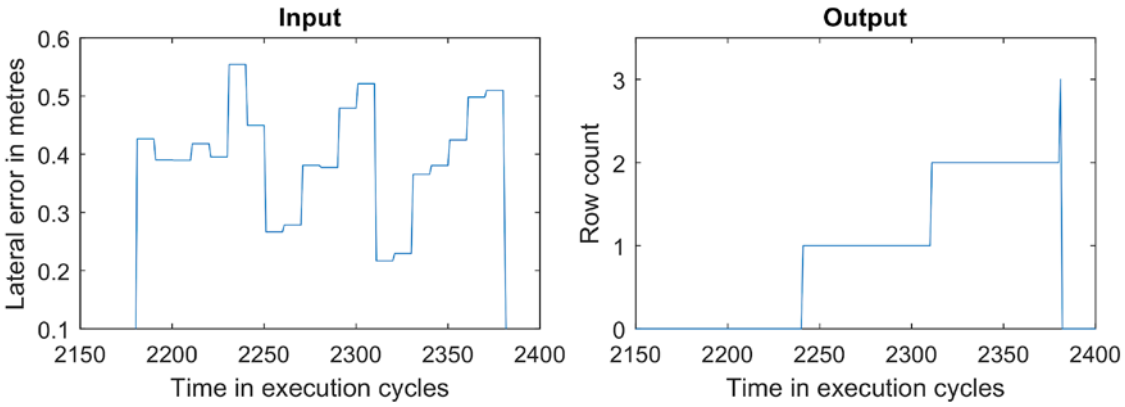


Figure 27: Row counting algorithm during headland navigation in task 2.

Once the correct number of rows had been passed, the robot would exit the headland and enter to correct row with the sequence described previously. This row counting algorithm was used in all tasks.

An alternative algorithm that merged both turning sequences and headland navigation into one state was also developed. The idea of the algorithm was to utilize Laizer Maizer's symmetric design and its ability to crab steer to move in a pattern similar to a triangle wave while counting passed rows. This alternative algorithm was therefore nicknamed "Crabwave". After the correct number of rows had been passed, the robot would virtually switch directions and drive back to the rows without physically turning.

The logic of Crabwave was tested successfully in simulations. Tests on the real robot however revealed Crabwave's weaknesses. First of all, the algorithm made no adjustments to steering angles which meant that if it began with angular error, this error would never be corrected. At worst this resulted in the robot slowly moving away from the rows and eventually losing them permanently. Perhaps this could have been fixed by allowing the robot to make minor adjustments to steering angles to correct its heading.

Secondly and more importantly, the positioning algorithm became unreliable as the robot's distance from the maize rows increased making it difficult to count the passed rows properly. The maximum steering angles of Laizer Maizer were approximately 20 degrees, which meant that the robot would have to move at least 1.15 metres forward to complete the transition from a row to the next row in one period of the triangle wave, which was our goal. This was deemed impractical and the frequency of the triangle wave was increased to keep the robot closer to the maize rows. This made Crabwave

significantly slower than our previous headland navigation algorithm and its further development and testing was halted.

4.6.4 Testing, Tuning and Performance

The navigation algorithm was tested with a simulator during the early phases of its development. Once the navigation algorithm was deemed to work correctly and safely it was tested in parts on the real robot. After all the individual parts had been tested, the entire navigation algorithm was uploaded onto the robot and we began tuning the parameters.

The navigation algorithm had over 40 tunable parameters, although some of them were never changed and some had become obsolete during the development of the algorithm. Regardless, this allowed us to use the same algorithm for each task and simply adjust it by loading suitable parameters from an XML-file. We had a different XML-file for each task, which allowed us to make the robot move slower in the curved rows of task 1 and faster in the straight rows of task 2.

Practically all of our tests before the event were done indoors with sticks on a wooden plank acting as maize rows. This meant that we had to drive slowly and stop the test whenever the robot was about to collide with the mockup maize rows. Nevertheless, we were able to tune the parameters to a point where the robot could drive autonomously for approximately five minutes without hitting the mockup rows.

We adjusted our parameters on the test fields the day before the competition and received 5th and 4th rank on tasks 1 and 2 respectively. In task 1, the robot drove for 46 metres and damaged only 2 plants. In task 2, the robot drove 56 metres but damaged 5 plants.

4.7 Selective Sensing Algorithm

It turned out that JeVois smart camera's object tracker module couldn't detect more than one object of the chosen color at once. Whenever an object of the chosen color was detected the smart camera would send a message containing the object's coordinates. Therefore, we decided to simply count the received messages and use their difference to deduce whether there were more blue or red tees. The messages for blue and red detections will be called blue and red hits from now on.

Lazer Maizer's color comparison algorithm was initialized by setting the numbers of previous hits and zero levels for both colors to zero and setting a reset counter to zero. The reset counter was used to detect when there hadn't been any hits recently. This allowed the robot to avoid detecting the same weed patch twice. The color comparison algorithm consisted of the following steps:

1. Compare the current numbers of hits to previous ones.
    a. If there is growth in either color, skip the 2nd step.

b. Else increment the reset counter.

2. Check if the reset counter has surpassed a threshold.
    a. If the threshold has been surpassed and there has been sufficient amount of hits compared to the zero levels, compare the numbers of hits and output 1 if there were more red hits and -1 if there were more blue hits. The output of the algorithm is otherwise zero. Reset the reset counter and set the zero levels for both colors to the current numbers of hits.
    b. If the threshold has been surpassed but there hasn't been sufficient amount of hits, only set the zero levels for both colors to the current numbers of hits.
    c. Else do nothing and go to the 3rd step.
3. Set the previous numbers of hits for both colors to current numbers of hits and repeat from beginning.

We didn't have much time for tuning the parameters of the color comparison algorithm or the object tracker module. Figure 28 below shows the inputs and the outputs of the color comparison algorithm during task 3 in the competition. The algorithm was able to detect six weed patches with four correct signals and two wrong signals. The minimum thresholds for weed patch detection and resetting were 7 hits and 25 operation cycles respectively.

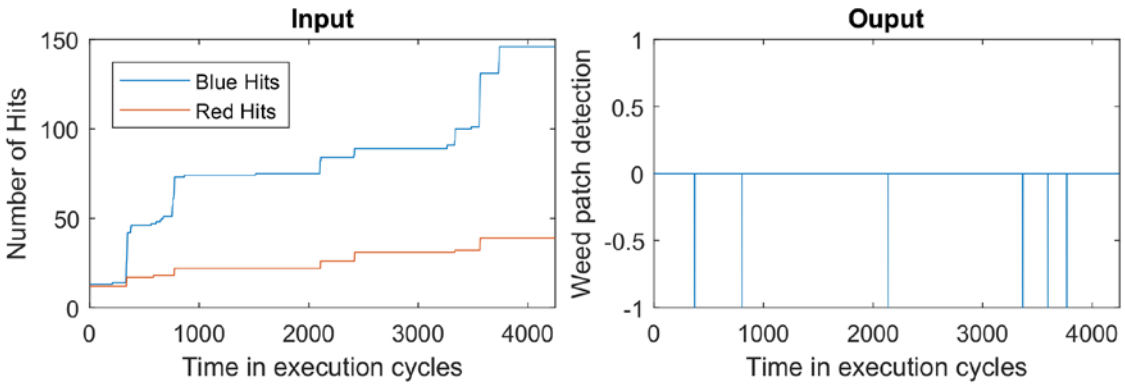


Figure 28: Colour comparison algorithm in task 3.

There are several explanations to why our algorithm didn't work as intended. The weather during the competition was very cloudy and dark and it is possible that the parameters used in competition were tuned during sunny weather and therefore didn't work properly in the competition. As seen below in figures 29 and 30, the cameras could detect blue and red objects when using the AMCap camera test program.

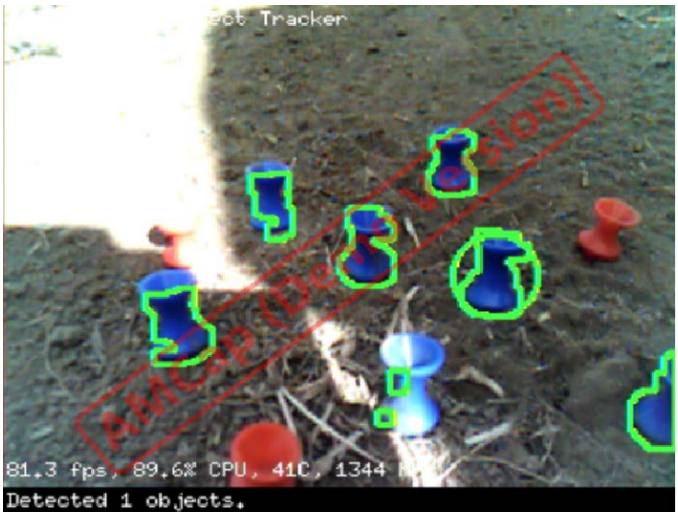


Figure 29: JeVois camera detecting blue objects visualized by AMCap program

Figure 30: JeVois camera detecting red objects visualized by AMCap program

The parameters of the color comparison algorithm could've been too strict. The data shown in figure 28 suggests that there should've been weed patch detections at 650 and 2450 approximately, but the number of hits was considered too low and disregarded as noise.

It is possible that the robot didn't drive straight over some of the weed patches and the cameras missed them. It is also worth noting that we used two smart cameras side-by-side with one camera for each color, so it is possible that sometimes the majority of red tees were on the side with the camera for detecting blue tees and vice versa.

Lastly, we had a fair share of technical issues with the JeVois smart cameras including having to ask for a second chance in the competition because the robot's computer wouldn't detect the smart cameras when we tried to begin task 3. Therefore, it is plausible that the cameras simply didn't function properly during the competition. Most

likely the real reason for our algorithm's subpar performance is a combination of these explanations. For task 4 we lowered the weed patch detection threshold to 3 hits, but that still wasn't low enough and the robot ignored the weed patch.

4.8 Simulator

Three different simulators were developed for Lazer Maizer. The simulators were used for developing and testing navigation algorithms, but they were not accurate or precise enough for tuning parameters of the algorithms.

The first simulator only simulated the robot's kinematics and was used for testing how the robot would react to different steering angles and velocities. The inputs were either constants defined at the beginning of the simulation or variables read from a gamepad. The maize rows and the robot were visualized with an S-function.

The Simulink model of the first simulator can be seen on figure 31 below. Starting from the left, the simulator generates steering angles for front and rear servos and velocities for front and rear wheels. These values are used to calculate average values for velocities in X- and Y-directions. Next these averages, steering angles and current heading, which is called theta in this figure, are used to update the location and the heading of the simulated robot. Finally, the robot is visualized with an S-function inside a block called Animate.

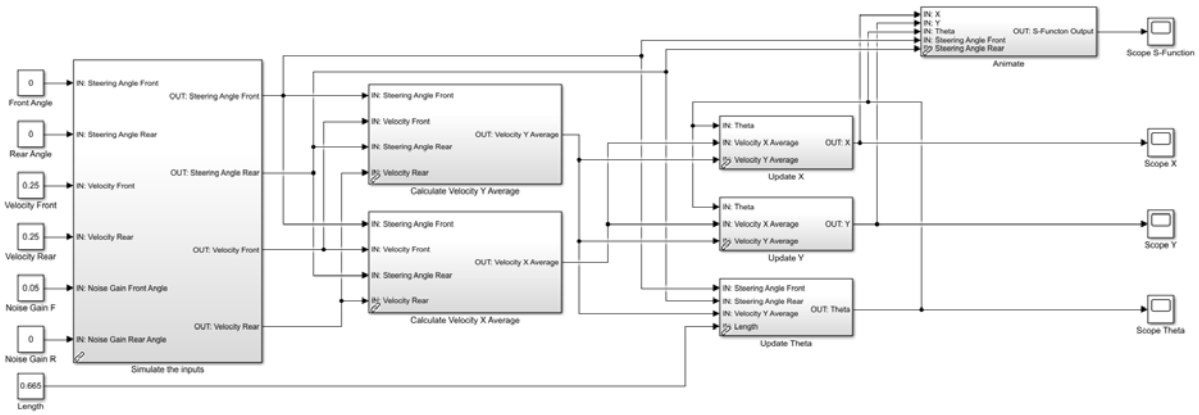


Figure 31: The Simulink model of the first simulator.

Unlike the other two simulators, the second simulator was a Matlab script rather than a Simulink model. It first generated simulated maize rows and then simulated laser scans of those rows based on the position of the simulated robot. These simulated laser scans were used as inputs for various prototype versions of navigation algorithms that updated the simulated robot's steering angles. The second simulator visualized the maize rows and the robot as well. Most of the navigation algorithms used in the final version of Laizer Maizer's software were developed using this simulator. A simulation of an early version of the navigation algorithm can be seen on figure 32 below. The simulated robot started from the sixth row, turned left skipping two rows, turned left

again skipping five rows, turned right skipping no rows, turned left skipping three rows and the finished after turning left once more.

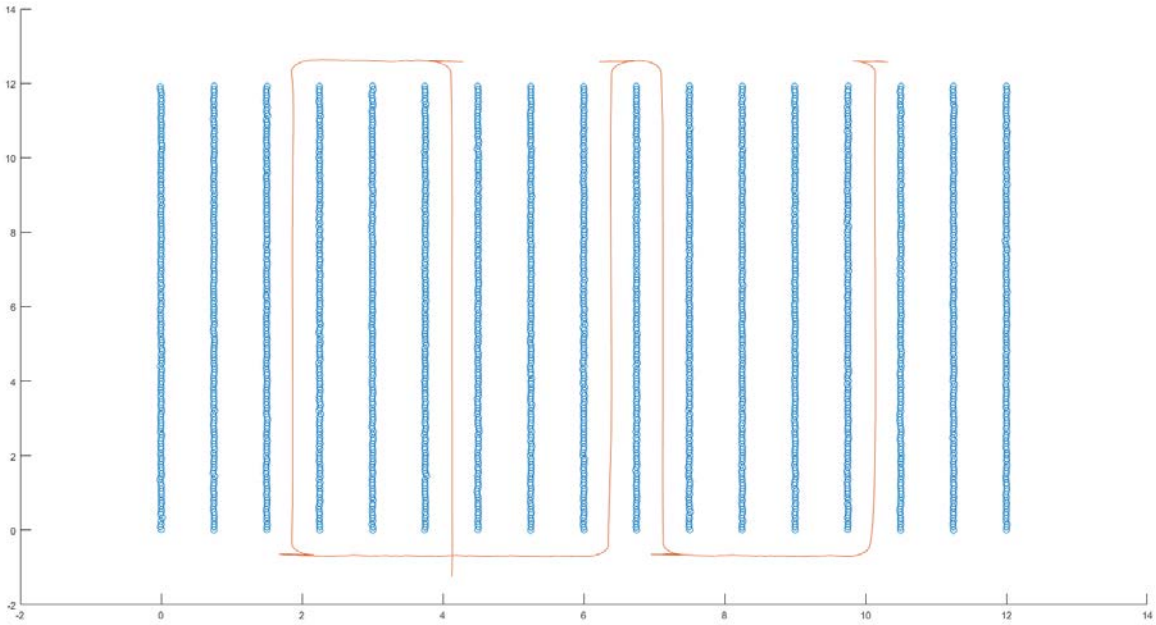


Figure 32: Simulation of the navigation algorithm using the second simulator.

The third and final simulator was essentially a Simulink version of the second simulator. However, it used the same AI subsystem block that was used for generating the code, which meant that any updates to the AI subsystem block would also affect the simulator. The AI block was connected to a "Virtual World" Matlab function that would generate simulated laser scans and update the location and the heading of the robot based on the commands received from the AI block similarly how the Matlab script in the second simulation worked. The S-function used in the first simulator was updated and used in the third simulator for visualization.

The Simulink model of the third simulator can be seen on figure 33 below. Starting from the left, the model consists of "Virtual World" Matlab function that generates simulated laser scans, the AI subsystem block that calculates new steering angles and velocities based on the inputted laser scans among other things and the same Animate block used in the first simulator. The Virtual World function and the AI block are connected to each other in a loop, which is why delay blocks are required to prevent an algebraic loop from occurring. In addition, there are various scope blocks for debugging the AI block.

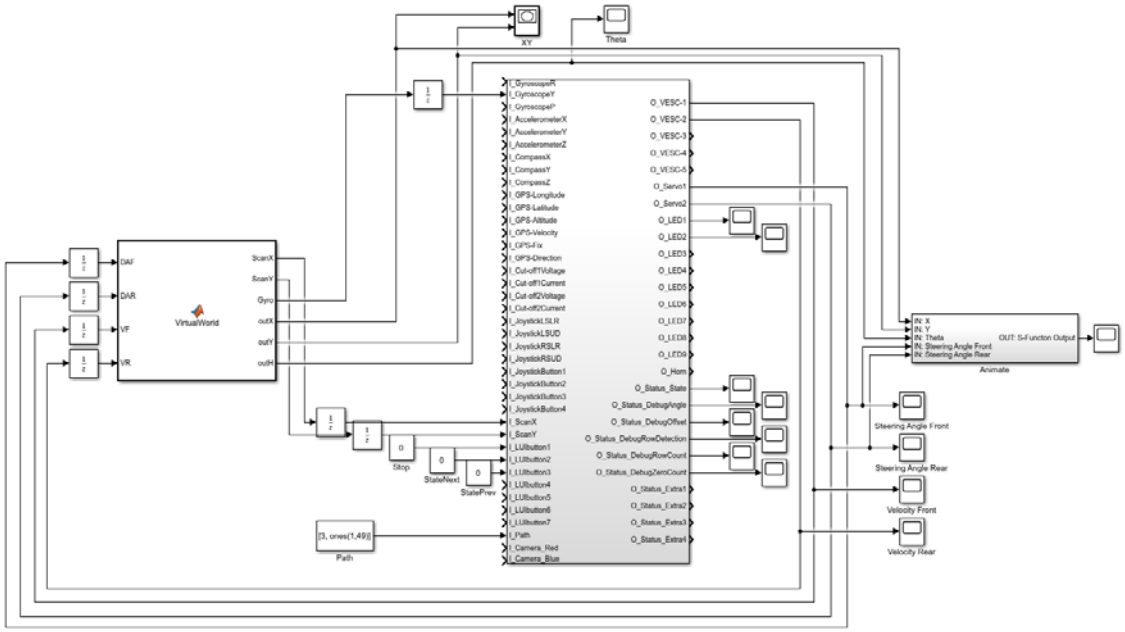


Figure 33: The Simulink model of the third simulator.

4.9 Remote User Interface

The remote user interface was constructed as a separate program running on a remote laptop. The communication to the NUC was performed over wifi using udp protocol.

The main purpose of the RUI was to control the robot and monitor the different parts during the run. Essentially, we used the RUI to initialize the state and task of the robot and start the run, meaning we could choose if we wanted to run manual or autonomous mode and which task to run. This command was sent as a parameter of a preassigned udp message to the NUC where the algorithms were executed. We also had full control of the robot during the runs, i.e. we could stop or pause the executing script at any time. Due to the specifications of the contest, we also added the functionality of being able to pause, unpause, start and stop the execution with a joystick connected to the laptop. All the commands passed from the RUI to the NUC were defined as separate udp messages and had their own preassigned ports to which the NUC was constantly listening (tables 2-3). The data in the messages had their own structs in the main program to which it was added once the message was received including the interval at which the main program checked to see if any message was sent to the port in question.

Table 2: UDP messages to NUC

Port	Description	Struct name
7001	Activation command and IP of RUI	IPstruct
7002	Commands from RUI	RUIcommands
7003	“Alive” signal and RUI status	RUIstatus
7004	Path for task 2	SequenceStruct

Table 3: UDP messages from NUC to RUI

Port	Description	Struct	Interval
7111	Measurements from VESC 1 (front)	bldcMeasure	20ms
7112	Measurements from VESC 2	bldcMeasure	20ms
7113	Inputs of Simulink model	ExtXTU_FRE_RT_T	20ms
7114	Outputs of Simulink model	ExtY_FRE_RT_T	20ms
7115	Measurements from gyroscope	GyroStruct	20ms
7116	Robot status	Status	Unused
7117	LED states + horn	LUILeds	20ms
7118	Message to servo Arduino	arduinoMessage	20ms
7119	Parameters of Simulink model	P_FRE_RT_T	20ms

The RUI was constructed in VisualStudio as a Windows forms application. All the messages were triggered by pressing generic buttons on the RUI which allowed for clear and intuitive usage of the RUI, even to relatively inexperienced users. The idea was that all of us should be able to operate the RUI in case of unexpected events. Figure 34 shows a screenshot of the RUI.

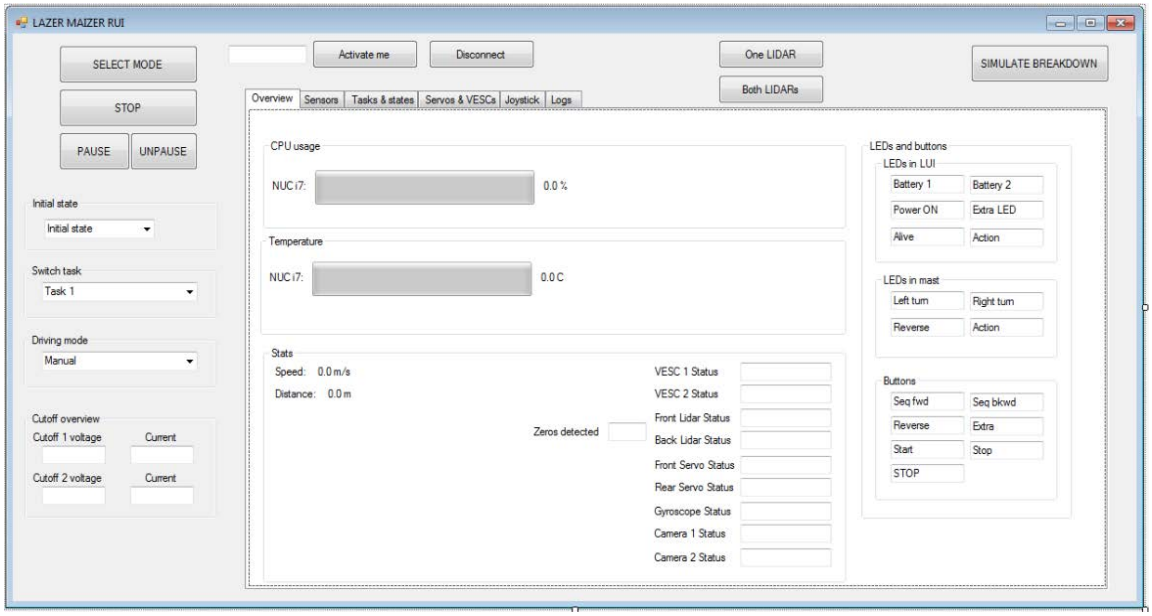


Figure 34: Screenshot of the Remote User Interface

4.10 Arduino Programming

The Arduinos were programmed using the Arduino IDE software developing tool provided by Arduino. The programming itself has a setup function that runs ones as the Arduino is initialized and a loop function which runs repeatedly at a certain interval. In essence, the IDE is a tool for calling C/C++ functions on the Arduino, most of the functionalities themselves are already built into libraries and functions which for the most part are open source. In our solution, this basically meant that we implemented serial communication through USB straight from the NUC to the Arduinos, through which we sent the values the Arduino then assigned to the pins accordingly. The first problem was that the serial port has a buffer at the beginning, which meant that our messages were pushed further than expected. We solved this by implementing a state machine approach which parses the message and only moves forward through the machine logic when the expected byte was read. These bytes were predetermined using a protocol of our own making. In short, the Arduino read one byte at a time until the ending bytes were read. Only then did it write the read values to the pins. This way, we managed to avoid false values being sent to the components due to message corruption.

For the servos, we used fast PWM (pulse width modulation) mode to send PWM signals to the servos. The signals were passed through optocouplers that generated the needed signals for the servos. The Arduino can be configured to operate fast PWM on certain pins, although the syntax for obtaining this is quite cryptical although well described in the Arduino documentation. For controlling the LEDs, the state of the pins simply had to be set to HIGH (5V output) when turned on and LOW (0V) when turned off. The pins connected to the LUI buttons obeyed the same logic, when the button was pressed, the output signal was HIGH, vice versa LOW when the button was untouched. For the LCD, we used Arduino's LiquidCrystal library. The horn was operated using a ULN2003

Darlington transistor array which works so that if the input of the transistor is set to HIGH, the output is set to ground. This way, by constantly providing the signalling horns with 12V input voltage, we could turn them on by feeding a 5V signal to the transistor array and hence feeding ground to the horns. This was needed because the Arduino Mega 2560 which we used doesn't have a 12V input voltage available in the control logic. Below is a screenshot of a segment of the Arduino code controlling the LUI (Figure 35).

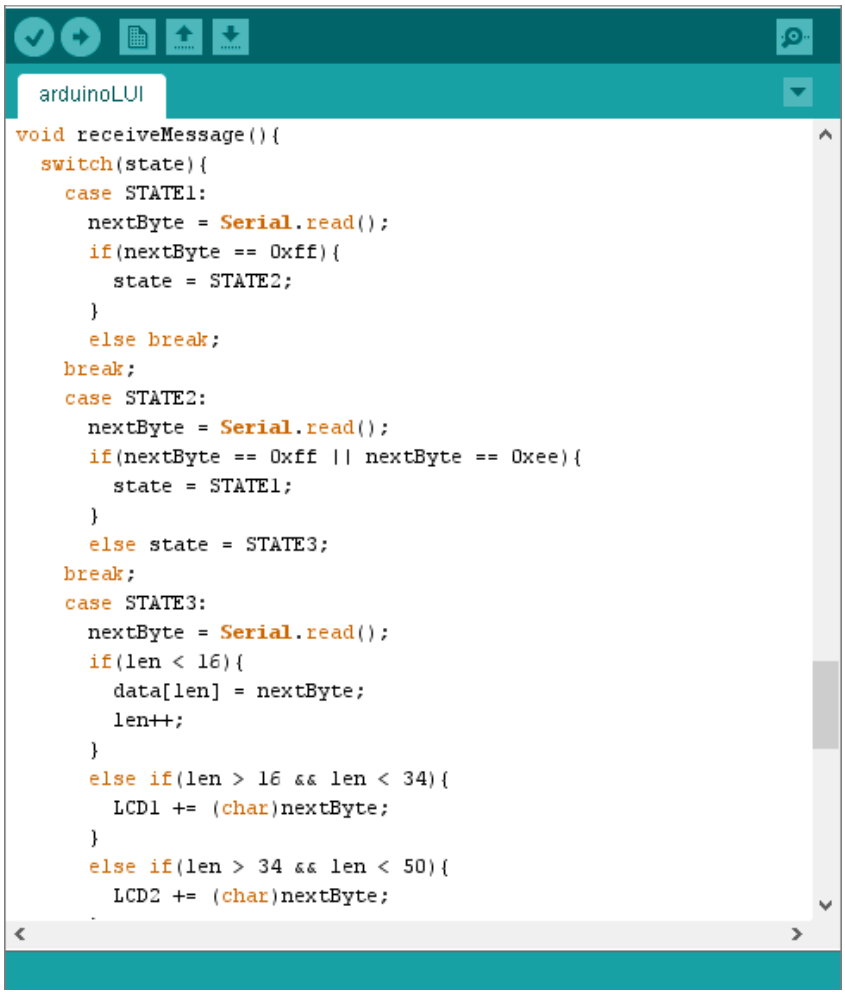


Figure 35: Segment of the code running on the Arduino controlling the LUI

5 Discussion and Conclusions

Overall, the project was successful. Lazer Maizer is able to drive autonomously in field conditions, observe its surroundings and adjust its course accordingly. The robot ranked 5th, 4th and 3rd in tasks 1, 2 and 3 respectively. The robot also ranked 5th in task 4, but this was actually the last rank shared with seven other robots. Lazer Maizer achieved the 4th overall rank shared with one other robot. The robot didn't participate into freestyle as we didn't have enough time to design or implement anything that we could've presented.

In terms of mechanics, there are some things that worked brilliantly and some that should be re-engineered if this project was continued. The equal force support system is one of the highlights of the robot, and also the construction of the axle module was well optimized for minimum complexity.

There are two things that would clearly need further work. the mounting of the belt drive pulleys with a tightening screw was unreliable causing the pulley to slip against the shaft. A solution would be to drill a hole through the shaft and mounting the pulley with a bolt and a locknut. Another issue with the robot is its weight. Because of the tight schedule and because easy manufacturability was emphasized in the chassis design, weight optimization had to be compromised. Electrical components are well placed inside the robot, but cable and wire management was not considered before assembling the robot.

The software used during the competition could be said to have worked as intended for most of the time. The logic of the navigation algorithm could've been optimized and polished for faster and more accurate movement if we had the time to do the necessary tests and make changes based on the results. All the device drivers worked very well during the competition with the exception of reading values from the cut-offs and messages from the JeVois smart cameras. The competition rules allowed us to use the RUI only for starting up the robot and telling it the direction of the first turn or the path in task 2. The RUI features many more features that were useful during various tests.

While the main program of Lazer Maizer worked well, it had become very long and complex by the time of the competition. This made it difficult to make any changes to it. The main program should have been cleaned up and split into multiple files to make it easier to edit certain parts of it separately but doing so would have taken time away for other things that needed to be done and splitting the main program hastily could have caused unforeseen problems.

6 Acknowledgements

Finally, we would like to thank our sponsors that made the project possible: AGCO, Koneviesti, Laserle, SICK and Suomen Kulttuurirahasto.

2 References

1. *Field Robot Event*, http://www.fieldrobot.com/event/
2. *FRE2018 Program Booklet*, http://www.fieldrobot.com/event/wp-content/uploads/2018/06/Booklet_2018_complete_v4-cover.pdf
3. *VESC - Open Source ESC*, http://vedder.se/2015/01/vesc-open-source-esc/
4. *TiM 561*, https://www.sick.com/gb/en/detection-and-ranging-solutions/2d-lidar-sensors/tim5xx/tim561-2050101/p/p369446
5. *PhidgetSpatial 3/3/3 Basic*, https://www.phidgets.com/?tier=3&catid=10&pcid=8&prodid=1025
6. *JeVois Smart Machine Vision Camera*, http://jevois.org/
7. *Top Agrar Online* https://www.topagrar.com/news/Technik-Techniknews-Robot-Field-Event-top-agrar-Design-Award-2018-geht-nach-Finnland-9258481.html
8. *UniCorn Fieldrobot 2017 - Final Report*