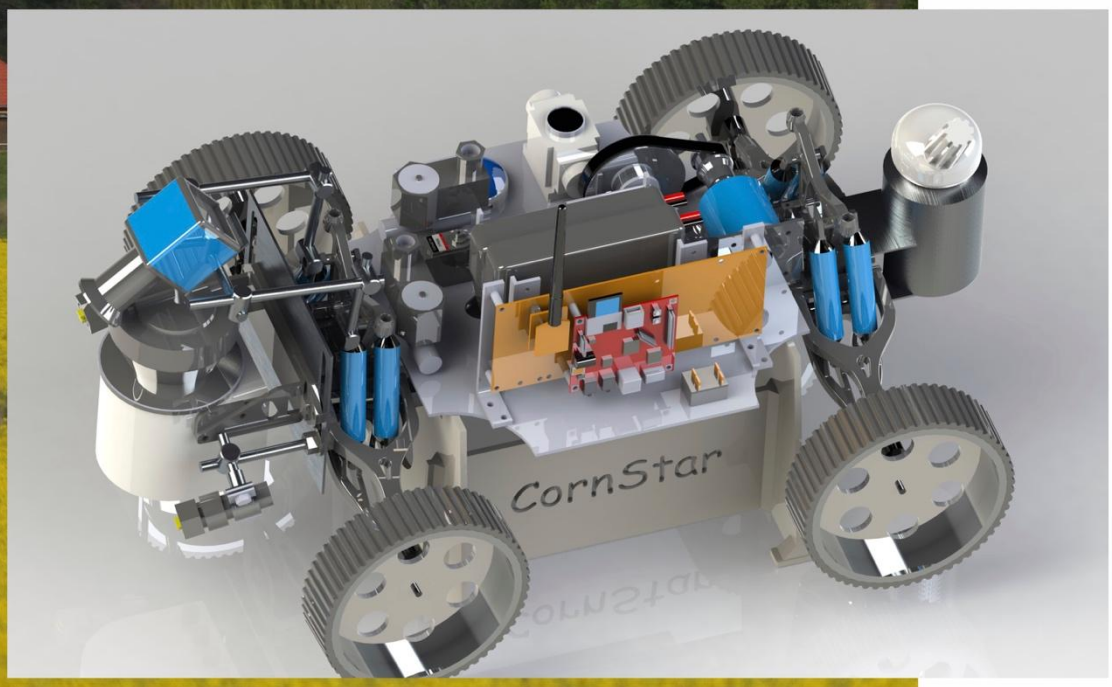
Proceedings of the 13th Field Robot Event 2015

University of Maribor

Faculty of Agriculture and Life Sciences



University of Maribor
Faculty of Agriculture and Life Sciences
Biosystems Engineering

FRE2015

Maribor - SLOVENIA, 16. - 18. June

FRE2015.UM.SI

EurAgEng

CLAAS Stiftung

Proceedings of the 13th Field Robot Event 2015

CIP - Kataložni zapis o publikaciji

Univerzitetna knjižnica Maribor

004.9:633(0.034.2)

FIELD Robot Event (13 ; 2015 ; Maribor)

Proceedings of the 13th Field Robot Event 2015, Maribor, Slovenia, 16.-18. June [Elektronski vir] / editor Jurij Rakun. - El. zbornik. - Maribor : Faculty of Agriculture and Life Sciences, Department of Biosystems Engineering, 2015

Dostopno tudi na: http://fre2015.um.si/images/Downloads/Proceedings-FRE2015.pdf

ISBN 978-961-6317-48-1 (pdf) 1. Rakun, Jurij

COBISS.SI-ID 85049601

Foreword

Welcome to the Field Robot Event 2015!

The 13th Field Robot Event (FRE) was organised by the Department of the Biosystems Engineering at the University of Maribor, Faculty of Agriculture and Life Sciences in Maribor, Slovenia from Monday the 15th of June to Thursday 18th of June 2015.

The FRE has been founded by the Wageningen University in 2003 in order to motivate students to develop autonomous field robots. In 2015 the event was organised for the first time in Slovenia and we have followed the tradition to challenge the robots in agricultural tasks, to promote meeting of international colleagues and simply to have fun during the contest!

18 Teams from 16 Universities from 10 Coutries have us presented some interesting and creative solutions in over all 4 tasks: basic navigation, advanced navigation, weed detection and freestyle.

A very special thank you goes to the sponsors Class Stiftung and EurAgEng that made possible for the event to come true.

On behalf of the organising team from Department of the Biosystems Engineering

Miran Lakota

More information is available on the FRE2015 website: fre2015.um.si

Editor's note

Field Robot Event is an annual competition that brings together multiple international teams. It helps to shape the engineers of tomorrow and presents their own view of what future machines could look like and what would they do. The members, students and mentors, form the teams to use their knowledge to solve specific agricultural problems; to automate the work, to complete it more quickly and to make it more precise. After endless hours of work, countless tries and painful steps each team constructs their own creation – a small robot with functionality that could be far beyond everyday agricultural equipment.

In June 2015 the 13th Field Robot Event was hosted by the department of Biosystems Engineering from the University of Maribor. It was our great honour to host 18 teams from different countries; these include Czech Republic, Denmark, Finland, Germany, Netherlands, Romania, Slovenia, Turkey and United Kingdom. The teams competed in four tasks: Basic navigation, Advanced navigation, Weed control and Freestyle. The winners were:

Place	Basic Navigation	Advance Navigation	Weed Control	Freestyle
1st	SIEGEN (Zephyr)	HARPER ADAMS (Florence)	TU DAENMARK (DTUni-corn)	AALTO UNIVERSITY & UNIVERSITY OF HELSINKI (GroundBreaker)
2nd	HARPER ADAMS (Florence)	CZECH UNI (Eduro)	SIEGEN (Zephyr)	UNIVERSITY OF APPLIED SCIENCE OSNABRÜCK (The Great Cornholio
3rd	TU DAENMARK (DTUni-corn	KAISERSLAUTERN (MEC)	CZECH UNI (Eduro)	WAGENINGEN UNIVERSITY (SmartTrike)

And the overall winners of the FRE2015 were:

Place	Institution	Robot	Country
1st	UNIVERSITÄT SIEGEN	ZEPHYR	GERMANY
2nd	CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE	EDURO	CZECH REPUBLIC
2nd	TECHICAL UNIVERSITY OF DAENMARK	DTUNI-CORN	DENMARK
3rd	HARPER ADAMS UNIVERSITY	FLORENCE	UNITED KINGDOM

Congratulations to all the teams!

Jurij Rakun

Table of contents

Beteigeuze - Karlsruhe Institute of Technologie, Germany

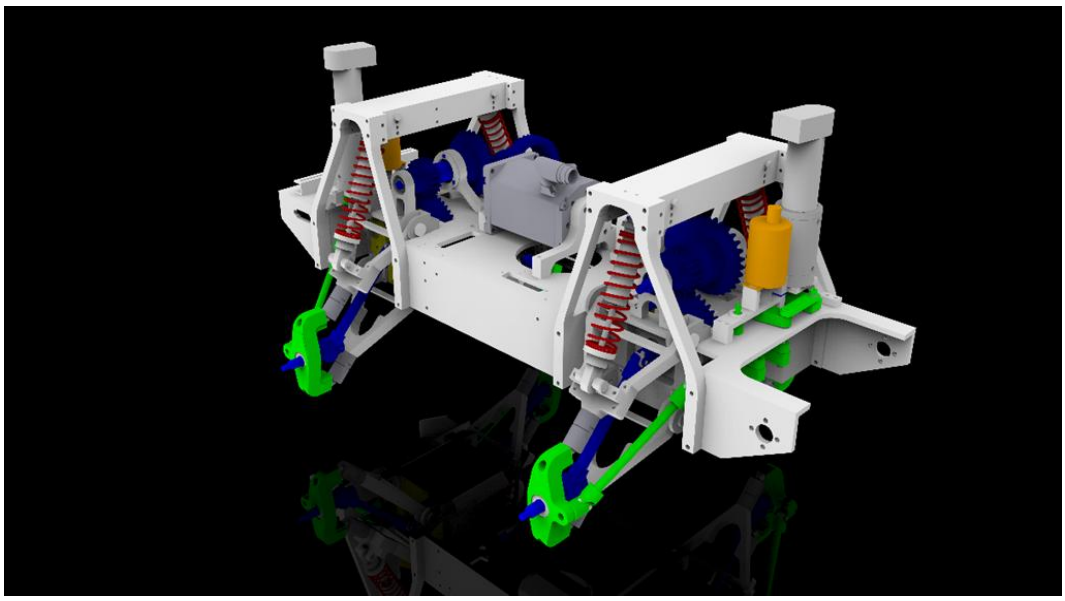
Lennart Nachtigall

Karlsruhe Institute of Technologie
Rintheimer Queralle 2, 76131 Karlsruhe, Germany
kamaro.engineering@googlemail.com

1 Introduction

This work describes the mechanics, hardware and software structure of the robot Beteigeuze from the group Kamaro Engineering e.V. It deals futhermore with the challenges faced at the Field Robot Event 2015 and how the robot dealt with them.

2 Mechanics



In order to fullfill the requirements of a robot driving in a field the drive chain was designed as a 4-Wheeldrive 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. As seen in [Image 1] each axle mounting has its own suspension. Even at the maximum speed of 11 km/h we can

ensure a smooth ride. The steering is limited to +/- 37° on each side. Front and back axis can be steered separately therefore also diagonal movements are possible.

3 Hardware

The central unit is a desktop computer mainboard in the mini-itx format with a Intel i5 quadcore processor and 8 Gb of ram.

Its located in the inner of the central base plate. Even though it is empty on [Image 2] the computer cage is recognisable.

The motor control is done via Can-Bus on a ARM Cortex M4 Board. This board handles most of the access to the hardware and abstracts the communication with the computer. Due to problems with the stability of the wiring all cables where encapsulated in modular boxes which can be changed separately. The emergency stop, the on/off switch and a 20 characters / 4 lines displays with 8 control buttons is mounted on the top board formed by the cable casing as shown in [Image 3].

3.1 Sensors

For the environment cognition Beteigeuze has several sensors on board: Some of them can be seen on [Image 3]. The main navigation is done by two laser scanners, one in the front (Sick LMS100) and one in the back (Sick Tim551). In addition to the laser scanners the robot has rotary encoders for the front and the back steering and a 9Axis orientation sensor, which both can be used for calculating the driving path and the current position relative to start position.

For detection tasks we use three webcams mounted on the pole in the front as shown in [Image 3]

4 Software

The completely self-developed software is organized in several modules even written in different languages. [Image 4] provides a short summary of the macrostructure.

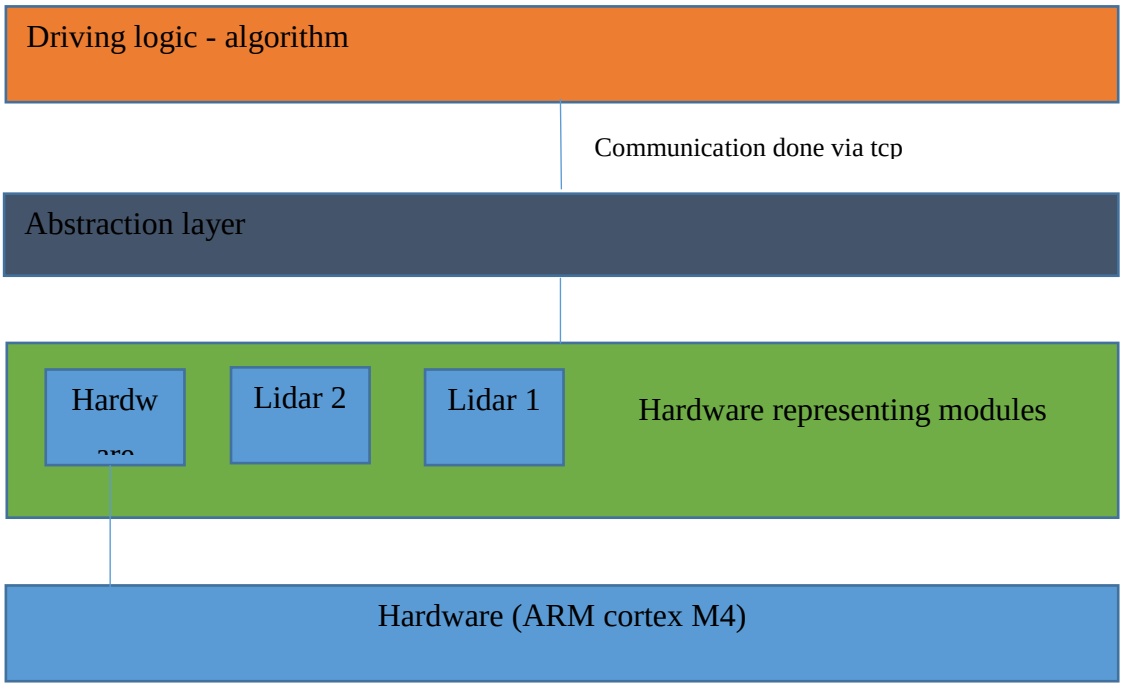


Image 4: Software macrostructure

Most parts of the software run on the computer on a Xubuntu 14.04 Linux. The modular structure was chosen after several attempts of a monolithic software design. It provides easy adding of new software functionality, improved stability and the possibility of trying new algorithms without restarting all necessary software components. The abstraction layer provides an interface to all abstracted hardware components via a tcp connection. Furthermore it controls and restricts access to some commands. E.g. all driving commands may only be used by a single specified tcp connection.

4.1 Overview on algorithms used at the FRE

For driving through the rows we searched for a free cone of 50cm length in the lidar raw data, beginning from the center. This gave us a line in the direction of the row. We

shifted this line to the left/right by an amount calculated from the distance of the robot to the plants to the left/right. We then used a PID controller to steer onto the point on this line that is 1m in front of the robot. The main problem with this algorithm was the high latency in our system which led to strong oscillations at higher speeds.

Turning was done in three steps. First we turned out of the row by driving a hard coded distance. We then followed the lines by least-squares fitting a line through the points seen by the lidar to the side of the robot and controlling the robot to drive on a line parallel to this line. We counted the lines passing by by counting the points to the side of the robot and putting this value into a Schmitt trigger like function. After passing the correct amount of lines we turned into the line using a hard coded distance again.

For the creation of the map we used our wheel odometry to offset the lidar scans to each other and brute force using random shifts and rotations to further align them onto each other. This created pretty detailed maps, but had problems with the rotation especially while changing the row.

5 Conclusion

The participation in the Fre 2015 showed us some advantages and disadvantages of the robot “Beteigeuze”. Due the huge amounts of rain on the first day most parts of the testing field were flooded. In this rough environment our four-wheel drive concept showed it superiority. Even though some other teams especially with smaller robots had problems with a too small amount of friction and traction we had no problems due to the muddy ground. Also the big weight of the robot (40 kg) contributed to a great friction. On the contrary the large size of the robot, the robot barely fitted through the rows, proved as a big difficulty in a stable control process. Therefore we had problems increasing the maximum speed of “Beteigeuze” without damaging any plants. From our point of view a robot may not damage any plants even if it is doing its work slower.

Cornstar – University of Maribor, Slovenia

Miran Lakota[1], Peter Berk[1], Jurij Rakun[1], Jože Kraner[2]

[1]Faculty of Agriculture and Life Sciences, University of Maribor, Pivola 10, 2311 Hoce, Slovenia
fre2015@um.si

[2]Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia

1 Introduction

Agriculture is one of the disciplines that is heavily influence by the breakthroughs made in computer science, electronics and mechanics. This is most evident for big food producers rely on the use of heavy machinery, but not the case for mid- and small-sized farms, which can pose a potential food safety problem in terms of transition media of diseases if handled manually. In addition, some work even for big producers still demands manual labor that is time consuming, exhausting and expensive. The thought of introducing a small army of intelligent robots to do the job quicker and more accurate seems appealing, but challenging. For one, natural uncontrolled environment poses problems with its changing conditions. An overview on the subject showed that there are some potentially good solutions where authors rely on specific conditions like night time or they simply move the work to controlled environments – in green houses. Some are simply too big or too heavy to be useful at this stage.

In order to achieve our goal, we decided to put our efforts to build a small autonomous self oriented robot, that could for instance serve as a potential tool for selective pesticide spraying, fertilizer applicator or even as a device that could estimate the yield at the end of the harvest by simply taking digitalized snapshots of the tree canopies. In the following section we start by describing our solution in detail.

2 Mechanics

The robot consists of four major components: mechanics, actuators, embedded electronics and sensors. The mechanical part includes a four-wheel drive, two axles that can be turned individual by servomotors, automatic gearbox with forward and backward two-speed transmission and a three-phase motor. The actuator part includes a pressured reservoir, two electro-magnetic valves and nozzles that enable the robot to spray. The onboard digital camera and a laser range sensor make up the sensoric part of the robot. The last but not least is the electronic part of the robot, which includes an embedded computer, peripheral circuit and a power supply. The render image of the robot is depicted on Fig. 1, showing all crucial components of the robot, while Fig. 2 depicts the robot in action while spraying roses.

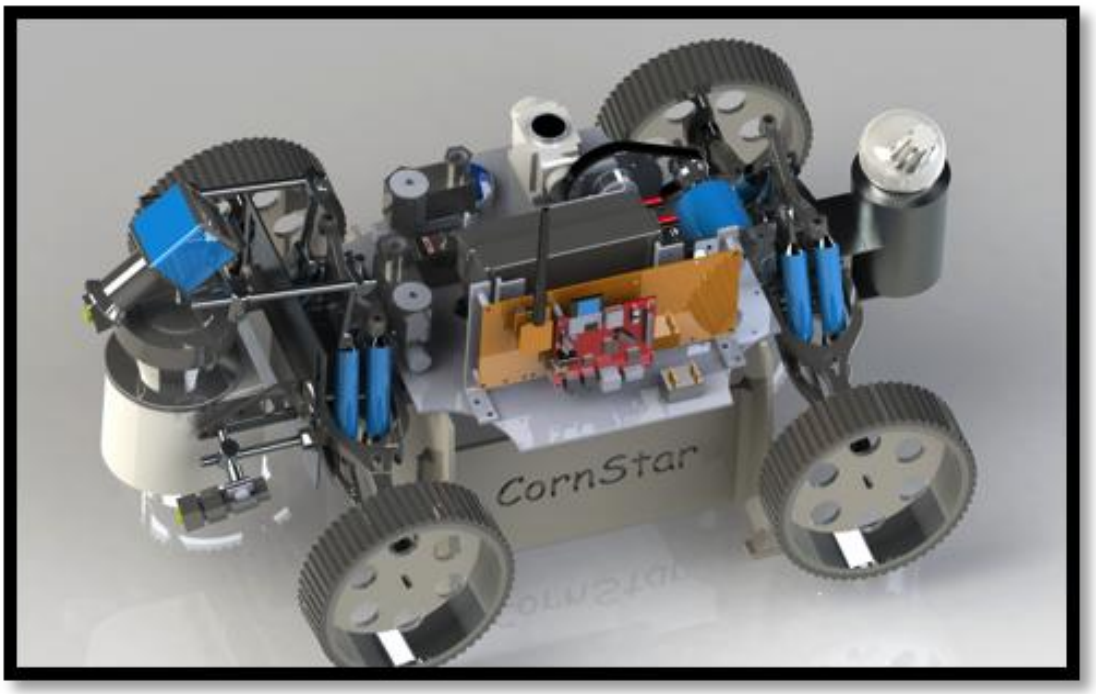


Figure 1 Render of a Constar robot – depicting all its crucial components.

Figure 2 Mobile robot while spraying roses.

3 Sensors

The sensors on the robot include an industrial camera (The imaging source's DBK31BU03) and a laser range scanner (SICK TIM 310). The camera captures Bayer encoded images with resolution of 1024×768 pixels at 30 fps. The camera is connected to the embedded computer by using an USB connection. The laser range sensor captures distances between a sensor and an obstacle in a 270º radius from 0.05 up to 4 m distance. The sampling frequency of the laser scanner is 15 Hz with 1º resolution. It is connected to the embedded computer via USB connection

3.1 Hardware

The electronic part of the robot is made up of an embedded computer (Nitrogen6x) and a peripheral expansion board build around a AVR ATmega 128 microcontroller. The computer is based on AVR Cortex A9 quad core processor running at 1 GHz, with 1 GB of memory and can reach a performance of 2000 BogoMIPS per core. It offers a vast

number of ports, where USB was used for camera and laser range sensors, while the UART port is used to communicate with the expansion board. An Ubuntu Linux (Linaro) distribution was selected for the operating system that was uploaded to a SD card, where a custom version of the kernel had to be compiled to support the USB based camera.

The camera, the laser range sensor and the embedded circuit are all powered by two 3 cell LiPo batteries connected in parallel. They can provide a constant 175 A or 350 A peak current (each), 11.1V voltage and 3800 mAh capacity (each). In order to power the circuit the voltage is lowered to TTL voltage levels using a switching regulator.

4 Software and strategy

The embedded computer runs on a Linaro Linux operating system where a custom build version of the kernel was compiled in order to support the camera. In addition, in order to control low-level hardware the mobile robot uses a Robotic Operating System (ROS), which is a meta operating system. ROS provides a number of libraries and tools for different robotic applications; it includes hardware drivers, device drivers, visualizers, messages passing and package management, all this as open source software. ROS supports code reuse in robotics research and development, it is a distributed framework of processes where processes can be grouped in packages. For the mobile robot we used device drivers for Sick laser scanner and other hardware, provided by ROS. In order to connect the embedded computer with the mobile base we created own program for command interpretation and communication protocol.

4.1 ROS code organization

Figure 3 illustrates the organization of nodes and messaging flow of robot control. The main node DRIVE_CONTROL is controlling the motor, wheels, fans and the nozzles. DRIVE_CONTROL is subscribed to laser scan data from LIDAR_SENSOR node. It uses the laser scan data to make decisions about the optimal path of the robot.

Node OBJECT_DETECTION is taking care, like the name implies, of object detection. It gets the images from CAMERA_SENSOR node which deals with onboard digital camera. Object detection procedure uses a color segmentation technique to separate the background from the known objects. If the object is detected, then the message is send to DRIVE_CONTROL node, which properly reacts to the detected object.

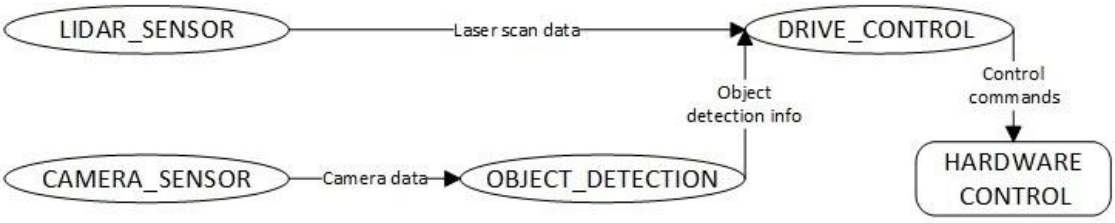


Figure 3 ROS nodes and communication diagram

4.2 Drive control algorithm

The navigation algorithm relies on the distance measurements from the LIDAR sensor. It detects obstacles up to 4 m away and then decides what is the optimal path the robot should take. Once the robot reaches the end of the crop line it uses data from the on-board compass and turns in the next row.

The procedure of finding an optimal path consists out of multiple steps.

As seen on Fig. 4 the**Napaka! Vira sklicevanja ni bilo mogoče najti.**algorithm uses rigonometric functions to calculate the distance of a segment between every two sequential points from LIDAR sensor. The segment distance must be at least 43 cm (for robot to drive trough). The segments that does not meet the criteria are disposed. **Napaka! Vira sklicevanja ni bilo mogoče najti.** Fig. 4 shows red colored segment istances written as: d_1, d_5, d_6, d_8 , d_9, d_{13} which are disposed and green colored distances: d_2, d_3, d_4, d_7, d_{10}, d_{11}, d_{12} which are retained for further procedure.

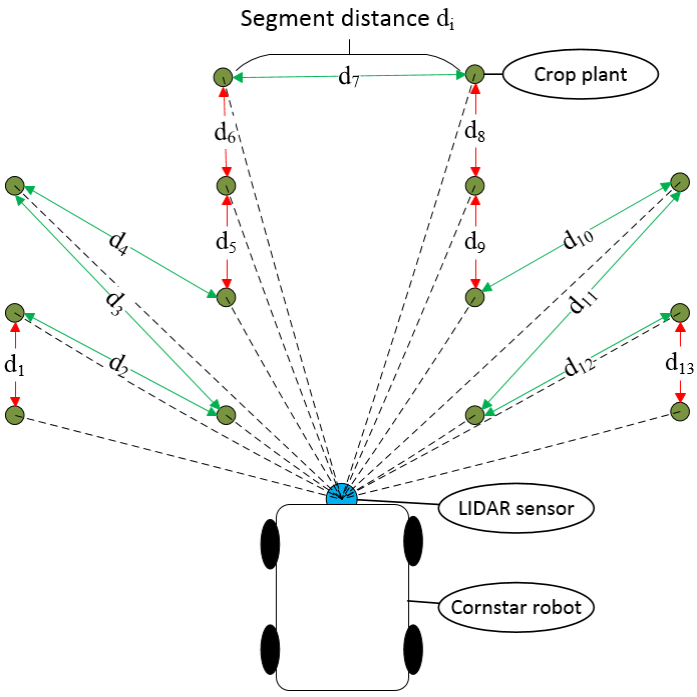


Figure 4 Robot calculating segment distances between every two sequential points from LIDAR sensor.

Figure 5 shows that only segment d_7 is appropriate for robot to drive trough. The procedure eliminates inappropriate segments as seen on Figure 5. Algorithm calculates the angles: α, β, γ in triangle limited between two sequential points and LIDAR sensor. If any of the angles α or β is bigger than predefined threshold (for example 100°), the segment is disposed. Disposed segments are marked with red triangle on Figure 5 and the optimal segment d_7 which pass the criteria is marked as green triangle.

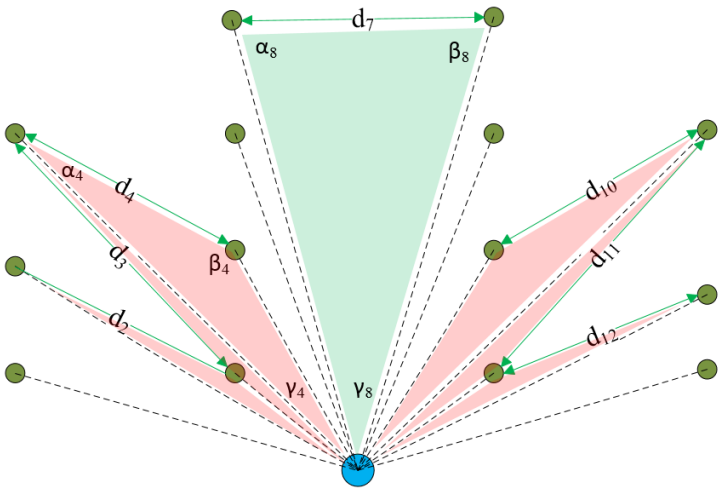


Figure 5 The procedure eliminates inappropriate solutions

When the optimal segment is determined, the algorithm calculates the angle for the front wheels to turn. The midpoint of optimal segment is calculated, for robot to drive towards. Angle δ on Figure 6 represents the angle for robot to turn front wheels to follow the midpoint of optimal segment.

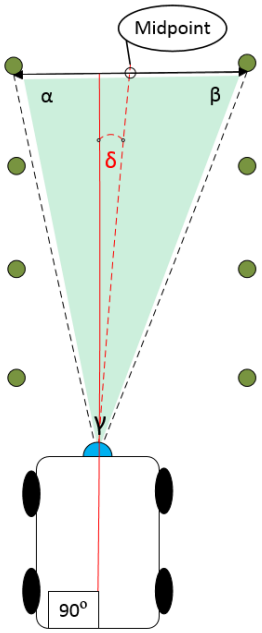


Figure 6 Calculating front wheels turn angle δ.

To prevent the robot from harming crop plants, when it is following an optimal path, a safety algorithm takes control, which adjust an optimal path to keep crop plants safe.

An algorithm is capable of dealing with various scenarios, such as missing or curved crop line and obstacle detection. In case of missing crop line, the orientation of existing line is calculated. Robot navigates forward, parallel with an existing crop line orientation.

Once the robot reaches the end of the crop line, it navigates perpendicular to crop line and enters a row-counting mode. A procedure for counting rows is based on calculating differences of the distance between closest crop plant and a robot in two sequential sensor measurements. Using the data from the on-board compass and row-counting algorithm the robot can navigate into any row that it is programed to.

5 Conclusion

The main problem we faced this year was the accuracy of the robot movement where the mechanical part should be replaced with a precise one. In addition it would be helpful to enable the robot to drive the robot back and fourth without actually turning the robot in the next row. Additional sensors such as odometry or IMU would also be welcome and are to be used next year in order to achieve a more settle and accurate movement of the robot.

Neaver the less with the improved version of the Cornstar field robot we are a step closer to a prototype robot that could serve as a small agricultural tool for various tasks. Of course this is only the first step toward building a universal tool that could do the everyday work quicker, more precise and without human interaction. We are satisfied with the current results, but still have a way to go.

6 References

[1.] R. C. Gonzales, R. E. Woods, 2001. *Digital Image Processing*, 3rd edition, Upper Saddle River: Prentice Hall PTR.

[2.] J. F. Thompson, J. V. Stafford, P. C. H. Miller, *Potential for automatic weed detection and selective herbicide application*, Crop Protection, vol. 10, pp. 254-259, 1991.

[3.] L. Shapiro, G. Stockman, *Computer Vision*, Prentice-Hall, Inc. 2001.

[4.] Bulanon, D.M., T. Kataoka, Y. Ota, and T. Hiroma. *A Machine Vision System for the Apple Harvesting Robot*, Agricultural Engineering International: the CIGR Journal of Scientific Research and Development. Manuscript PM 01 006. Vol. III.

[5.] Guo Feng, Cao Qixin, Nagata Masateru, *Fruit Detachment and Classification Method for Strawberry Harvesting Robot*, International Journal of Advanced Robotic Systems, vol. 5, no. 1, 2008.

[6.] Grisetti, G., Stachniss, C., Burgard, W., *Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters*, IEEE Transactions in Robotics, Volume 23, pp. 34-46, 2007.

[7.] Kohlbrecher, S., Meyer, J., Peterson, K., Graber, T., *Hector SLAM for robust mapping in USAR environments*, ROS RoboCup Rescue Summer School Graz, 2012.

[8.] Stone, H. S., *A Fast Direct Fourier-Based Algorithm for Subpixel Registration of Images*, IEEE Transactions on Geoscience and Remote Sensing, Volume 39, Number 10, pp. 2235-2242, 2001.

DTUni-corn - Technical University of Denmark, Denmark

Simon G. Larsen, Joakim Langkilde, Mads Hjertsted, Signe Munch

Supervisors: Nils A. Andersen, Ole Ravn

Automation and Control, DTU Electrical Engineering
Elektrovej Build. 326
2800 Kgs. Lyngby, Denmark
or@elektro.dtu.dk

1 Introduction

This report describes the development of a Field Robot, for competing in the Field Robot Event 2015. The event was held at the University of Maribor in Slovenia.

The main objective was to win the competition or at least to finish with the three best teams, the result was an overall 2nd place thus fulfilling the second objective.

2 Mechanics

This year's robot is based on the hardware from the previous year's competition. The robot configuration is the same, with significant software changes and a new sensors as cameras.

The robot runs DTU Mobotware, a robot operating system developed at DTU. The operating system consists of several servers and plugins, which all collaborate in to make the robot move.

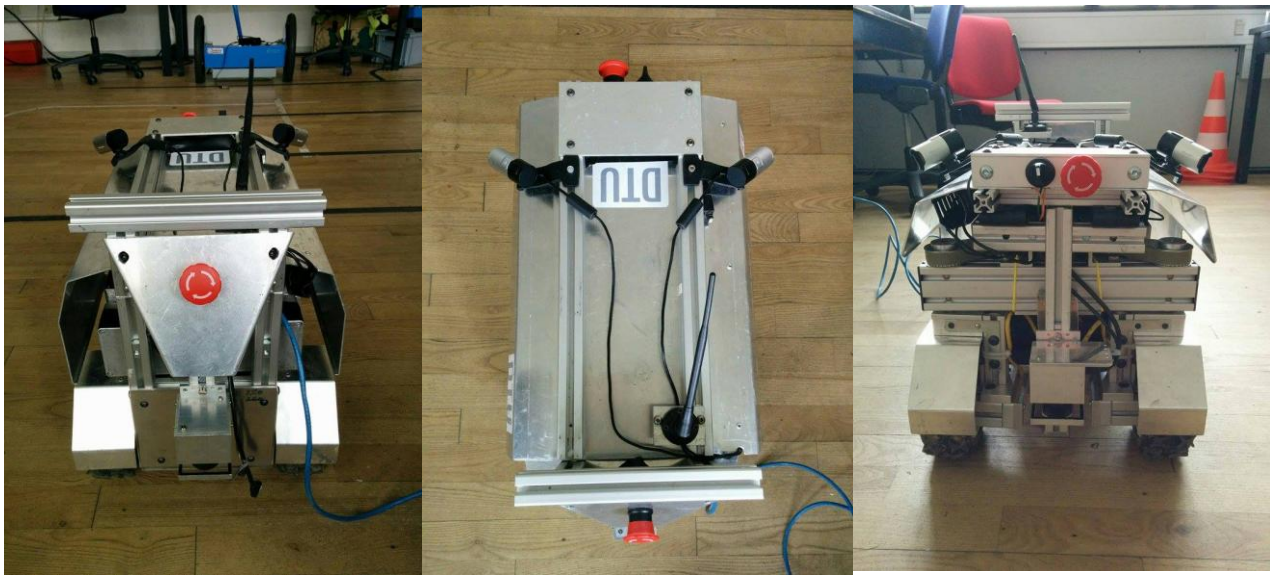


Figure 1: Picture of Field Robot. Right picture is robot seen from back. Centre picture is robot seen from above and the left picture is robot seen from the front.

The robot mechanics and hardware is described in [4]

3 Software

3.1 Dried plant detection

One of the objects of task 3 is to detect dried plants, which in this contest are simulated as regular corn plants with add of brown spray colour. In Figure 2 is shown a picture taken by the camera that is used for detecting the plants.

Figure 2: Picture of row of corn with dried plants between fresh ones.

The obvious way to detect these plants is by use of camera. Two Microsoft LifeCam Studio webcams were chosen for finding all the plants during one run. These cameras can deliver 1080p and is connected to the computer through Hi-Speed USB. To minimize data needed for the image processing a resolution of 240p were chosen.

A practical issue encountered by using two cameras is that the USB bandwidth on the Fieldrobot is not large enough to run both cameras at the same time. The solution is use one camera at a time. This is done by only looking at one row of corn during a row drive and then at the end of the row switching to the other camera such that when the robot drives down the next row it will check the row of corn between the two rows. Unfortunately the switching between the cameras takes quite some time and it is therefore necessary to wait at the end of a row for the switching to finish.

Another practical issue with the camera is that even though the resolution is very small and the frame rate is minimized to 30 fps the pool list is still getting stacked up, which means the picture that are being processed isn't necessarily the picture corresponding to what is positioned next to robot. This issue will of cause increase proportional with the size of the stack in in the pool list. Worst-case scenario would be if the image being

processed actually were a picture taken from the other camera due to the delay. One could try to solve this by decreasing frame rate or resolution, both solutions that would increase the chance of not seeing dried plant. Another approach is to angle the cameras to look ahead of the robot, thereby compensating for the software delay. Combining this solution with a drive speed of 0.5 m/s the point of reference will be at the back wheels of the robot. This point is also due some delay contributed by the way the Fieldrobot plays sound in order so signalling if dried plants are found.

The strategy of row drive while detecting dried plants is explained in Figure 3 below.

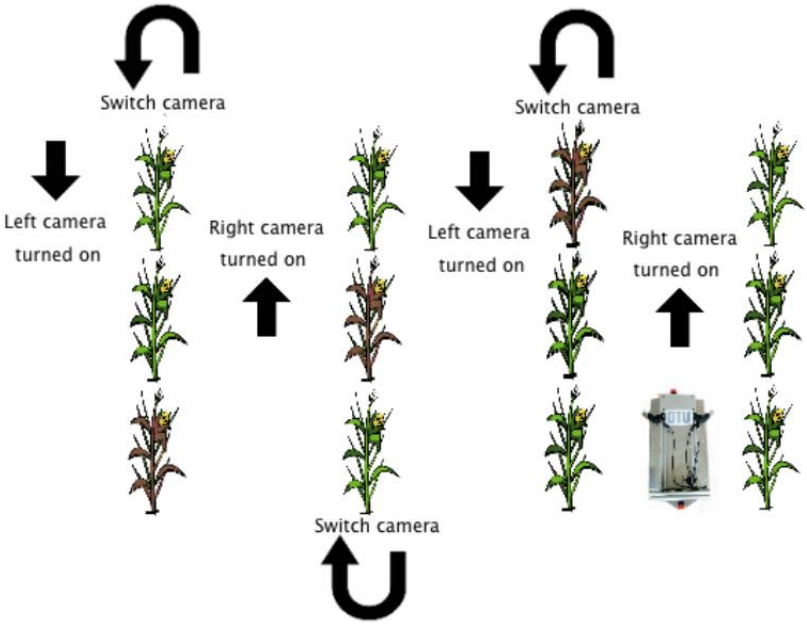


Figure 3: Strategy of driving trough rows detecting all dried corn plants by switching between right and left camera.

Detecting the dried (brown) plants is done by taking the top image of the pool list and then converting this to HSV colour-space, which is more tolerant to illumination changes and easier to extracting colour information from. The image is then run trough an OpenCV threshold function called $InRange(\ )$ which thresholds the image with respect to a minimum and maximum value of the Hue, Saturation and Value. The challenge in finding the brown colour is the perceived luminance, which can change a

lot. Therefore calibrating these parameters is a question of finding the right Saturation and then finding the relating Value that corresponds to the perceived luminance. Choosing these parameters one must also face the compromise between the chances of detecting false plants versus the change of not detecting right plants.

After thresholding, the image there will still be some noise from small objects that are within the HSV threshold values. These are removed from the resulting image by a $erode(\ )$ function followed by a $dilate(\ )$ function. The resulting steps of the image pre-processing are shown in Figure 4 below.

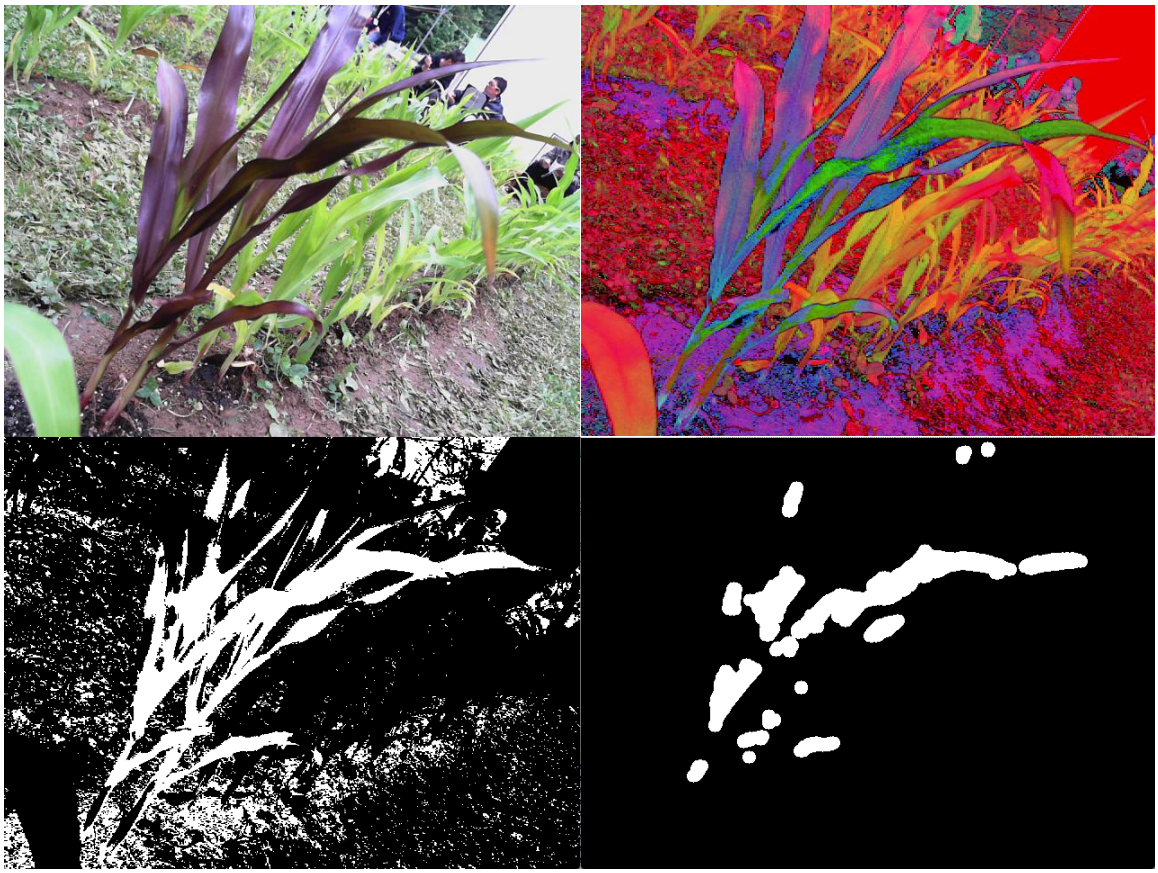


Figure 4: Picture of the image pre-processing steps. The original image is shown in the (Top Left Corner). The HSV transformation can be seen in the (Top Right Corner). The result of the threshold function can be seen in the (Bottom left corner). Finally after the removal of the small objects the resulting image can be seen in the (Bottom Right Corner).

When the image pre-processing is done, the resulting image is ready to be evaluated in terms of whether there is a dried plant or not. Calculating the moment of the resulting shape in the image does this. The C++ class $moment$ has an entity called $m00$, which holds the area of the shape. This area can be used as a limit to decide whether it is a dried plant or not.

The Boolean result of the dried plant detection is then send back to SMRCL script as a variable to be used to play audio and as a parameter to the $aurosebot$ plugin, where the mapping of the field is done.

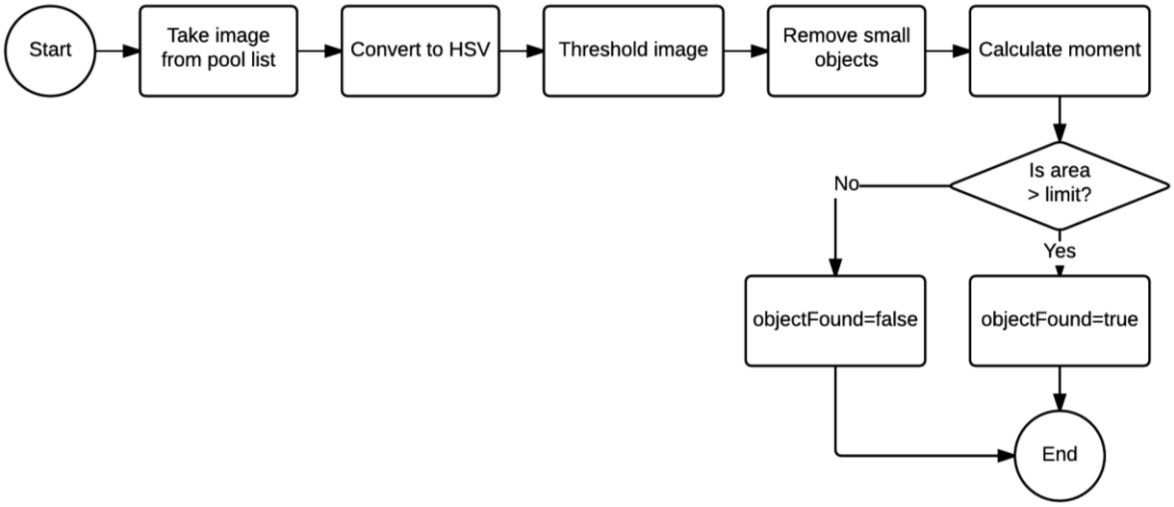


Figure 5: Flowchart of dried plant detector.

Figure 5: Flowchart of the plant detection is shown.

The actual implementation of this is done in an existing plugin called $auweed$. The command for running the algorithm is "$vision\ weed\ img = "cam"\ cornCheck$" where cam can be 0 or 1 depending on which side the robot is looking. The vision part of the command of cause implies that the function is run through the ucamserver. The result of the algorithm can be seen in the variable $vis1 from the plugin.

The two webcams are run through the $auv4lgst$ plugin, which are initialised with desired specification in the $ucamserver.ini$.

3.1.1 Partial conclusion

The implementation of the dried plant detection went really well. We were the only team that were able to detect all six dried plants. There were no false detections and the point of reference was very precise with the positioning of the robot relative to the dried plants. Part of the success is due to the calibration of the parameters. This was done five minutes before the beginning of the task. We choose our parameters to ensure no false detection and some chance of detecting 1 dried plant, but in the actual task there were three dried plants next to each other, which meant that the calibration were perfect in terms 100 % detection of dried plants.

Optimizations in this kind of task would be some way of calibrating the threshold values trough measuring the illumination level either by a separate sensor or with the camera. Increasing the speed would not be an optimization recommended from the experience we had with this task, but if still desired, maybe a look into the USB bandwidth-issue could optimize this. In addition, a better/faster way to play audio instead of using the *play* command would be preferred.

3.2 Sunlight avoidance

When driving the Fieldrobot outside in the sun we encountered some difficulties with the laser scanners. We observed when running Task 1 the robot sometimes saw a row even though there were none. This observation was verified when the robot constantly saw obstacles in Task 2. At first, we thought this was caused by high grass, but the issue continued even after adjusting the height of the laser.

Going through the laser data from the auclient we saw noise occur randomly throughout all the laser measurements. Analysing laser measurements in sunlight versus no-sunlight a possible cause of the noise could be sunlight.

This was confirmed when looking at the datasheet for the laser scanner. We saw the light source was a semiconductor laser diode with a wavelength of 785nm. Comparing this with the spectrum of sunlight seen in Figure 6, we see that the wavelength of the

used laser is very concentrated in natural sunlight. It is therefore obvious that the laser will intercept some of the beams from the sunlight.

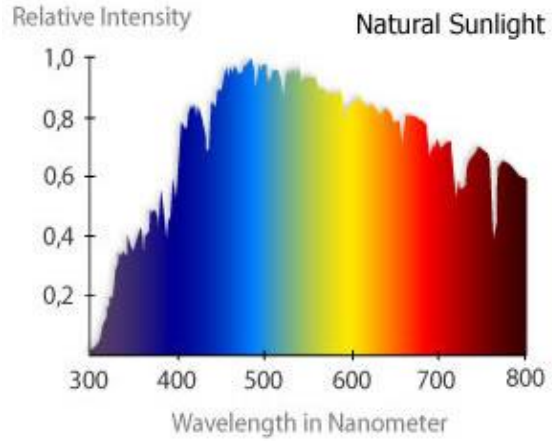


Figure 6: Relative intensity of natural sunlight.[1]

A way to solve this is trying to avoid the laser from seeing sunlight. This can be done by shading the laser but due to the reflective property of light, it is very hard to avoiding all the sunlight. To insure maximum protection, the shading of the laser needs to be both on the top and on the bottom of the laser, while the width needs to be big enough to avoid sunbeams from the sides.

Another approach to handle the sunlight is instead of avoiding it one could try to compensate for it through software. By this is meant a filter.

Analysing the presence of the sunlight in the laser measurement data one can observe that the sunlight mostly occurs in a group of 1-4 data points. This is illustrated in Figure 7 below.

[1] http://www.a-levelphysicstutor.com/wav-em-waves.php

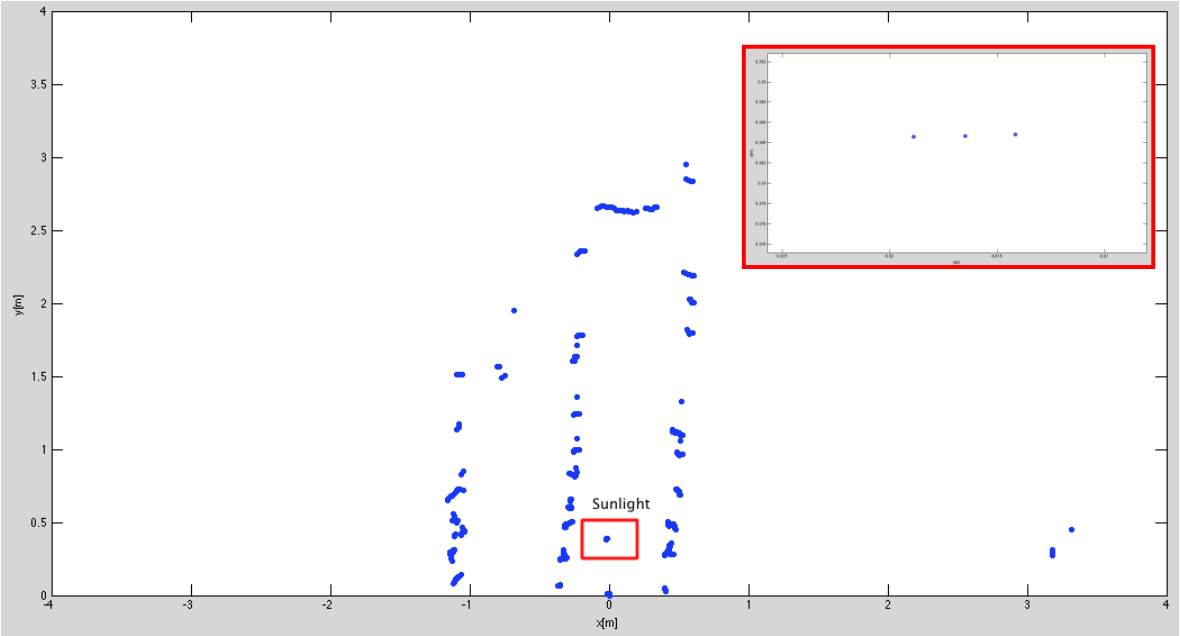


Figure 7: Image of laser data, which is plotted from MATLAB. The robot is placed in 0,0, heading into the row. In front of it is a marked box where the observed sunlight is present. By zooming, we see the sunlight occurs in a group of three data points.

The actual laser measurements of the plants are mostly observed in groups of 5-30 data points, these will of cause also be affected by the filter and it's therefore necessary to consider the cost of losing wanted data by deleting unwanted data.

For the implemented filter, groups of up to four data points is filtered out, still leaving the row detection unharmed. The filter algorithm is as follows.

First the algorithm gets the range r from the laser to the detected object, if the difference in range for the current point and the range for the three previous $r_{k-1}, r_{k-2}, r_{k-3}$ and three next laser points $r_{k+1}, r_{k+2}, r_{k+3}$ are not within 10 cm the laser measurement are put in $(0,0)$ which means it won't be used anymore. Also if some laser measurements are closer than 10 cm these will also be put in $(0,0)$. The algorithm is implemented in $aurosebot$ plugin and can be seen $ufuncrosebot.cpp$.

The result of the algorithm can be seen in Figure 8.

As wanted most of the disturbance from the sunlight is now deleted and the only thing remaining in front of the robot are the measurements from the rows of corn.

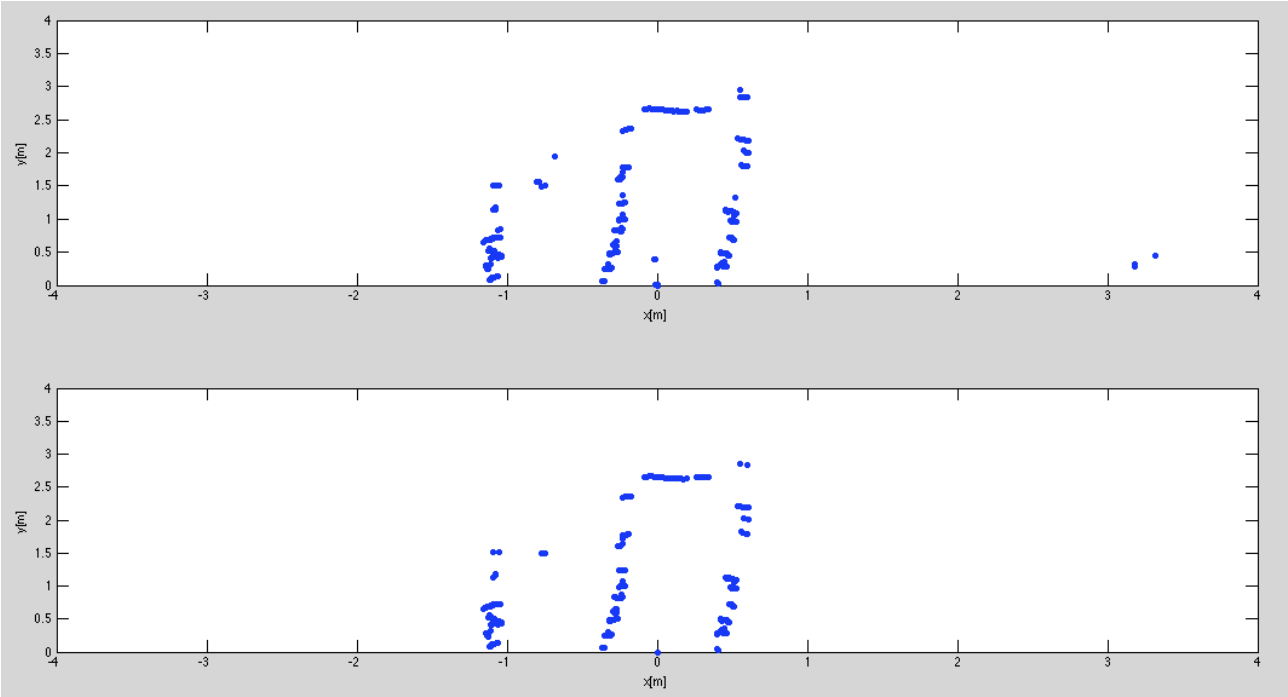


Figure 8: The plot at the top shows the original laser data, while the plot at the button shows the effect of the filter algorithm.

3.2.1 Partial conclusion

The implementation of the algorithm was a success. The robot is now able to drive in sunlight with minimum disturbance, though it's still recommend to shade the laser if the sunlight is really strong which maybe will affect the sunlight data points to occur in groups larger than 4.

3.3 Driving backwards

The code from last year implemented Task 2 to be able to drive backwards. However, the testing did not go as expected. Reviewing the code exposed several problems.

First, the laser device setup in the ulmsserver had to be changed, so that the two lasers were recognized probably. Then the row drive algorithm needed a brush-up. It turned out; they used the same transformation from the backwards laser as for the forward laser.

So in order for the robot to drive backwards through the row, a couple of changes were made. The rosebot plugin was, from the earlier years, able to return a line, given in odometry coordinates. This is due to a transformation first from the laser to robot, then from robot to odometry. When driving backwards, this was not the right transformation, and therefore a new transformation was needed when driving backwards.

When driving forward, the transformation from laser to robot is simple. The laser is pointed in the same direction as the robot, and there is only an offset in the x-direction.

This is different when going backwards, as there is a rotation of π as well.

The standard transformation is given by:

$$\begin{pmatrix} x_n \\ x_n \end{pmatrix} = \begin{pmatrix} \cos(th_0) & -\sin(th_0) \\ \sin(th_0) & \cos(th_0) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

$$th_n = th + th_0$$

As there is a rotation of pi, and a displacement in x:

$$Line_{x[robot]} = line_x * \cos(\pi) - line_y * \sin(\pi) + laserx$$

$$Line_{y[robot]} = line_x * \sin(\pi) + line_y * \cos(\pi) + 0$$

$$Line_{th[robot]} = line_{th} - \pi$$

This was implemented by giving the option $backward = 0\ or\ 1$ in the rosebot command.

In the plugin code, a simple if-else was added, to implement the new *backward* feature.

Afterwards, the transformation from robot to odometry is the same. Now the *aurosebot* plugin can handle the backwards laser just as well as the forward laser.

Testing this showed some problems, there seems to be a problem in the *mrc*, that made the robot drive backwards bad. This was due to a sign mistake in the mrc code.

Changing this and recompiling, the tests showed a positive improvement and the robot drives almost as good backwards as it does forward.

3.4 Task changes

Changes to the code for each task are described below.

3.4.1 Task 1

In Task 1 the robot had a special way of turning where it drove out the row, made a backward 90 degrees turn and then continued going straight backward until it saw the row it just came from. It would then drive forward until it sees the next row and then make a 90 degrees turn into that row. This was a very slow and inefficient way off of turning which needed to be changed.

The most intuitively turn one can think of is a 180 turn, the disadvantage though is that we don't know the angle by which the robot leaves its current row, thereby having the possibility of the robot not getting straight into the next row. This could also be caused if the rows are not aligned, which cannot be relied 100%. Therefore, in order to make the turn function reliable, the 180 turns must be combined with some knowledge of the robots location. The way to do this is to make the robot turn some of the way and then turning the last degrees while running the *aurosebot* plugin to drive into the row. To adjust the turn one can easily change the degrees of the two *turnr* commands as well as their radii. Right after the robot sees the end of the row there is a drive offset, which also can be adjusted. To handle situations where the robot does not see the row after a

turn, some fault tolerance has been implemented. If the robot sees a dead end after a turn, it is most likely a plant in front of it. It should therefore drive 0,1 $meter$ backward and take a new measurement. On the other hand, if the robot sees open space after a turn, one could think the robot is standing towards the entrees of a row, but is too far away for the $aurosebot$ plugin to handle it. The robot should then drive forward 0,10 $meter$ and take a new measurement.

Using a gyroscope instead of wheel odometry is because of the soft and uneven ground, which does not affect the gyroscope, but can cause a big odometry error.

3.4.2 Task 2

Compared to last year, the same implementation of driveplan was used as this seemed as an optimal starting point. The turning, obstacle check and general driving was much simplified and improved.

For checking the obstacles, an improved check was implemented. If a dead end is detected, as this could happen because of random noise/leaves, the velocity is set to the slowVel. Then, a further check is done with the mrclocate and then, if an obstacle is detected the robot backs out of the row.

Turning functions was improved to implement both the same rowturn as Task 1, but for turning more than one row the robot turns 90 degrees and then looks for the rows, and counts them as it goes past them. Then when the correct row is reached it turns down that row.

When going backwards out of a row, the turn is also different from the others, as it needs to go forward down the next row.

For backwards driving a lot of the code was optimized, so different variables controls which laser to use, and which direction to drive. Of course the code in the rosebot plugin was also changed to implement a proper backwards driving function as described in the earlier chapter.

3.4.3 Task 3

Task 3 was changed quite a bit, as it was not possible to use two cameras simultaneously. Instead the implemented smr-cl code drove as in Task 1, with the exact same turns but a slower speed in order to detect the weed. The script has several initial values:

Variable	Value	Integer	Description
Sign	R/L	-1/1	First turn is right/left
Cam	camR/camL	0/1	First cam to use
turnLim	0-99	0-99	How many turns before stopping

The robot detects the plants by, one row at a time looking at one side. Figure 9 shows the idea.

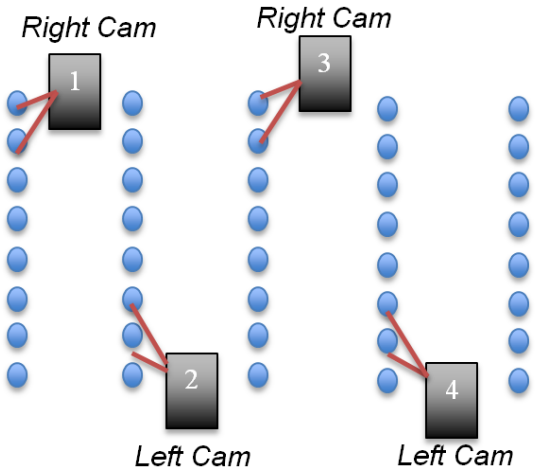


Figure 9: The figure shows how the weed detection is implemented in Task 3.

In order to detect all the plants on the field, it is necessary to drive one more row than what the task describes. As the camera to use was changed each row it was not possible to implement the camera plugin as a push command. So instead the rowdrive command

was modified so that it took a new picture for every 4th rowdrive command. This was to slow the update of camera down a bit, during testing problems occurred when polling the camera to fast, as the pictures would then queue up, and an increasing delay would destroy the whole idea of detecting plants.

Label to switch the camera was also implemented, which is some pretty simple code.

When weed was detected, a command was sent to the rosebot plugin to signal a weed detection. This is used for the map.

To signal the user, and crowd during the competition, a sound was implemented. To make the sound only play once, even when several pictures contained a positive detection, a simple algorithm was implemented in smr-cl code to prevent the sound from playing multiple times.

4 Discussion

4.1 Task 1

As explained in the introduction the purpose of Task 1 was to drive as far as possible in three minutes. 1 point was given for each meter passed while 2 points were subtracted if the robot drove down a plant or the team had to touch the robot in order to get it on the right track. Our approach to this task was to complete it in the most autonomous way without interfering with the robot and without any plants destroyed, meaning not necessarily being the fastest robot.

The robot performed well in Task 1. We were able to drive $97\ m$ which gave us an average speed of $0.53\ m/s$, this was a little slow compared to the fact that the robot had a speed of $1\ m/s$ down the first $11\ meter$s of the rows whereas the last $1\ meter$ the speed was reduced to $0.8\ m/s$ to insure detection of end of row. This means the largest loss of speed was in the turns. We didn't expect to lose that much time in the turns because they actually were meant to be driven with a speed of $1\ m/s$, but the reason they took longer time was because the robot didn't see the next row immediately. To

handle a situation like this, we had implemented some code to make the robot go a 0.1 m back or forward depending whether the robot saw a dead end or free space. The implementation of fault tolerance in the turns were a great success and should not be missed out, however the execution time were too long due to some *wait* statements between the situations encountered. Nonetheless, the robot made a stable performance with no plants hit and with only one little push. Comparing with the other participants in this task it is not an exaggeration to say we had one of the most stable and autonomous robots present since the other robots hit many plants and the teams interfered excessively many times with the robot.

4.1.1 Suggestions for optimization

For the robot to increase its average speed one could try reduce the handling time of the situation where no row is found after a turn, this could be done by removing or decreasing the *wait* statements and checking if the stability of the turns are the same. Another solution is making a different type of turn. We were pleased about the 180 turns our robot did, but another team had their robot driving out of the row and then switching to the back laser, which enabled them to drive backwards into the next row, and thereby completing task 1 by alternately driving forward and backward through the field. This is definitely a method worth testing. Trying to increase the speed of the row drive is not recommended since it will affect the stability of the performance.

4.2 Task 2

Task 2 did not go as well as expected. Minutes before the task, a possible problem arises: The cone, which was to act as the obstacle, was placed in the middle of two holes. This was NOT an optimal position for the algorithm and code that was implemented, but none the less it was our robots turn to compete. The robot drove very steady down the rows and had no problems following the driveplan and turning multiple rows but when the obstacle was reached it did not detect it, as expected and feared. Due to the obstacle being placed in the middle of a row with a hole on both right and left

side, the robot actually tried to avoid it; it read a hole at either left or right side of the cone, as it was actually able to drive through the hole as illustrated on Figure 10.

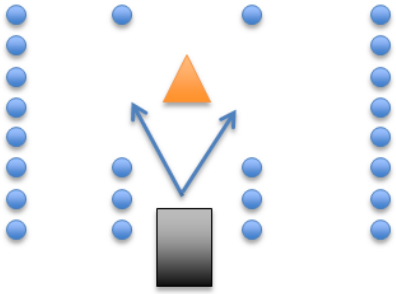


Figure 10: The problem that occurred in task 2. A cone was placed with holes on either side, and the robot did not see the obstacle because of a possible way around the cone.

A 7th place was reached even though the major mistake occurred.

4.2.1 Suggestions for optimization

In this case, it was not really our fault. It was not expected that the obstacle would be placed in the middle of the holes, which our code was not able to handle. The best advice to give to next year contestants is, always prepare for the worst! Small things can change, and you should be able to do it anyways. In our case, just picking up the robot would actually destroy it, as the laser would return open space, and the robot would initiate the turning process.

A great improvement for this task and the others is to make the turning depend on how far the robot has gone, so that it cannot turn until it has driven a specific distance. Furthermore, test before the competition what happens if you have to pick the robot up or change its orientation. So that you have an idea of what happens.

4.3 Task 3

This task went very well at the competition and a 1st place was collected!

This is mainly due to the image thresholds. Those was found just a few minutes before the robot had to drive in the competition, and they are an important part of the task, as each plant detected gives 2 points. All plants were detected and not a single missed.

The *auweed* plugin functioned very well and in terms of driving the robot, there was almost nothing to improve. The robot navigates very steady through the row, and the fault tolerant turning method made it easy to drive through Task 3.

The map that was turned in during the competition was accepted, and 5 points was given. The plants were not mapped on the map, which probably is a minor mistake in the code. The ulmsserverlog indicates that the plants were found, so they have simply not been plotted onto the final picture.

Suggestions for optimization

The best way to correlate the map would be using Matlab afterwards. This simplifies the plugin code significantly, decreases CPU use and makes changing the code much simpler. The robot would then act as only a measurement collector, and the computer afterwards can do the real processing.

Several teams used a much simpler method for obtaining a map of Task 3. By using a pre-defined layout, then filling in the distance driven and the detected plants the map is created. This is a method that is much easier to implement, but it does not address the real idea behind this Task, which is to map an unknown landscape by following rows of corn.

5 Conclusion

As presented in this paper some improvements has been done from last year's work. The most significant are the improved turn method, which definitely improved the performance in terms of speed and stability, and the enabling of the backward drive, which unfortunately did not, came to use in Task 2, due to some unlucky event.

In all three Tasks the driving through rows were very stable, with no plants hit and no need of interference, this was our main goal of the participation of FRE2015.

As discussed previously, Task 1 went really well with a 3th place. The only possible improvement is some speed optimizations in the turns. Combining this with some stricter penalty points, we cannot see any reason for the robot not having a top three ranking next year in this task.

For Task 2 we failed locating the obstacle, we conclude that this was due to the placement of the obstacle, which made the robot see open space and thereby starting turning. If the robot had seen the obstacle we have reason to believe that it would have finished the given drive plan without any interference. This is of cause something that needs improvement for next year participation.

The dried plant detection in Task 3 could was very successful. As the only team, all six dried plants were detected while having no false detection. Our map got 5/10 points, insuring a 1st place for this task. The missing 5 points were because of the map did not had the dried plants on it. This is due to some minor error in the map function in the *aurosebot* plugin, which should not be too hard to find and fix. In conclusion, a simple and fast algorithm, and our calibration of the thresholding parameters, which were done just before our run, to insure that they matched the light conditions, caused the very successful detection of the dried plant.

The overall evaluation of the Field robot event 2015 competition is very positive. We are pleased with our performance and preparation from home, but also with this year organizer of the event, who had made sure that each team had their own small workplace with a table and benches. However, the penalty for interfering with the robot in the competition should be made more severe. After Task 2 where robots that were only able to drive straight forward, received more points than robots that could actually navigate and drive autonomously trough the field.

6 References

[1.] Siegwart, Nourbakhsh, Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed

[2.] Beck, A. B., Andersen, N. A., Andersen, J. C., & Ravn, O. (2010). Mobotware – A Plug-in Based Framework for Mobile Robots. In IAV 2010. International Federation of Automatic Control.

[3.] Field Robot Event 2012 - task 1, Rosebot winner of the championship

[4.] Proceedings of Field Robot Event 2014

GroundBreaker – Aalto University & University of Helsinki, Finland

Laine[1], Jussi Linko[1], Tuomas Ruotsalainen[1], Juuso Autiosalo[2], Alexander Brink[2], Teemu Koitto[2], Juho Kalliomäki[3], Kimmo Kankare[3], Eemeli Saarelma[3], Timo Oksanen[1,3*] & Jari Kostamo[2*]

[1]) Aalto University Department of Automation and System Technology
[2]) Aalto University Department of Engineering Design and Production
[3]) University of Helsinki Department of Agricultural Sciences
[*]) Instructor & Supervisor

1 Introduction

Field Robot Event is an agricultural robot competition held every year somewhere in Europe. In 2015 it was held in Maribor, Slovenia, and once again Aalto University and the University of Helsinki gathered an interdisciplinary student team to take part in the competition. The goal was to build a robot capable of reliably navigating a maize field without any human intervention. The robot was also expected to be able to perform some useful functionalities in the field.

The team consisted of nine students from three different faculties: three automation- and systems technology students from Aalto University's School of Electrical Engineering, three mechanical engineering students from Aalto University's School of Engineering and three agricultural technology students from the University of Helsinki. Due to the project's large scope, cooperation was vital for success - nobody could have the necessary technical knowledge and expertise for every aspect.

The robot was built from scratch, but extensive experience from previous years was used to define quite strict boundary conditions for the design. For example, everything from the basic structure to communication protocols and the simplest algorithms was given and thus the team had somewhat limited room for improvisation and new ideas. It's worth mentioning though, that all 3D-models were drawn from scratch and no line of code from previous years was reused. The basic idea was to build another robot from an already proven concept. The reason for this was to ensure a working robot for the competition and keep the project more educational.

The project lasted for 10 months, in other words a whole academic year. The autumn term was used to study necessary skills and do 3D design, while the building and programming was mostly done on the spring term. It proved to be extremely laborious feat, and every member was forced to make some sacrifices to free time, work, and other courses in order to complete the robot on time. However, plenty of practical

experience in technical problem solving, teamwork, interdisciplinary cooperation, to name a few, was gained as compensation.

2 Mechanics

2.1 2.1 Design

Some general features of the robot were already decided from the beginning. It was to have four wheels powered by two electric motors and have four-wheel steering with one steering servo per axle. The robot was to carry the load of the electronics, sensors and computers. Many of the components had been chosen in advance. These requirements made the start easier. We got to focus on details instead of trying to figure out the main concept. There was still plenty of work to be done.

The mechanics of the robot was designed by three of the team members. The task was to design and build the mechanics and mechatronics of the robot. Previous robots were carefully analysed and the best aspects were incorporated into our design. The frame of the robot was built using RC car parts, laser-cut sheet aluminium and machined aluminium. 3D models were done from scratch and we had to machine every part that couldn't be manufactured with laser. Wheels and rims were the only thing we borrowed from last year's robot in order to save some time.

The design started from the axle modules, which were the most complex part of the chassis and were therefore the first design challenge. DC motor, differential, gearbox, steering servo, electronics, cabling and connectors were all inside the aluminium boxes between the wheels. Although this was the starting point, everything had to be designed more or less simultaneously. The axle modules were connected to the suspension. The mounting points had to be figured out before finalizing the design. The suspension in turn was connected to the frame. So before finishing the suspension, the frame had to be figured out. Everything was connected together and had to be designed as a whole.

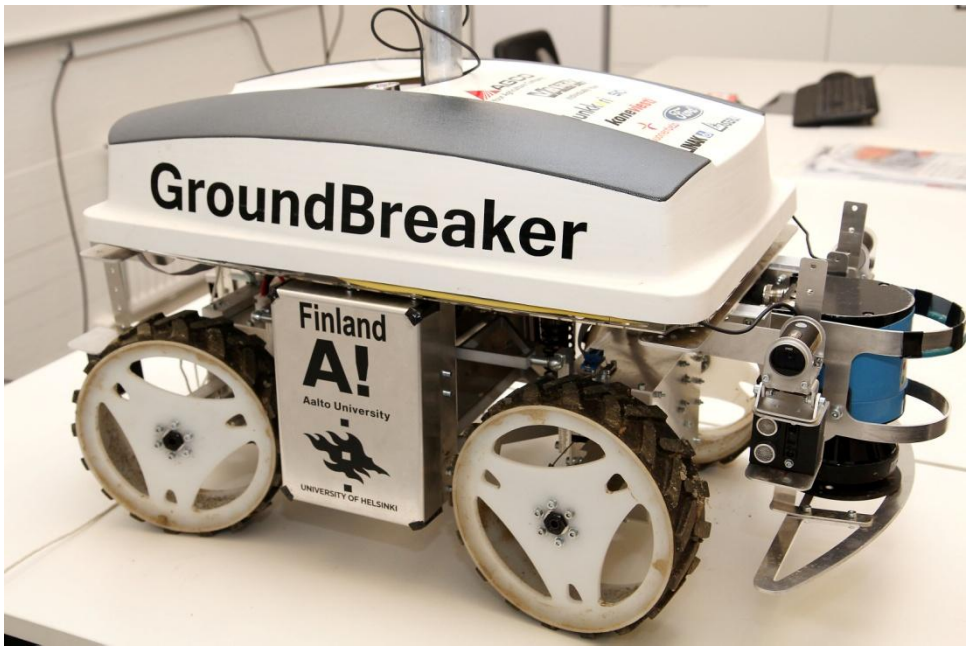


Figure 7 The finished robot

2.2 Type of chassis

The goal with the chassis was to create a stable tractor like frame that could reliably carry the components of the robot. A stable ride is required for many of the components to work properly. For instance, the encoder mounted to the motor axle depends on the wheels touching the ground at all times.

A tractor like chassis, where both axles can swing without resistance around the longitudinal axis, was recommended. There are many variations of this kind of chassis. Some designs have independently twisting axles or the axles can be connected so that twisting one axle twists the other one in the opposite direction. Especially the latter provides good traction and balance in very rough terrain.

The two strongest options in the chassis design were the one we called the monster truck chassis and the other one that had been used in previous field robot projects. In the monster truck version, the suspension for each wheel axle works independently. Front and back are not coupled together. The one that had been used in previous robots

connects the front and the rear together with horizontal springs on the side. When the front axle is twisted to the left, the springs pushes the rear axle to the right and vice versa. The springs together with dampers help to stabilize the ride.

The monster truck chassis is slightly simpler than the one with the springs on the sides. Pivoted rods connect the frame and the axle modules. The mechanism enables the axle module to move up and down and twist in a controlled path. Add dampers and springs on both sides of the modules and we have a working suspension. The disadvantage of this kind of suspension system is that the front and back axles work completely independently. On the other hand, the movement of the dampers is proportional to the movement of the wheels due to the direct connection. It is easier to find suitable and optimized dampers and springs. The difference is however barely noticeable and was therefore neglected when making the decision.

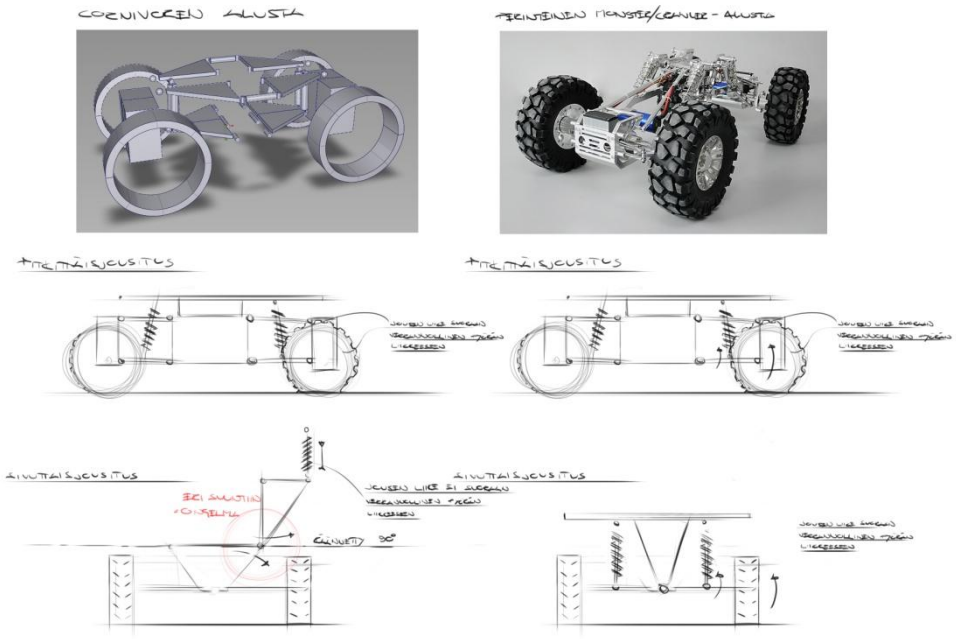


Figure 8 Comparing type of chassis

In the picture above you can see the two types of suspension systems we considered to use in our robot. The one on the left has the front and back suspension system connected

together with the horizontal spring. The monster truck type on the right has four pivoted rods per front and back connecting the axle module and the frame together. The sketches show the tiny disadvantage the slightly more complicated suspension mechanism brings. However, the advantage of having the axis connected together was considered so important, that the leftmost option was chosen.

As mentioned, the suspension system that we chose had been used in robots before. We had therefore a great advantage as we could analyse how it behaved and improve it.

Looking at the mechanism from the side, a parallelogram linkage provided the vertical motion of the wheels. For the wheels not to twist when moving up and down, the linkage arms had to be the same length. This was a feature to strive for.

The axle modules were pivoted along the length of the robot to provide suspension for the rolling motion. We discovered that placing the pivot axis as close as possible to the wheel axis was recommended. In this way sideways movement of the wheels is minimized when rolling.

A third feature that we realized played a quite big role was the wheelbase. The shorter the wheel base, the smaller the turning radius will be. This was something very desirable in the competition. A short wheelbase however, made the chassis design more challenging. The suspension needs space to move, the linkage arms, the springs and dampers all require some space. Our aim became to pack the mechanism into the smallest possible package, without compromising the performance.

Our first approach was to simply shrink down the mechanism and in that way create a shorter wheelbase. This was not an optimal solution. It affected the performance. This was however not the only limiting factor. The battery case on the side caused problems. The wheels would have hit the batteries if we shortened the wheelbase. The dimensions of the battery case is determined by the battery. In the original design the batteries were horizontal and placed perpendicularly with axles, enabling enough space for the

connecting horizontal spring that were situated above. There was not enough space to place the batteries in any other position unless you changed the design and got rid of the spring above, so that is what we decided to do. We thought about changing the mid-section of the robot and place the batteries parallel with the axles. Finally we came up with the solution to pivot the linkage arms on the outside and place the springs close to the center, inside the frame. This enabled us to make the mechanism considerably narrower without weakening the mechanism by bringing the pivot points closer together. This allowed us to place the batteries in a vertical position and therefore we could achieve a slightly shorter wheelbase.

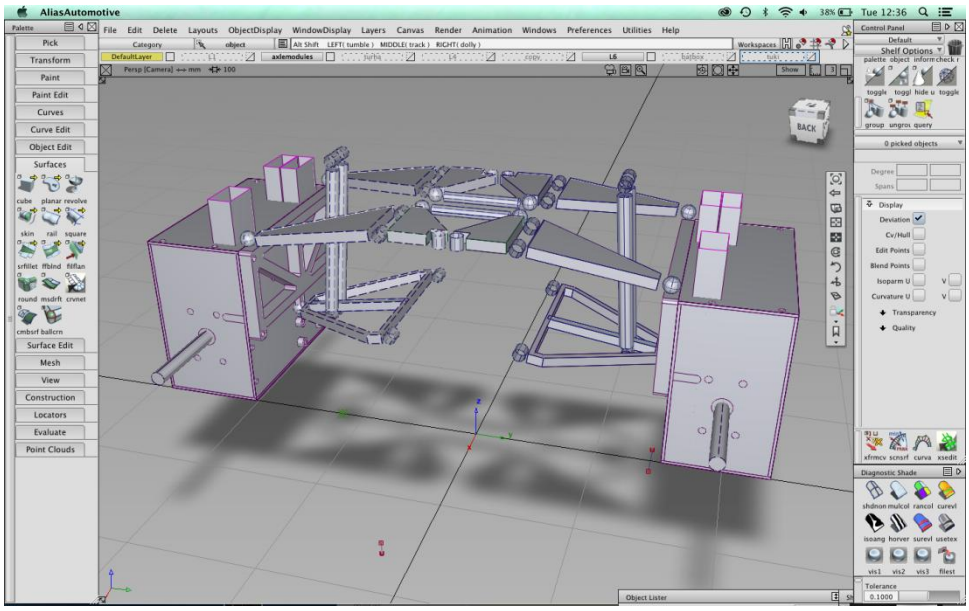


Figure 9 More space for the battery cases was created by placing the horizontal springs closer to the center

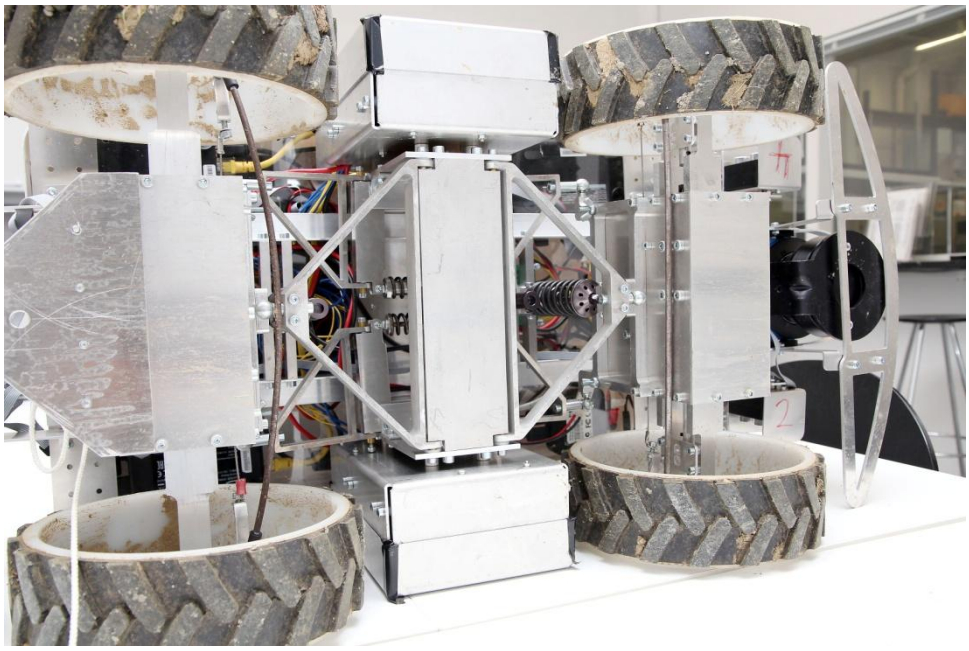


Figure 10 The rolling movement of the rear and front axis are connected through the horizontal dampers in the middle

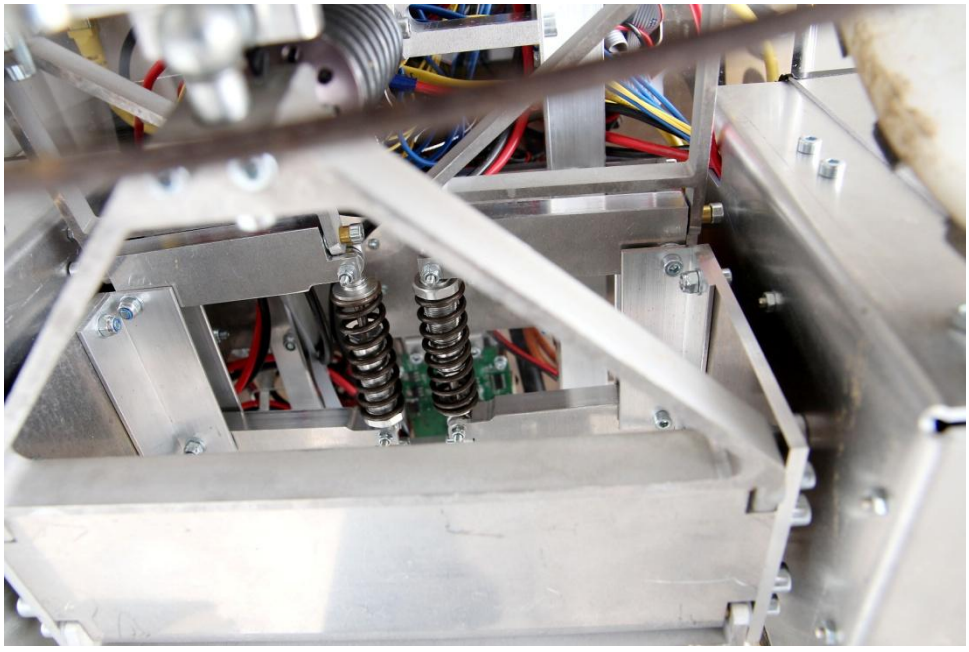


Figure 11 Placing the dampers in the middle enabled a shorter wheelbase

2.3 Axle modules

The wheels are powered by brushed 12 V RS-540 DC-motors, one per each axle module. To distribute the power correctly, a differential had to be added to each motor. This enabled the wheels to spin at different speeds. A gearbox was installed between the motor and the differential, coupling was achieved using a additional spur gear which increased the overall gear ratio. We estimated the loaded motor speed and dimensioned the gearing to generate a maximum speed of about 2 m/s. RC-parts were used to build the power transmission between the motors and wheels.

Steering is done by a Dynamixel MX-28 servo. Steering cable is connected from knuckle to knuckle through the axle module while doing a loop around drum styled servo horn. Because the steering cable can only pull, a tie rod was attached between the steering knuckles.

In addition to the mechanical parts of the axle modules, electronics were also fitted. Module was designed to be easily assembled and disassembled, electronics and DC motor can be quickly removed and replaced.

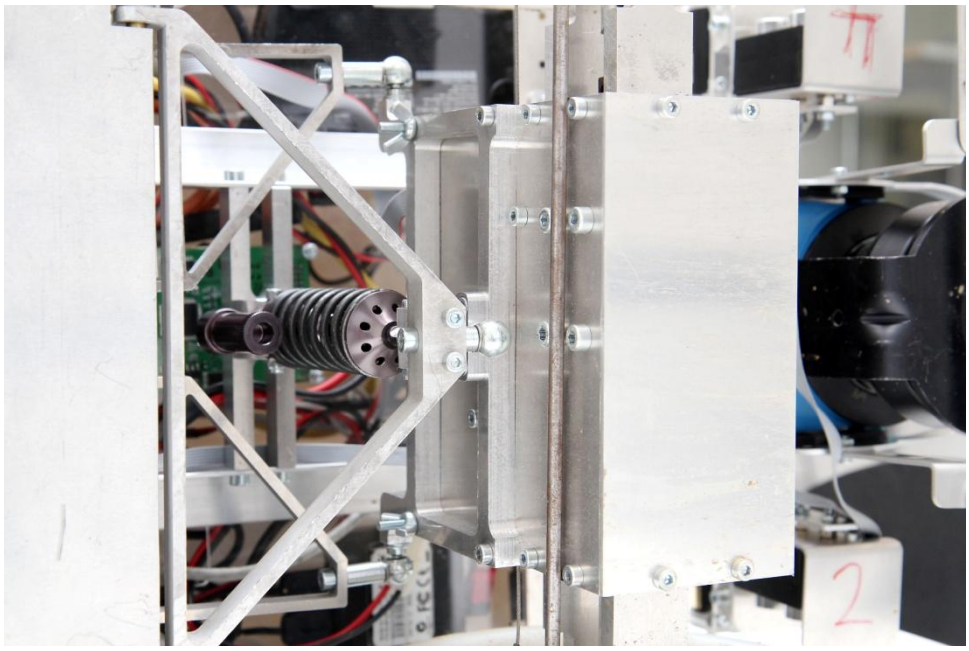


Figure 12 The axle modules hold a lot of technology that were enclosed with aluminium plates

2.4 Knuckles and Ackermann-angle

When a four wheeled vehicle like a car turns, a difference in rotating speeds of the inner and outer wheels occurs. This is because the inner wheels travel a shorter distance than the outer wheels, which turn in a tighter radius. Shifting the wheels the exact same angle when turning would therefore not be optimal. The inner wheels have to turn more. This is called the Ackermann steering geometry. Optimal solution would be if all the individual wheel axes would coincide at a single point when turning. This can be achieved by connecting the steering knuckles with a pivoted tie rod. The steering knuckles we used were spare parts for RC-vehicle. The problem with these knuckles is that they are designed for vehicles with only two steerable wheels and for a vehicle with a shorter wheelbase. In our case, all four wheels are steerable, resulting in a much tighter Ackerman angle which meant that the tires would have to slip when turning.

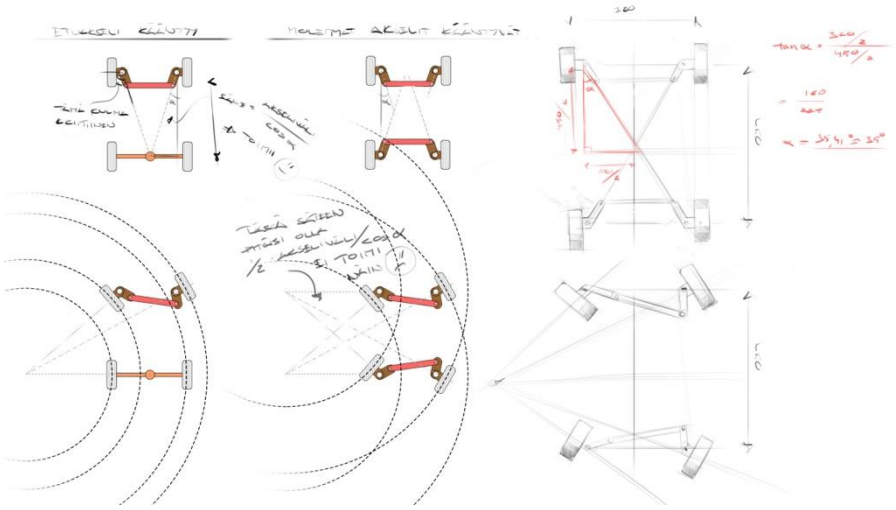


Figure 13 Ackermann geometry. A comparison of two and four wheels turning

Figure 14 The knuckle was connected to the steering servo with a cable and to the opposite knuckle with a tie rod enabling Ackermann-steering

2.5 Dampers and springs

Chosen chassis construction has two horizontal and two vertical spring-damper elements. Vertical springs are subjected to significantly greater loads as they must carry the load of the vehicle and dampen pitching motion. The horizontal springs, the ones that dampen the rolling movement, do not carry the weight of the robot and are significantly smaller.

The dimensioning of the dampers and springs was done by analysing components in previous models and calculating some damping forces based on the knowledge that the robot was going to weigh around 20 kg and geometry of the chassis. The weight prediction turned out to be spot on. The finished robot weigh 20 kg.

Choosing the spring stiffness was the most crucial task. More attention was therefore put on choosing the correct dimensioned springs. The dampers were chosen based on what was reasonable for the available space.

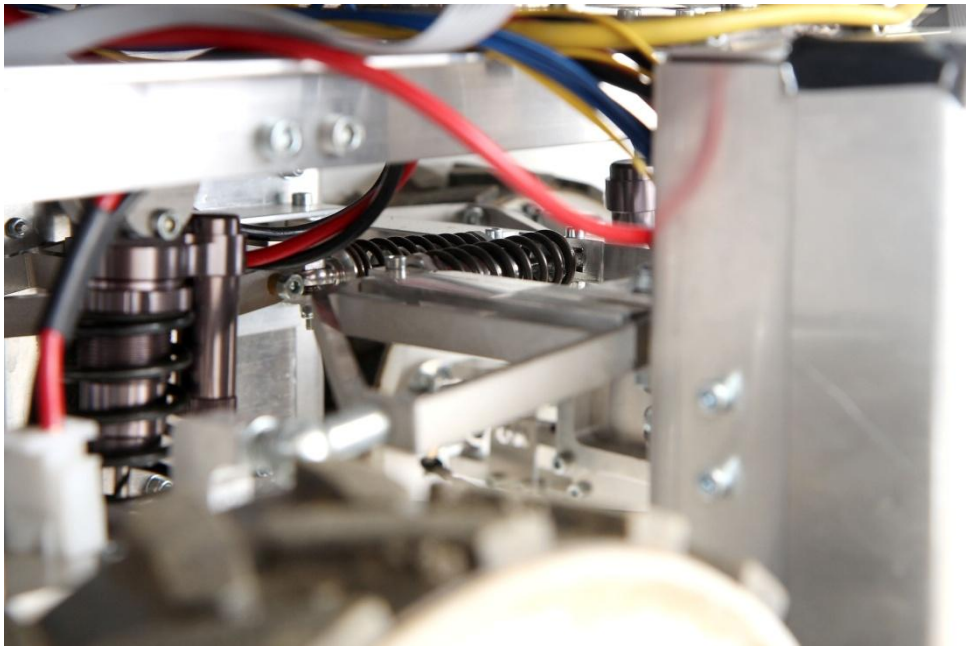


Figure 15 The vertical springs damped the vertical movement and pitching. The horizontal ones in the middles damped the rolling movement

2.6 Axle module quick coupling

The axle module holds a lot of components, both mechanical and electrical. Because of its complexity, easy access is preferable. When a malfunction occurs, it is a big advantage to be able to quickly dismount the module from the robot.

The main component that enables the quick coupling is the mounting plate situated behind the module. The linkage arms of the suspension system all connect to this. In this way, all the arms don't have to be dismounted separately when removing the axle module. It also keeps the suspension system intact and everything in place when the axle module is not in place. Beneath you can see sketches developed during the ideation of the quick coupling.

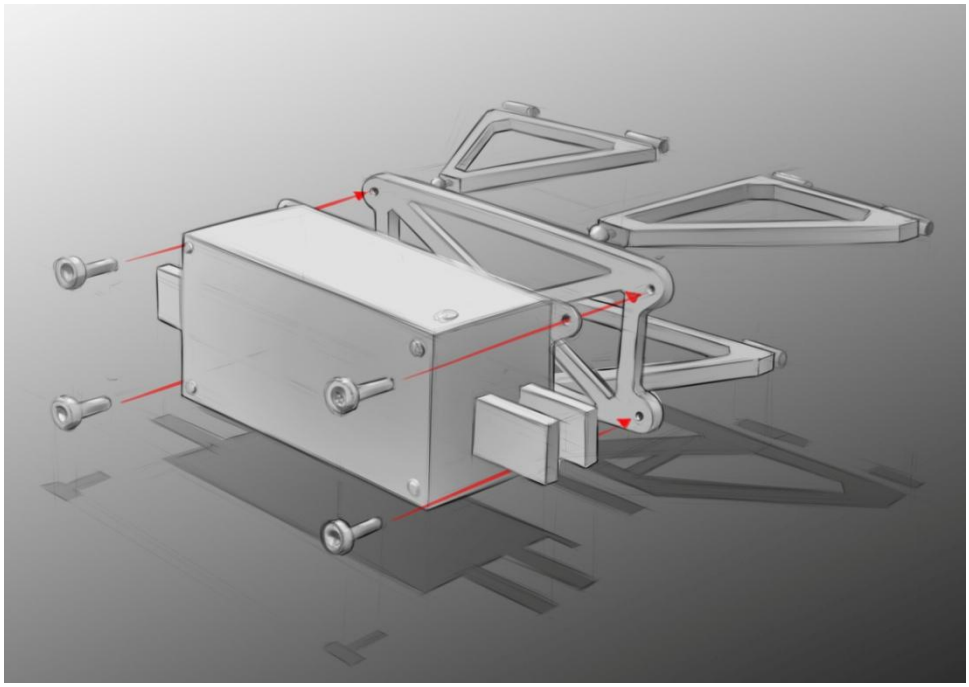


Figure 16 The mounting plate holds the linkage arms in place although the axle module is detached

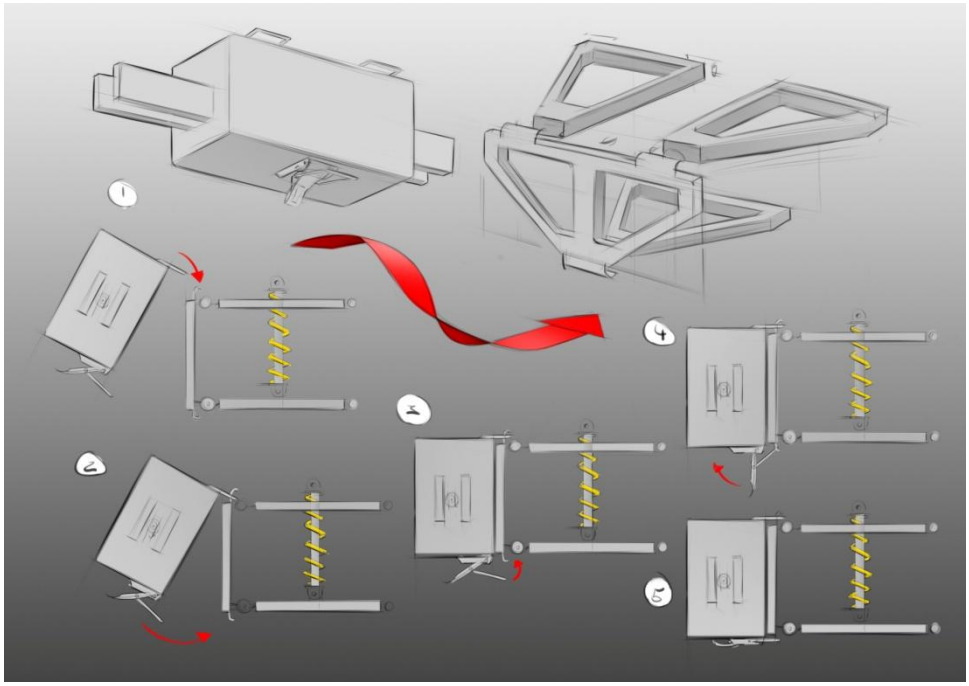


Figure 17 One idea of a quick release for the coupling

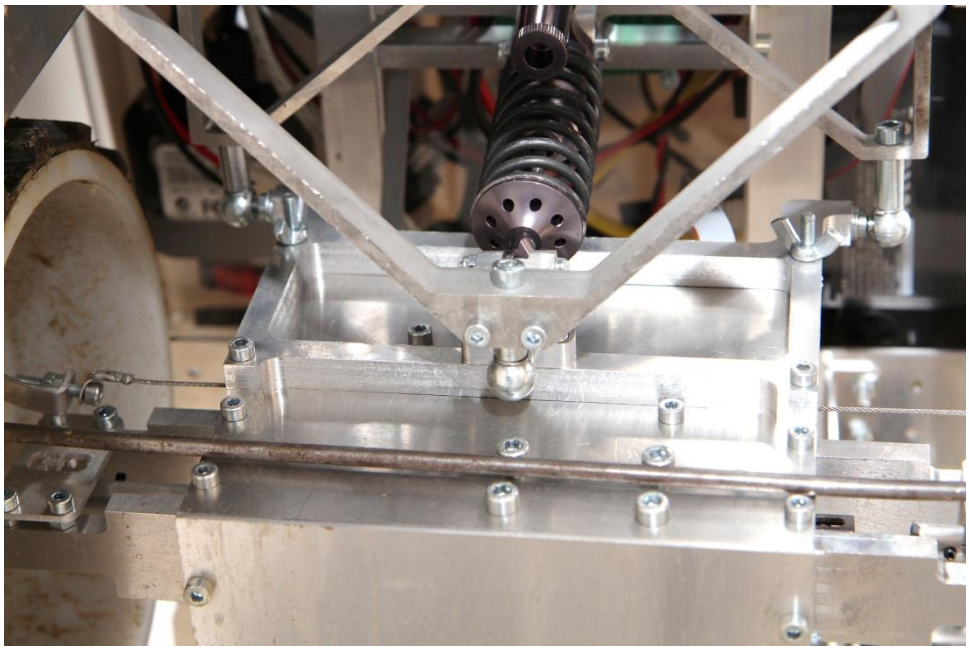


Figure 18 The finished quick coupling

2.7 Top plate

The upper part of the robot is designated for the computers and main electronics. The user interface through which the robot is operated also attaches to the top plate. Top plate was manufactured from a sheet of acrylic, which was bent by the edges to give stiffness. The shape and mounting holes for the components were pre-cut using a laser. A special acrylic bending devices was thereafter used to bend the edges of the sheet. A thin strip of the sheet was heated after which it was bent. Components were either screwed, taped or attached using Velcro tape. Acrylic was chosen because it can be laser-cut and does not conduct electricity which reduces the risk of shortcuts.

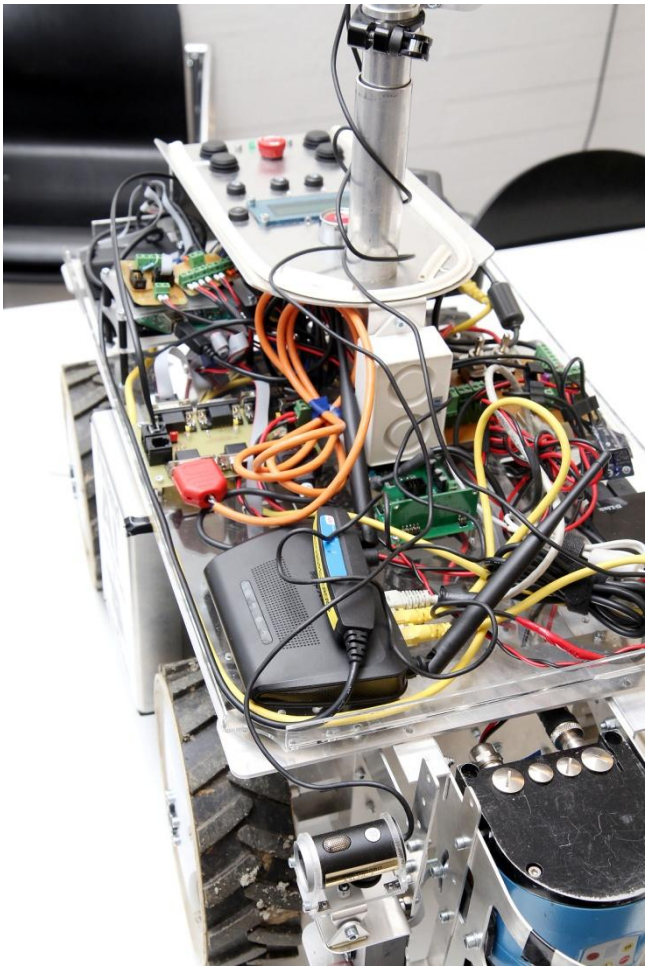


Figure 19 The top plate held all the electronics. The edges were bent to add stiffness

2.8 Cover

The cover was made to protect the robot, mostly the electronics from rain and dirt. It also made the appearance of the robot cleaner and served as a surface for logos and texts.

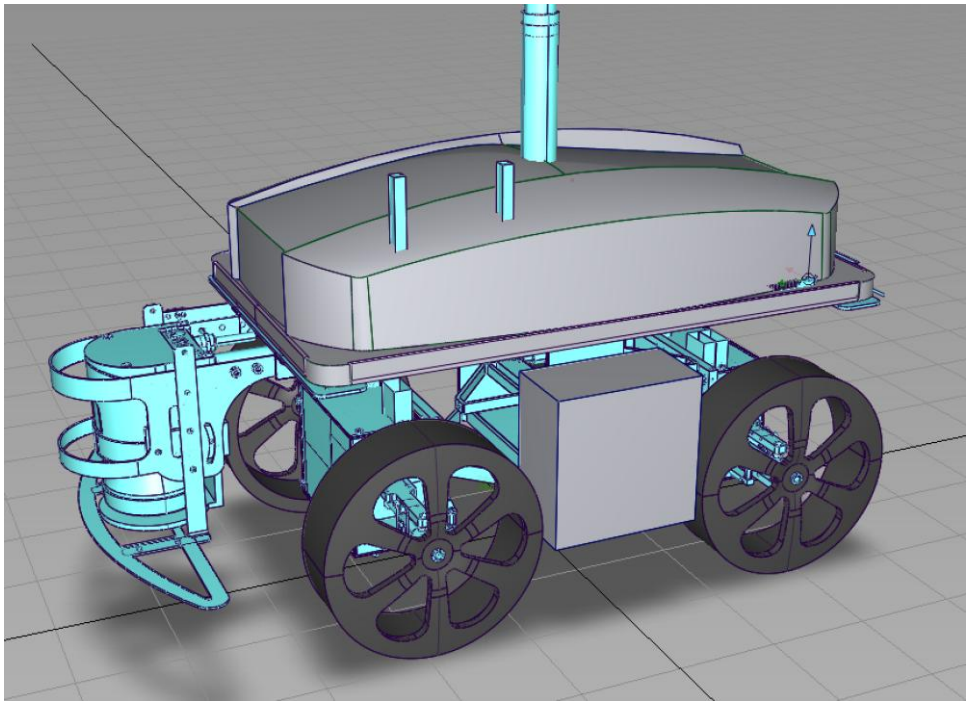


Figure 20 The cover was designed using CAD

The cover was made out of vacuum formed plastic. Therefore we had to make a mold, which must have draft so that it is possible to remove the finished cover from the mold. Other important design aspects were bend radius and how to achieve uniform vacuum. CAD allowed us to precisely design the mold so that it nicely sealed the mounting board and the user interface, but did not rattle and the fit was so tight that it hardly required any mounting screws.

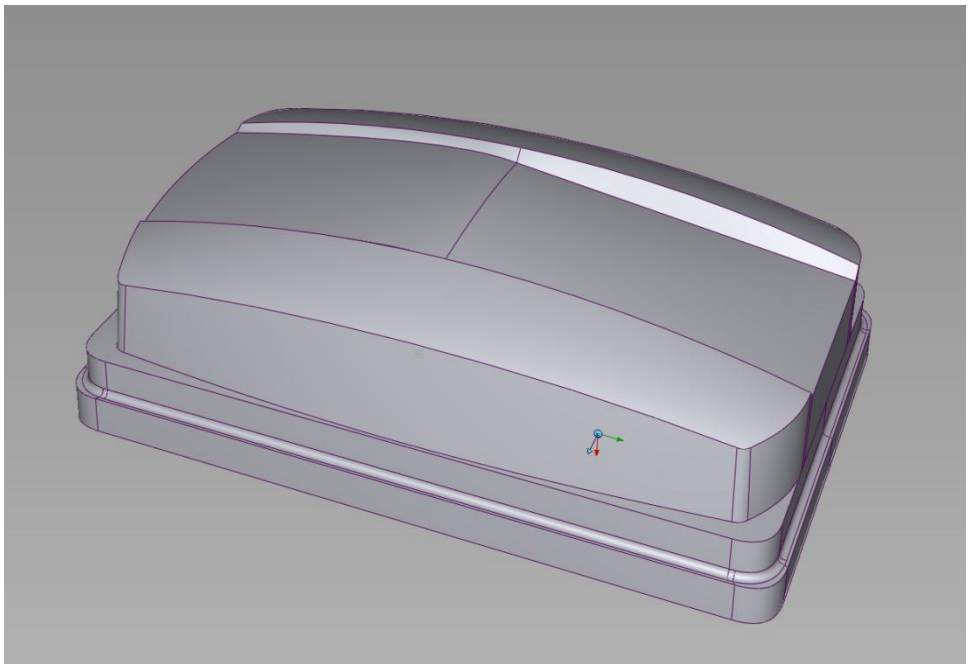


Figure 21 CAD model of the cover-mold

The mold was made from wooden block that was manufactured by gluing thick plywood sheets together. CNC-milling machine with a ballnose milling cutter was used to shape the plywood into a mold. Mold required holes and air channels that distribute the vacuum evenly, these were done by hand after which the surface was sanded to produce an even surface.

Figure 22 The vacuum formed cover

Figure 23 The cover not only served as protection but also as a surface for text and logos

2.9 CAD

The primary tool for designing the mechanics of the robot was Computer Aided Design. For this project, the Creo Elements 3D CAD-software was used.

When the main dimensions of the robot were figured out, the main structure was modelled using a skeleton model. Due to its complexity, an own skeleton model was created for the axle modules. The skeletons were used as reference when modelling the actual parts. This gave us flexibility and allowed all three members to work simultaneously on the same model. When changes were made to the skeletons, parts referenced to these also got updated.

The aim was to create a 100 % complete CAD model. This means a model were all components have been modelled, nuts and bolts included. Surprises in the assembly stage with parts not fitting were eliminated.

The only components that were not completely modelled were the electronics. There was no point in modelling them in detail. Instead, boxes were created to ensure enough space was reserved. Room for cabling was also taken into account in the 3D model.

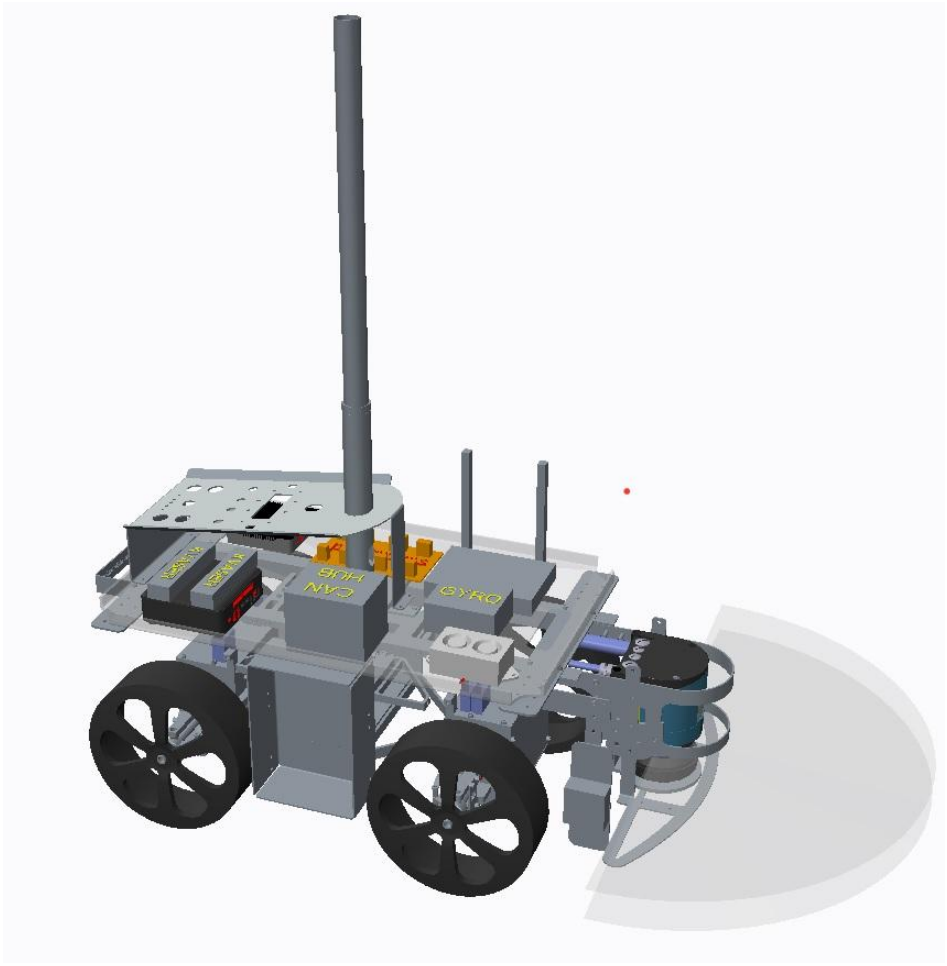


Figure 24 CAD model

2.10 Manufacturing and assembly

Design, machining and assembly was done at the university's workshop. The mechanics were designed so that they could be manufactured with the available machinery. The material for the structures was chosen to be aluminium because of it machining friendly properties, lightness and strength. Plates of various thicknesses were mostly used. Most parts were manufactured with manual machinery, but some parts were machined with

CNC which was better suited for more complex shapes and was a great learning experience. The CAD model that was created was directly used when creating the CNC-machined parts.

A lot of sheet metal parts were also used. These were manufactured by one of the sponsors, Laserle Oy. They were cut based on the drawings we had made from the 3D models, bends were then done at the university's workshop.

The assembly was also something that was taken into account when designing the robot. To make life easier, welding was ruled out as an assembly method. The main assembly method was screws and bolts.

The hinge joints were made from bolts and spacers. Spacers were purchased and the design was adjusted according to their dimensions. The ball joints were also bought as off-the-shelf components.

Figure 25 Some parts were manufactured using a CNC-mill

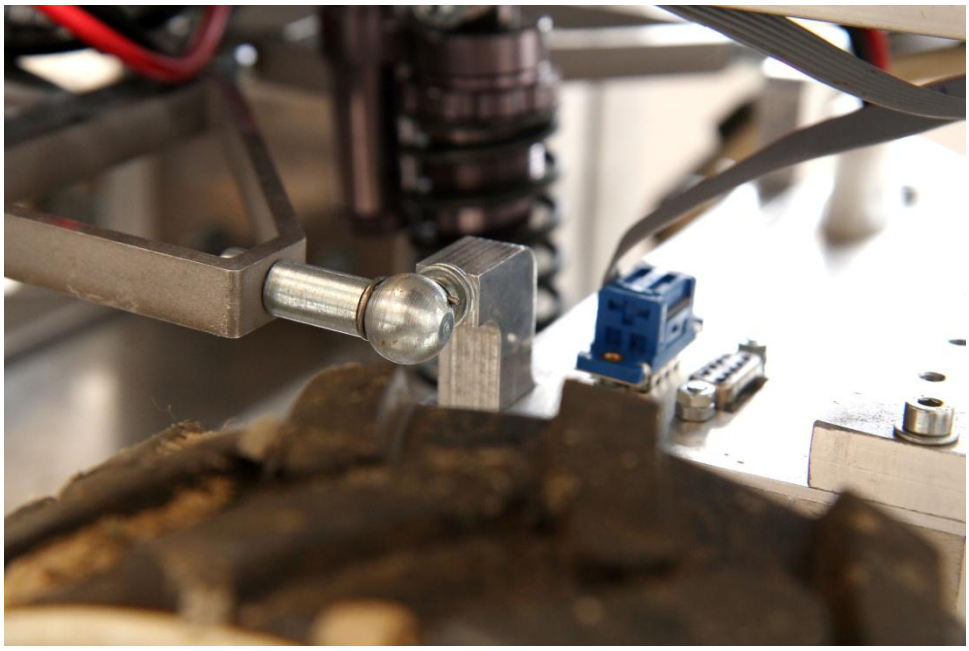


Figure 26 The construction was assembled using bolts, nuts and ready made joints

3 Hardware

We utilized the same microcontroller modules as the previous field robot by the same universities. These Crumb AT90CAN128 (www.chip45.com) modules meant less soldering and modular design makes it easy to replace the microcontroller after a short circuit or over voltage damage (Vepsäläinen, J. et al.). CodevisionAVR was used to produce the embedded C code.

3.1 Power Electrics

Robot is powered by three RC LiPo batteries. One provides energy for drive motors and steering servos, other two are dedicated to electronics. Batteries are not connected in parallel which makes battery management easier, because battery voltages and capacities don't have to match. Three separate circuits also decreases electromagnetic interference coming from the brushed DC drive motors. Communication with drive motor circuit is galvanically isolated from the other two circuits, but batteries share a

common ground which prevents a build up of static electricity and prevents problems related to different ground potentials.

The two circuits that power the electronics are divided into smaller circuits that are all connected to a distribution board that has fuses and switches that enable resetting or disconnecting certain circuits. Single distribution board makes wiring and changing the fuses easier.

LiPo batteries are unprotected and must be monitored to prevent over-discharge. Each battery has its own protection circuit that has an Atmel Atmega88, main fuse and a relay. Microcontroller measures the battery voltage using the built-in ADC, when voltage drops below a safe threshold microcontroller uses the relay to disconnect the battery from the circuit and prevents over-discharging the battery.

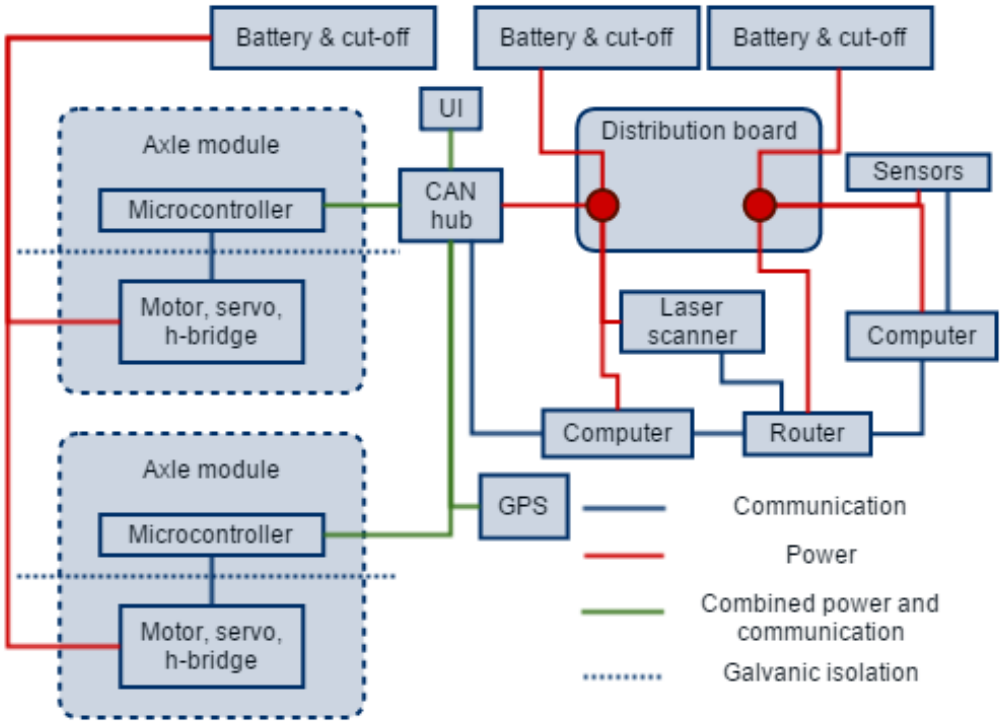


Figure 27 Power electronics of the robot.

3.2 Axle Module

Axle module houses a custom designed H-bridge and another PCB (printed circuit board) for control electronics. H-bridge is constructed using N-channel mosfets that are mounted to the axle modules casing in order to provide cooling. Stmicroelectronics L9904 (http://www.st.com) controls the MOSFETs and Allegro microsystems ACS710 (www.allegromicro.com) measures motor current, mainly to monitor for excessive current that would damage the drive motor. Control electronics board has AT90CAN microcontroller that receives commands and sends information via CAN bus. Microcontroller commands the H-bridge and communicates with the steering servo, in addition it reads the motor speed encoder (Cui inc AMT10, http://www.cui.com) and monitors motor temperature with NTC thermistor. Control electronics are galvanically isolated from the h-bridge and steering servo using isolated transceiver and digital isolator.

Drive motor speed is controlled with a closed loop discrete PI controller running on the microcontroller based on the encoder readings.

3.3 Physical UI

Robot is equipped with 12 buttons, 7 LEDs and a 16x2 alphabet LCD. LEDs indicate whether robot is about to turn left or right, reverse and most importantly signal if the robot believes it has reached the end of the row and switches the navigation algorithm. The buttons were provided by AGCO/Valtra company, the buttons are the same used in tractor cabin, so they were suitable for the environment. Low battery and power on lights displays information from the battery protection circuit. LCD is used to show battery voltages and display commanded turn sequence. The finished UI is displayed in Figure 28.

Buttons are used to manipulate the command sequence i.e. if the robot makes a mistake it can be corrected and will not ruin the entire competition run. Emergency stop button physically cuts the power to the drive motors and a press of the start button is required

at the start and after emergency stop has been applied. Momentary power buttons are used to energize battery protection circuits relay coils, after which microcontroller keeps the relay latched. This arrangement completely eliminates any standby current and decreases voltage drops in the UI wiring.

Weatherproofing is achieved by gluing a laser-cut acrylic on top of the LCD, by using IP rated buttons and window seals together with slanted design that allows water to flow away from the top plate and its sensitive electronics.

Figure 28 The physical UI attached to the robot.

3.4 Computers and Sensors

3.4.1 Navigation PC

ICOP's eBox-3350MX Simulink C code and had written C# code. Visual Studio 2008 allowed us to remotely deploy the code to the target robot directly from the IDE.

3.4.2 Machine Vision PC

The Intel NUC with Core i5 was deployed to allow higher processing power needed to process the machine vision and laser scanner algorithms. The results of the positioning algorithms were transferred from NUC to eBox through an Ethernet connection.

3.4.3 Laser scanner

Robot was mainly guided by SICK LMS100 (www.sick.com) 2D laser scanner.

3.4.4 RangePack

These modules have been a traditional component in Aalto University's field robots for many years. Sensor module consists of a Sharp GP2Y0A41SK0F infrared ranger and SRF05 (www.robot-electronics.co.uk) ultrasonic sensor for longer distances (Vepsäläinen, J. et al.). Atmega88 processes data from the sensors and communicates information to the computer via RS485 bus. Robot has a RangePack module on each corner, which are all connected to a single bus that also provides power for the RangePacks. Computer is connected to the bus via USB adapter and polls the sensor data from the RangePacks.

3.4.5 Webcams

Robot has two webcams for plant detection. Cameras are attached to a mast in order to provide the best possible view, they are mounted at the middle of the mast looking down towards the sides of the robot. These were used to detect dried plants in task three. One camera was to be mounted on top of the mast to provide computer vision for navigation, but this was discarded because detecting green corn from grass background was considered to be unreliable. Navigation camera would have been mounted to a servo that would allow 180 degrees of rotation and would enable the camera to look backwards and aid in reversing.

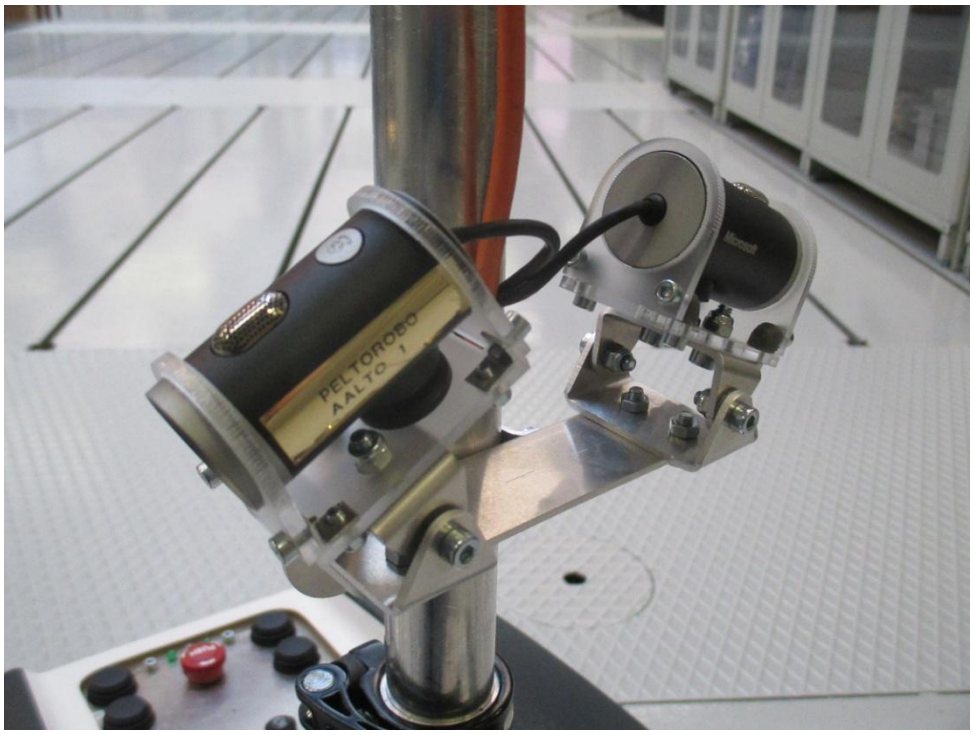


Figure 29 Cameras attached to the middle of the mast.

3.4.6 Gyro

Robot is equipped with a MEMS gyro from Murata Electronics. This sensor was especially useful at row end turns (Piirainen, P. et al.).

3.4.7 GPS

GPS 19x marine grade GPS unit from Garmin was selected because of its update frequency (10Hz) and NMEA2000 communication protocol, which is compatible with CAN bus that was used as the main communication bus. Intention was to use the GPS for gathering bearing and velocity information.

4 Software

4.1 Toolchain & architecture

The core of the toolchain was a Simulink model in Matlab. Functionality of the robot was mostly implemented with Simulink models that were then automatically generated into C code. This code was then wrapped with C# in Visual Studio. The compiled code was then run in two separate computers, Intel NUC with Windows 7 OS and eBox with Windows Embedded CE 6.0.

Simulink was chosen because it provides a graphical presentation of the program structure, which provides a better overview of the functionality of the code. Support for more complicated mathematics was also seen as an asset. To improve the clarity of the complete models for NUC and eBox parts of the model were packaged into their own library blocks. These library blocks included standard Simulink blocks, state machines, embedded Matlab code and other library blocks. Another benefit of the library blocks was that some structures were needed several times with different input parameters. Library blocks provided an easy way to multiply the used solution, as well as editing different instances of the block together or separately.

To further improve the visual appearance of the model, most inputs and outputs at higher levels were connected with buses to reduce the amount of connectors. Different inputs and outputs were then separated from the buses inside the library blocks. Unfortunately saving the buses was not entirely simple process. The buses had to be loaded into workspace and then saved as a separate file. When starting the model, the file had to be loaded with a separate init file that also included other parameters needed in the model.

To improve the debugging properties of the model, all blocks, inputs and outputs were adjusted to avoid the default inherited behaviour. This meant setting the sample time, data type and port dimensions to fixed values. Additional tool used for debugging in

Simulink was the use of scopes in the finished model and connecting it remotely to the software running in the robot.

When the highest level of the Matlab model was ready for testing and built without errors it was compiled into C code from a drop-down menu in Simulink. This also required some settings in configuration in Simulink under the Code Generation tab. Also, continuous blocks would not work in the compiled code, so the discrete blocks had to be used. The C code was automatically generated into the active Matlab workspace. To use the generated code, a dynamic link library (.dll) had to be created. On the eBox the used libraries were for C++, whereas on the NUC the .dll had to be for the C#. Microsoft Visual Studio 2008 was used for creating the .dll files and the rest of the code.

For the C# files a new C# class library project had to be created. After that the generated code could be used with static methods. C++ .dll files for Windows CE were more complicated to create. This required making a Visual C++ project for smart device, then choosing eBox 3350MX as the device and choosing .dll as the file type. The created .dll could then be added to the project as a link. To use the .dll in the project, *"using System.Runtime.InteropServices"* reference had to be added to the beginning of the file where the .dll was used. Also, a Visual Studio command prompt had to be opened, and then navigate in it to the file location of the .dll. In the command prompt *"dumpbin /exports <nameofdll>"* had to be written. This would result in a list of entry points for functions that returned values. The entry points were needed to surpass the security measures in Windows when passing data from C++ to C#. The library functions could then be used as methods in the project with a DllImport method that utilized the entry point found with Visual Studio command prompt.

Another interesting feature with the .dll files was how to use the same source file in both C# and C++ .dll files. This was simply done by creating three projects. One was only for the source code and the others were for C# and C++ .dll. Then the source file project was simply added to both .dll projects as a link.

NUC needed a C# Windows 32-bit console application as a main file. A Windows form application was also created as a graphical interface for using the robot remotely, although it was never properly finished. The existing C# .dll that included the NUC Simulink model was added to the console application project. Then the desired functionality was coded and the program compiled. A remote desktop viewer was used to initialize the compiled .exe file in the target PC (NUC) once it was moved to the computer.

The Ebox needed somewhat different approach. The created project for the main program had to be a smart device project for Windows CE and a device application. Linking a .dll with the eBox Simulink model was done similarly as with the Win32 project. Once the project was built, it had to be deployed to eBox. For this the IP adress of the eBox had to be defined to the VisualStudio settings in a device tools and there the eBox3350MX as the device. For deploying the software the computer also had to be connected to the same network as the eBox. Once this was done the program could be deployed by selecting the eBox3350MX as the target. Running the program in the eBox required a Remote Display program. With the Remote Display program the eBox could be used via normal computer where the Remote Display program would act as a screen of the eBox. Then the deployed project could be used by simply running the .exe program in the eBox.

The most difficult challenge with the toolchain and the architecture was probably the unique structure, which meant that it was hard to find help from the internet and the methods used were new for us as well. Debugging the software was also difficult as sometimes it was unclear whether the bug was in the Simulink model, the code or the Visual Studio's settings. Numerous parameters and settings that had to be adjusted for each file and computer were also difficult to remember despite writing down almost every setting that was adjusted. There were also connection issues between the target hardware and the computers used in the programming. As a result of this a large portion of the time used by the coders was spent on solving problems with the toolchain.

4.2 Communication

Last year's good experiences had an influence when deciding what communication technologies to use for GroundBreaker. Therefore, the communication was implemented pretty much the same way as previously. There are four different communication methods: A CAN bus, a UDP-based Ethernet, the USB ports and a single connection using a RS-485 bus.

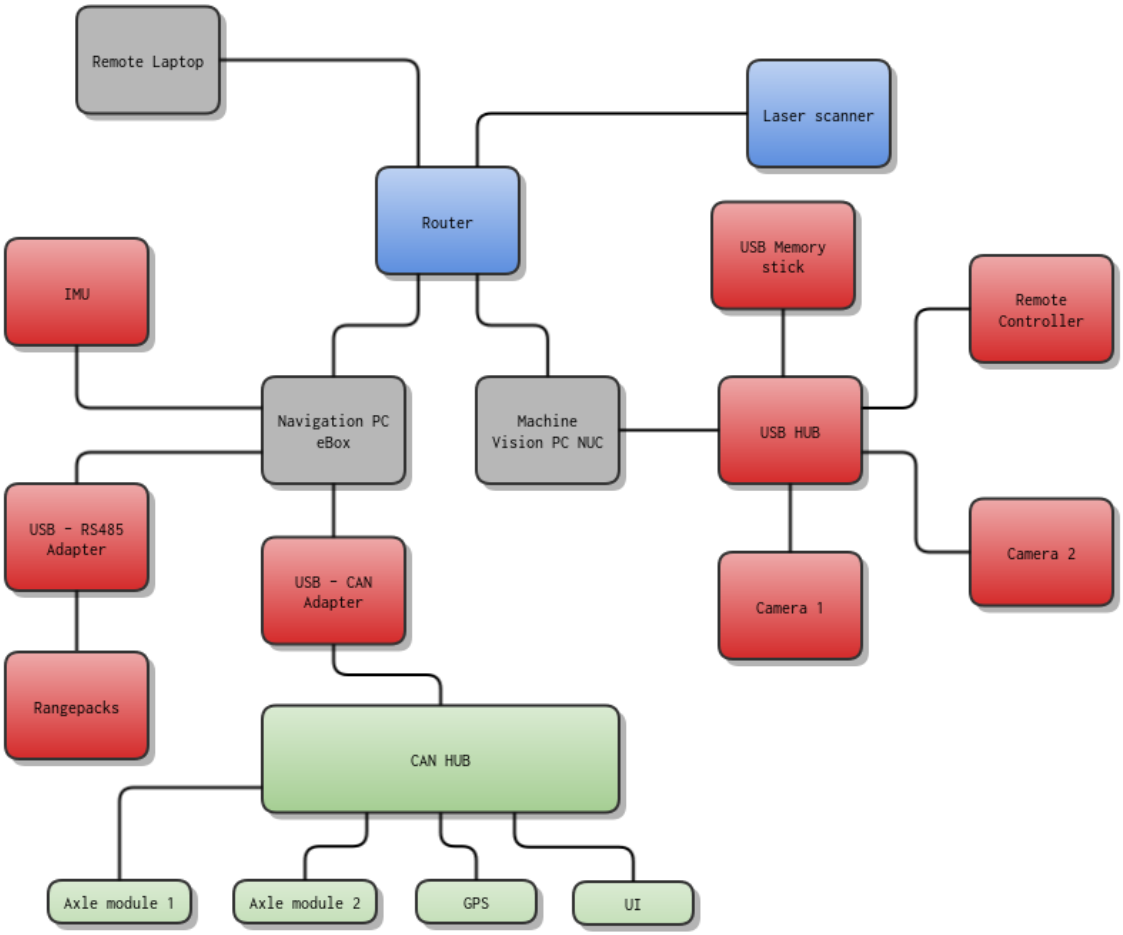


Figure 30 The communication diagram of the robot

4.3 CAN bus

Every axle module - the front, the rear and the trailer module when the trailer is connected -, the physical interface, the GPS receiver and the onboard eBox PC are connected to the CAN bus. They all act as a independent node in the network. There are many reasons behind the use of CAN. Firstly, the technology is commonly used in many existing commercial applications, and thus it offered valuable learning experience to the team. CAN is also quite robust communication method - if one of the nodes suddenly fails, the other parts of the network will still remain functional. This makes debugging easier and also adds a little robustness to the robot.

A Kvaser Leaf Light HS CAN bus analyser cable with CANTrace and CANKing programs were used for analysing and debugging. The CAN bus uses CAN 2.0b J1939 standard with the extended 29-bit identifier. Some simplifications were made, e.g. no address claims or request address claims are made, the independent identifiers were manually hard-coded in order to prevent the possible conflicts from the duplicate identifiers.

A custom CAN hub acts as the central node for the CAN bus. Two types of connectors are used: DB-9 and RJ-11. The DB-9 connectors were used for the axle modules, Kvaser modules and GPS. RJ-11 connectors were used for the UI and trailer connection. CAN bus is used to supply a voltage rail (12V) to the connected devices with the exception of Kvaser and trailer connections which don't require power.

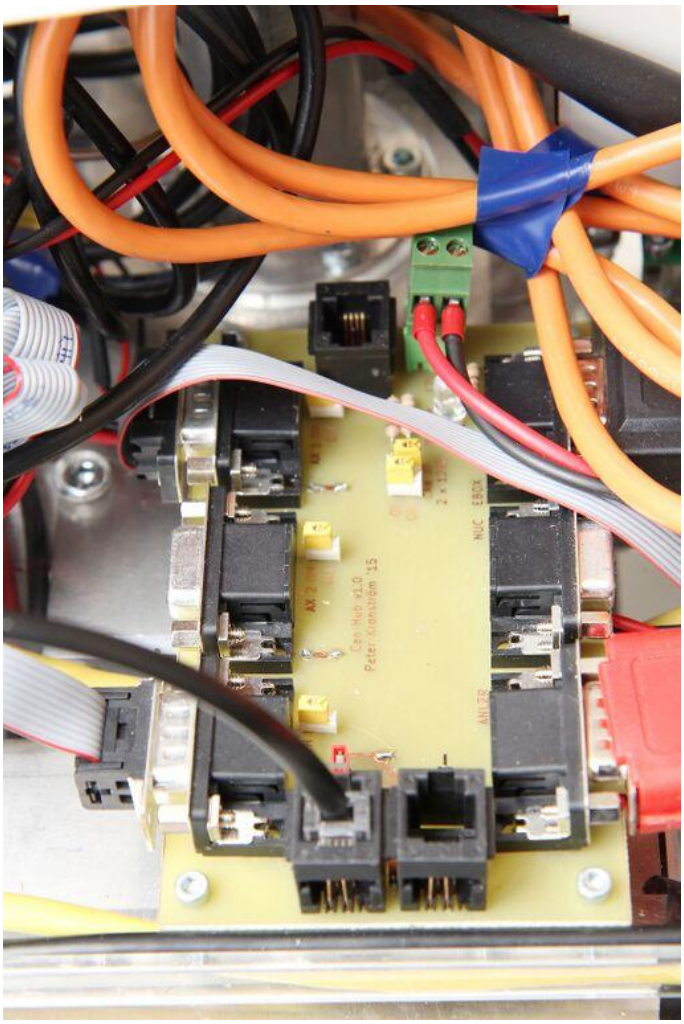


Figure 31 The CAN hub of the robot with some devices attached.

4.4 UDP and Wi-Fi

At first it was discussed whether to use the CAN-hub for communication between the two PCs on board. Due to the limits in the CAN-hub's capacity, it was decided to implement another communication link for the purpose. The two PCs are connected to a Wi-Fi router with the ethernet cables and an UDP-based communication logic was implemented for relaying the messages. UDP was chosen because of its simplicity. Nevertheless, UDP is a good technique for this purpose as the system can survive an occasional lost message.

The Wi-Fi router is also used to connect a laptop to the robot's network. This is necessary for starting and terminating applications running on both of the robot's computers.

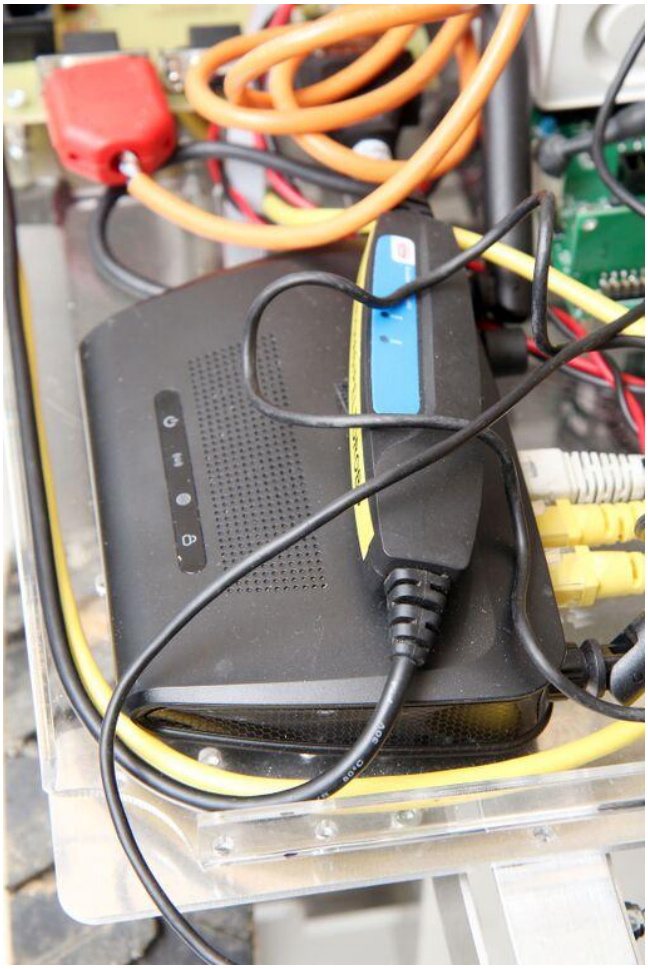


Figure 32 The Ethernet router used in the robot with connections to the laser scanner, eBox and NUC.

4.5 USB and RS-485

There are a wide range of sensors and a manual controller that are not connected to neither communication net discussed above. Every camera, a gyroscope and a Logitech Cordless Rumblepad 2 are connected to the PCs directly by USB. Drivers for them were written using C#. A gyroscope using a specific "Field Robot Protocol" was given to the

project team and it fell to us to write a driver to interpret the messages. An existing C# library was used to interpret messages produced by the Rumblepad. As the PCs don't have sufficient number of available USB ports to support every device, an external USB hub was required.

The RangePacks are connected to a RS-485 bus that is attached to a computer via USB to RS-485 adapter.

4.6 Parameters and Logs

The parameters and logs are an important part of the system's infrastructure – they are essential for adjusting the robot's performance on the go, calibrating the robot for a specific field and collecting feedback data for later performance analysis and debugging. There are many situation specific variables that cannot be hardcoded in advance, and when something goes wrong it is necessary to have some data of the incident if one wants to find the reason. Next both parameters and logging will be discussed in detail.

4.6.1 Parameters

The system has three different parameter levels: P0, P1 and P2. P0 level parameters are constants that are not ever supposed to change. P0 level parameters include for example the physical characteristics of the robot, such as the width, length between the axles and turning radius. They also include some natural constants, like the approximation of pi, and some sensor calibration constants that do not change when the location changes. Virtually every value that could be hard-coded as a magic number should be a P0 level parameter. To change these parameters one must re-compile the entire application. Therefore, the user of the robot is never supposed to want to do this.

The next parameter level, P1, contains calibration parameters designed to improve the performance of the robot. Most of the algorithms contain one or more adjustable variables that can be adjusted to reach a maximum performance on a specific field. P1

level parameters also include some site-level constants, such as height of the plants and the width of the plant row. The pre-competition calibration process includes collecting data from a test field, and adjusting these P1 level parameters using either a simulation or real runs to achieve as good result as possible. Initially these parameters were supposed to be possible to adjust using a laptop user interface, but due to the time and resource constraints this was never accomplished. Instead, the P1 level parameters can be adjusted by manually editing an XML file containing all these parameters between runs. This means, that the system's software has to be rebooted to adjust the P1 parameters.

Finally there are the P2 level competition task specific parameters. These parameters are technically similar to P1 level parameters: they can be changed between different runs by editing a parameter XML file found on both of the onboard computers. P2 parameters focus on competition tasks. For example, there is a P2 parameter to define the current competition task. In the first task the robot doesn't look for obstacles, and in the fourth task it listens to the move orders from the trailer, for example. Also, the move sequence for the task two ordered by the competition staff is given to the robot as a P2 parameter.

4.6.2 Logs

The logs have multiple purposes. After a run they provide the data for analysis and performance improvements. Without the logs it would be almost impossible to tell what went wrong, or how to improve the performance. The log files are also used for the third competition task. The robot is supposed to create a map of the field. This map is built using a custom Matlab function after the run.

A log file is a text file containing information gathered by the system. It includes all the sensor data, such as the ultrasound, infrared, laser scanner and gyroscope measurements, the axle module commands and all the other internal messages such as communication between the two computers and the messages sent by the axle modules.

However, it doesn't contain any analyzed information, for example all the stored sensor measurements are uncalibrated.

The system stores a new row of measurements every 50ms. A single row on the text file contains all the data for one time step. Both of the PCs create their own log file storing their respective information. For example, all the laser data gets stored to the NUC's log file and RangePack measurements get stored to the eBox's log file.

Two common use cases for the log file are on-site system calibration and map making for the task three. In the initial calibration phase one run using manual Rumblepad control is made in the competition field. The log files produced from these runs are used to simulate various algorithms with the Matlab. An algorithm's performance is tested, a parameter change is made and the performance is tested again. This kind of try and try again – approach provided sufficiently good results at the competition site.

The other use case at the competition site was a map making. For the third task the robot is supposed to create an accurate map of the competition field. The team's solution was programmed overnight on the site as there was no time for this beforehand. A Matlab function uses only certain information from the eBox log file – the ultrasound and infrasound measurements, plus the gyroscope and odometry data. It uses this data and a few suppositions to build the map. The suppositions are that the field consists of plant rows which are straight and parallel to each other. The map of the competition field can be seen below.

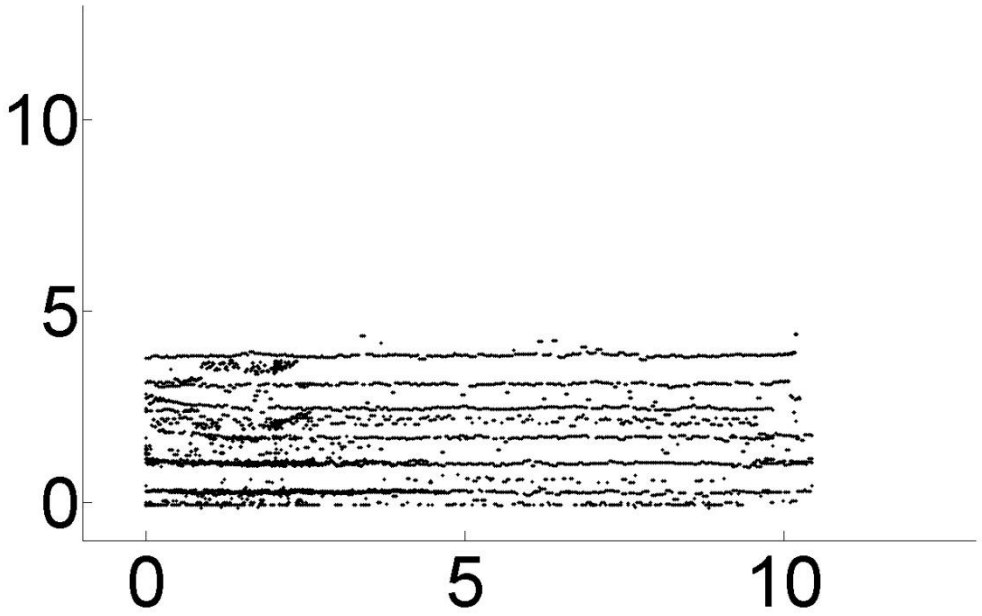


Figure 33 The map created of the competition field for Task three

4.7 Algorithms

To autonomously drive on a maize field, various algorithms are required. The robot needs to localize itself and correct its pose on the maize field. In principle the robot is a complex line following robot. This means that the navigation of the robot is completely based on its position relative to the maize plants and the ordered turning sequence. No global positioning is considered in the navigation algorithms.

The algorithm design of the robot worked from simple to complex. Therefore the algorithms are classified based on that. The algorithms complexity levels were expressed with letters. The A-algorithms are the simplest algorithms and the complexity increases with each letter in alphabetical order. The robot was designed so that the different algorithm versions were interchangeable with each other. The various algorithms used in the robot are divided in to different subtitles and presented here.

4.8 Positioning Algorithms

The positioning algorithms uses different sensors for detecting the maize field the robot is navigating. The earlier algorithms used mostly just the ultrasound sensors, while the latter algorithms used the 270 degree laser scanner mounted in front of the robot. Because the maize field looks quite different when the robot is between the field rows and when at the row ends, the robot uses different positioning algorithms when on the row ends. Therefore positioning on the field rows and row ends are divided in to different subtitles.

4.8.1 Positioning on Field Rows

On the field rows the robot is surround by the perpendicular rows of plants. The approximate distance between the plant rows can be taken as a parameter in to the positioning algorithms because it is nearly a constant on a particular maize field and can be measured manually. The plant rows can have gaps, turns and other irregularities, therefore some redundancy is required from the positioning algorithm. This requirement for a redundancy rules out many of the trivial solutions for detecting the distance and angle to the centre of the field row.

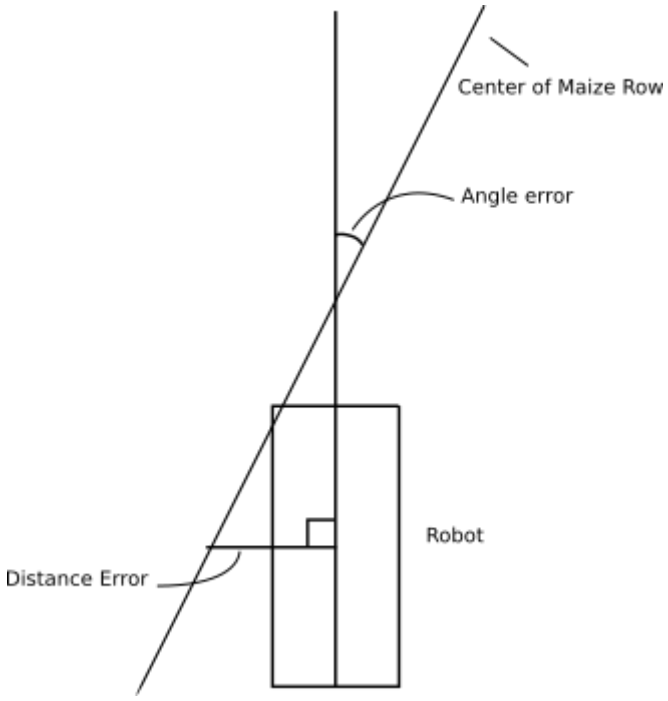


Figure 34 An illustration of the error terms used for the row positioning

4.8.2 Row Positioning A

The A algorithm for row positioning was based on the readings from the RangePacks. The algorithm achieves redundancy by including past measurements in combination with the current ones. Each range measurement is considered as a point corresponding to a maize plant. The positions of the old measurement points relative to the new measurement points are calculated using the odometry from the robots wheels and steering servos. The algorithm can be divided in to 6 steps:

1. Transform all the old measurement points to the current robot coordinates based on the odometry.
2. Calculate the new measurement points from the distance measurements.
3. Add new measurement points to the measurement point buffer.
4. Transform measurement points of the plants to corresponding row centre points by using the row width.
5. Use points corresponding to the row centre line for the linear least square mean error linear regression.

6. Use the calculated regression straight parameters to calculate the angle and distance from the row centre line.

4.8.3 Row Positioning B

The B algorithm for row positioning employed the laser scanner data for better positioning information. Because the scanner was mounted on front of the robot, only the field in front of the robot is visible in the scanner data. However, the larger data sets provided by the laser scanner allowed a more robust and history independent algorithms to be used. The B algorithm deployed for the row positioning is an extension of the A algorithm for the laser scanner usage.

To use the same linear regression algorithm for a laser scanner, the multiple plant rows showing up in the laser scanner must be combined into a single line. This is done by first rotating the laser data so that the field rows are in the direction of the X-axis. After that we take the modulo of the data along the Y-axis by using the row width parameter. The algorithm can be summed up in the following steps:

1. Filter data based on the distance and position.
2. Rotate the point data with the constant increments.
3. For each rotation of data:
    a. Create histogram along the Y-axis.
    b. Calculate variance of the histogram.
4. Choose the rotation with the highest variance. This results in vertical lines because vertical lines create high spikes in the Y-axis aligned histogram.
5. Calculate modulo of the data along the Y-axis.
6. Use the linear least square mean error regression for finding the position and angle of the robot relative to the field rows.

4.8.4 Row Positioning C

The B algorithm worked fine on straight field rows. However on curved maize rows taking the modulo of the field point data caused discontinuities in the data. The

discontinuities created two different groups of data points that dominated the line fitting. The goal for C algorithm was to get rid of these discontinuities.

The C algorithm borrows ideas from a colour histogram tracking algorithm used in the computer vision. The idea is to first write a function that returns the error of the current model when fitted to the data. In this case this model is a finite group of parallel lines. The error is calculated more efficiently by taking the y-axis aligned distance to the nearest straight for each point and squaring it. Therefore this uses squared error but only in the y-direction, avoiding the performance hit of calculating the equations for the distance between a point and a line.

Because the model we are trying to fit is a finite group of parallel lines it can be characterized by four parameters:

1. the coefficient of the first line,
2. the constant of the first line,
3. the number of the lines and
4. the distance between the lines.

Out of these only the parameters 1 and 2 are unknown at the execution time. Parameters 3 and 4 can be considered as the tuning parameters for the algorithm. For each scan the algorithm takes the best model parameters found for the previous scan and tries to search the parameter space around that starting guess to find the best parameters for the new data. The algorithm can be divided to steps as follows:

1. Filter data based on the distance and position.
2. As initial guess, use the parameters that were the best fit for the previous dataset.
3. For a set number of iteration, repeat the following:
    a. Take the current best fitting parameters and calculate the error.
    b. Calculate error with the following parameter pairs:

Parameter 1	**Parameter 2**
best_parameter_1 + search_step_size	best_parameter_2

best_parameter_1 - search_step_size	best_parameter_2
best_parameter_1	best_parameter_2 + search_step_size
best_parameter_1	best_parameter_2 - search_step_size

c. Compare the errors to find the new best fitting parameters.
d. If the old parameters had smallest error, reduce the search step size by a half.

4. Calculate the robot position from the found parameters.

4.8.5 Positioning on the Edges of the Field

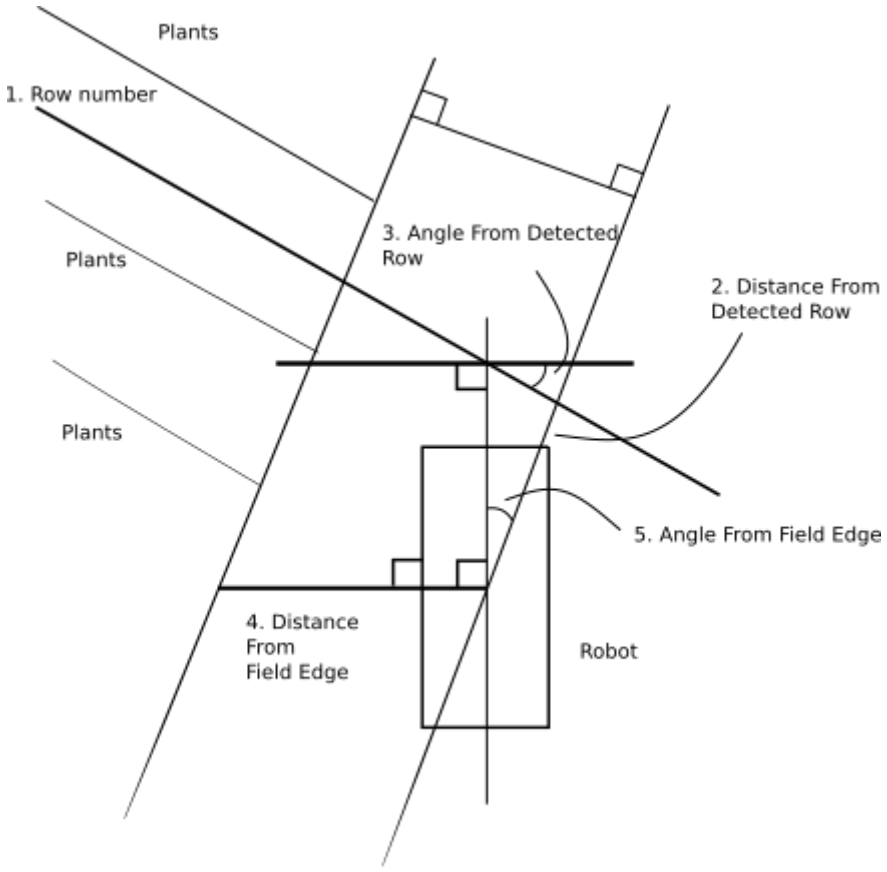


Figure 35 An illustration of the error terms used when driving at the edge of the maize field.

The pose when driving on row ends is expressed with five variables:

1. Current detected row number, where 0 is the row robot drove last. The positive rows correspond to the rows on the left from a last driven row and the negative rows to the right.
2. Distance to the current detected row along the direction of the row edge.
3. Angle of the current detected row from the robot y-axis (y-axis is to the left).
4. Distance to the field edge along the robot y-axis.
5. Angle of the field edge.

Therefore, in addition to detecting the field rows, the row end positioning algorithm has to worry about how far are we from the edge of the field and which direction does the edge of the field go. All the positioning algorithms used at the edges of the field are based on the laser scanner data, because no other sensor can detect the maize plants reliably. They use the row positioning algorithm C for detecting the parallel plant rows. However additional processing is required for the detecting parameters 1, 4 and 5.

When the robot drives at the edge of the maize field, the maize plants appear on the view of the laser scanner row by row disappearing after having been passed by the robot. By filtering the data based on position, the maximum amount of plant rows visible to the robot in a single point cloud can be fixed. By fixing the amount of parallel lines we try fit to be the same as the number of maximally visible lines in the filtered area, we can make the distance to the detected row jump only when one row disappears and another appears. These jumps can be used for counting the rows passed by the robot and to determine the variable 1. This way of determining the row number was used in all of the positioning algorithms.

4.8.6 Row End Positioning A

In row positioning A parameters 4 and 5 were detected by rotating the point cloud a few degrees at a time, making a histogram and comparing the variances of the histograms similarly to row positioning B. In this case the histogram with the highest variance should be the one that has the clearest open path in the middle and the plants on the

sides. After the correct direction has been determined, the histogram made from that direction is used to find the edge of the field. The edge of the field should have the steepest slope in the histogram.

Algorithm step by step:

1. Filter based on the position of the data points.
2. Rotate the data cloud with the fixed increments.
3. Calculate the histogram in the direction of the y-axis (y-axis points to the left from the robot).
4. Calculate the variances for all histograms and choose the one with highest variance. The angle corresponding to highest variance on the histogram is the direction of the edge of the maize field.
5. Take the histogram with the highest variance and find the steepest slope by calculating the differential of the histogram.
6. Calculate the distance to row edge by using the position of steepest slope in the histogram.
7. Use the Row Positioning 3 algorithm for detecting the maize rows.
8. Calculate the distance and angle to detected row from the parameters of the straight returned by the Row Positioning 3.
9. Increment the row number if the distance to detected row is changed by more than a half of the plant row width.

4.8.7 Row End Positioning B

Row end positioning A works but produces extremely noisy data for variables 4 and 5. Later, when designing the control algorithm for driving in a row end it was seen that the parameter 5 is not really needed. Therefore, this algorithm variable 5 is assigned as a constant value and variable 4 is found with a simpler and less noisy algorithm:

1. Filter based on position of the data points.
2. Create a histogram of the data in y-direction and find the bin with highest number of the data points.
3. Starting from the centre of the robot. Select the first bin that has more than 30% of the number of points in the highest bin. The distance of this bin from the

centre of the robot is the distance between the robot and the edge of the maize field.

4. Use the Row Positioning 3 algorithm for detecting the maize rows.
5. Calculate the distance and angle to detect row from the parameters of the straight returned by the Row Positioning 3.
6. Increment the row number if the distance to detected row changed by more than a half of the plant row width.

4.9 Control

The control algorithms of the robot steer the axle modules and adjust the driving speed to keep the robot in its desired path. For driving between the maize rows, it is enough to just keep the robot centered on the drive path while for the row end driving the distance to the goal row has to also be considered. However, the main control algorithms are the same for both cases.

4.9.1 PID Control

The first idea for driving the robot was to control both axle modules with separate PID controllers. By using the robots y-directional error from the desired position and the robots angle error, the y-directional error for the axle modules are calculated. The error is fed in to the PID controllers that produce a control signal for front and rear axle module steering servos.

The control algorithm works, but easily results in the robot driving partially sideways. The PID algorithm does not provide a speed control and cannot slow down before turning to a row. Therefore the PID algorithm was deployed only for driving on a field row.

1. Calculate the axle module position in relation to a goal line.
2. Calculate the PID gain separately for both axle modules.

4.9.2 LQG Control

The LQG (linear-quadratic-Gaussian) controller is an optimal controller based on a physical model of the robot. It theoretically always produces the best possible control for a given physical model. An accurate physical model is therefore essential when using LQG control. The algorithm was used for driving on the field rows and row ends.

The robot was modelled with three states:

1. The X-coordinate of robot
2. The Y-coordinate of robot
3. The heading of the robot

The control vector of the axle modules has to be converted to not include any angles and to operate only on the speed components. The converted control vector has the following terms:

1. Front module X-directional speed in robot coordinates.
2. Front module Y-directional speed in robot coordinates.
3. Rear module X-directional speed in robot coordinates.
4. Rear module Y-directional speed in robot coordinates.

These terms are then converted back to speed and steer values usable in the axle modules. This simplifies the control algorithm and makes the behaviour much better by removing some of the problematic trigonometric functions from the model.

The LQG algorithm was used with a state estimator that estimated the global pose of the robot. However we did not have enough time to use the results of the state estimator for anything or test its accuracy. The control algorithm can be divided into multiple parts as follows:

1. Calculate the desired goal point in the robot coordinates.
2. Convert the odometry measurements to a control vector form presented above and use this as the previous applied control vector.
3. Linearize the model and calculate the new model matrices.

4. Calculate a state estimate using the state estimator.
5. Calculate a desired goal point in the world coordinates.
6. Use LQG controller with the model matrices and the estimated state and goal state.
7. Convert the control vector to the axle module speeds and steering values.

4.10 Row End Detection

Row end detection algorithms were used for detecting the end of the maize row. These algorithm were relatively simple and needed to be tuned separately for the different field environments. After tuning they performed sufficiently well. Therefore more complex algorithms were not designed.

4.10.1 Row End Detection A

The first row end detection algorithm used the quality values of RangePack measurements to determine when we have reached the end of the maize row. The quality values of the RangePacks are determined from the distance measurement values. Because the RangePacks were less reliable on a long distance, the long distance measurements result in lower quality values. Therefore if the average quality value over the 20 measurements is low enough, we are at the row end:

1. Add new quality values to the buffer of the old quality values.
2. Calculate the average quality value for the buffer.
3. If the average is lower than a given static threshold we are at the row end.

4.10.2 Row End Detection B

The Row end detection B is also a static thresholding algorithm, but uses the wide area of the laser scanner data instead of the measurement history. If there are less laser data points inside the predefined area than the detection threshold, the robot has reached the end of the row:

1. Calculate the number of the points inside a square.
2. If there are less points than the detection threshold we are at row the end.

4.10.3 Dried Plant Detection

The machine vision was used for detecting the dried up maize plants. The algorithm relayed on detecting the brown colour associated with the dried plants. If enough of that colour showed up in an image, a dried plant was detected.

The colour detection relied on an ECCI "Excessive colour" -transform (Maksimow, T. et al., 2007). The transform transfers the image from a RGB colour space to a new vector space where one of the axles contains a chosen RGB colour. The other two axes are the intensity value of the pixel and the cross product of the intensity and chosen colour. This allows us to easily determine how much any specific pixel in a RGB colour space resembles the chosen colour.

1. Read the RGB image from the web camera.
2. Transform the image with the ECCRI.
3. Transform the ECCRI image to three binary images with a static thresholding.
4. Use an erode and dilate to filter out the stray pixels.
5. Use AND operation to combine the three binary images to a one image.
6. Count the black pixels of the image. If there are enough black pixels in the image a plant is detected.

5. Trailer

Trailer was designed for the Task four, which is the Freestyle task in the competition. The object is to improve or upgrade the existing robot to do something new that is related to the agriculture. The only limitation was that the ability had to be presented in the competition area.

Many different ideas were suggested by the team and the winning idea was to build a trailer which conducts a soil sampling autonomously. The trailer can dig three soil samples and store them in the containers which are on a rotating platform. Trailer can also take a penetrometer measurement and it has the capability to measure the soil conductivity.

The trailer works autonomously and only requires the robot to control the trailers own axle module. The movement speed is requested by the trailer by using an ISO 11783 Class 3 Tractor (ISO 11783-7 2002) pattern based CAN bus message. Trailer was fitted with the same axle module as the robot, but was not equipped with a steering servo because the trailer was considered to be too heavy for the servos.

The trailer was fitted with an UI panel, from which the user can enable and disable the various functions of the trailer. Also some state and debugging info is displayed on the LCD panel in the UI.

5.1 Mechanics

The trailer was made mainly from aluminum. The frame was constructed using 40x40x4 mm and 30x30x3 mm L-profile aluminum. As same as with the main robot, laser cut aluminum was used as much as possible. Most of the aluminum sheet was 2 mm thick, but some 3 mm thick sheet was also used. The penetrometer spike was constructed from a hard alloy drill bit. The digger utilizes many off-the-shelf components. The two motorcycle chain wheels were bought and attached to a laser cut aluminum sheet. A motorcycle chain was also bought for transporting the dirt in to the sample cups. Excavator blades (a large spacer cut into two halves) were welded to the chain.

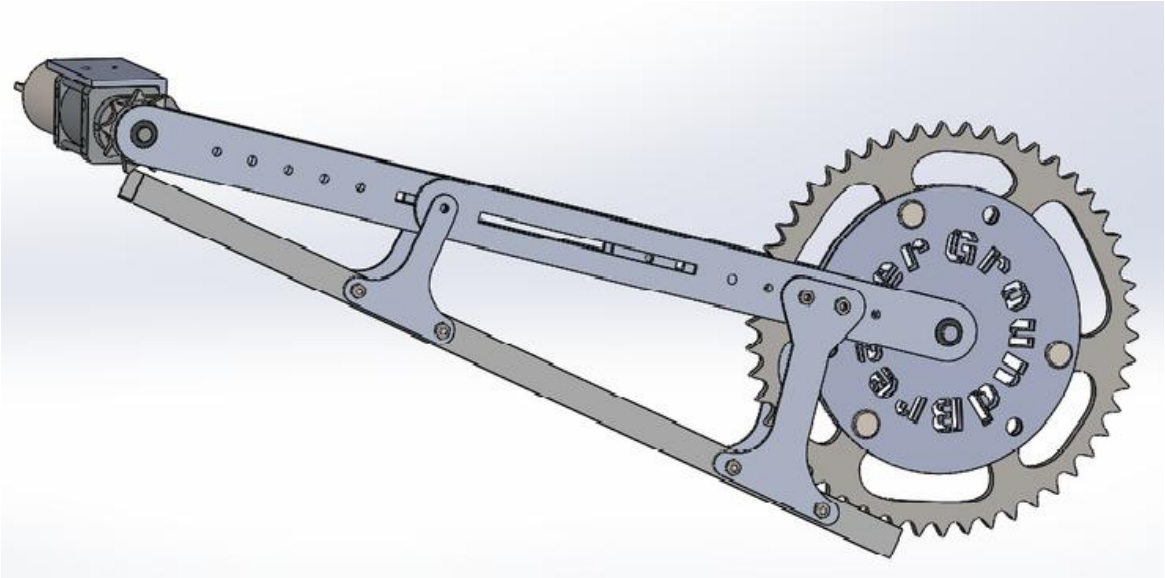


Figure 36 3D-model of the digger (Chain not in picture)

When the robot starts to drive, the main frame is lifted with a scissor lift mechanism that is powered by a linear actuator (Linak LA12). This increases the ground clearance making it easier to design the digger mechanism and allows the penetrometer to go deeper in to the soil. The penetrometer measurements are made using a second linear actuator.

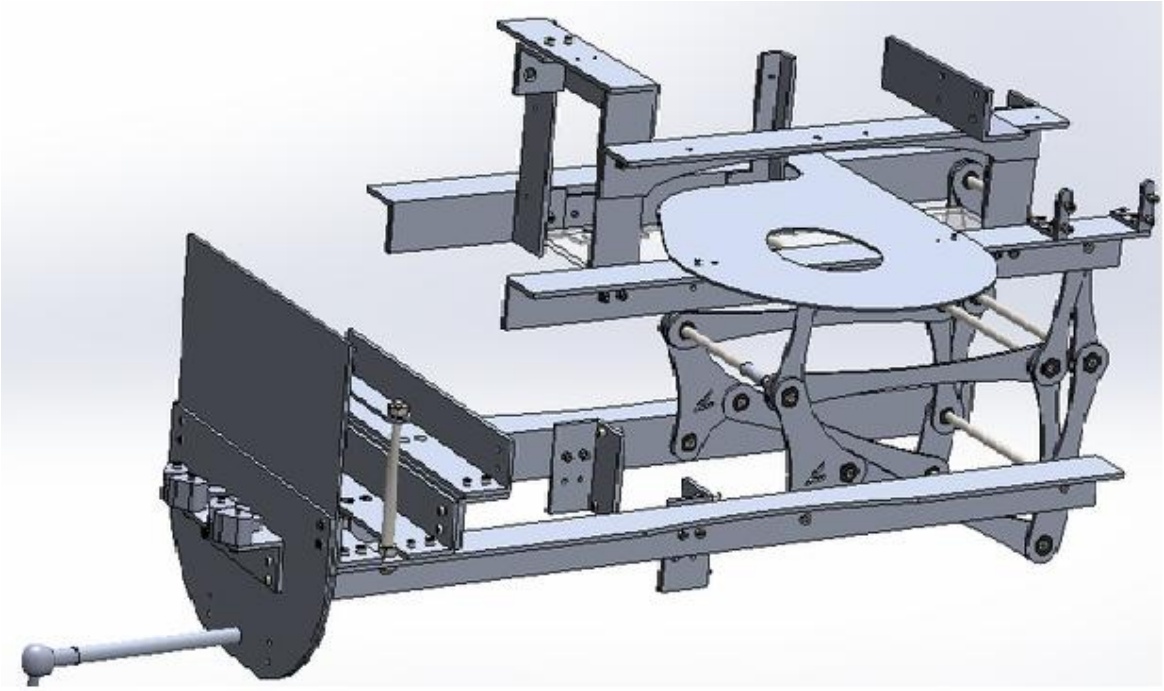


Figure 37 The trailer frame with the lifting mechanism extended.

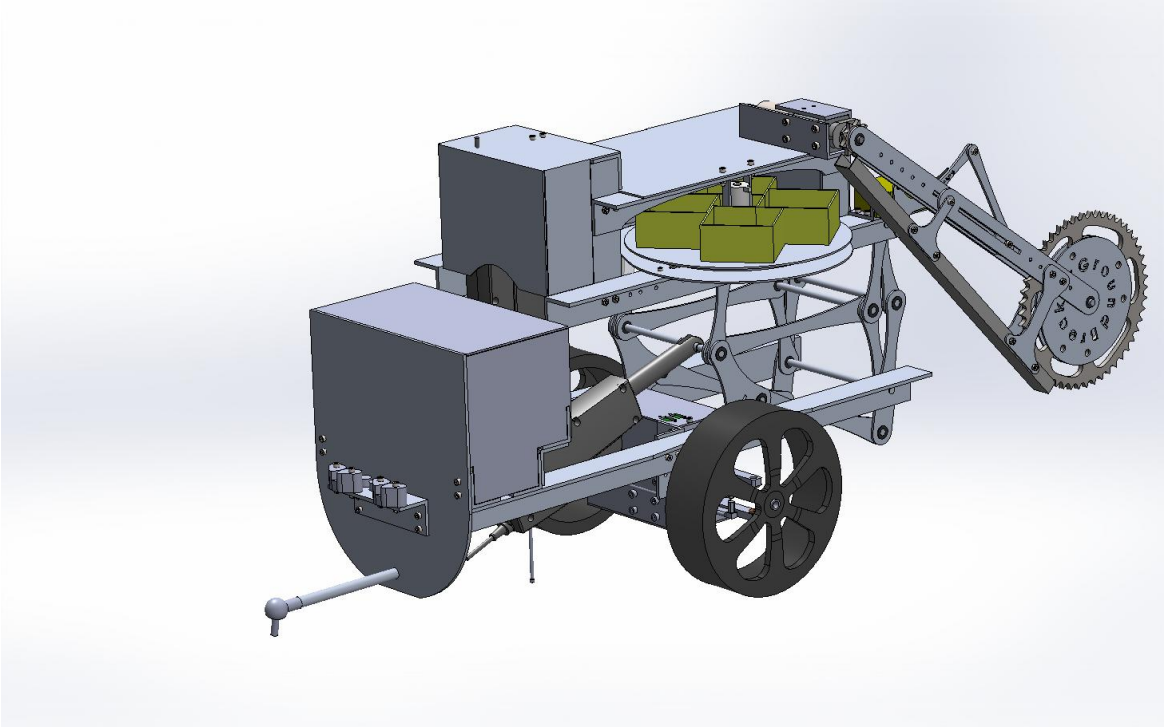


Figure 38 3D model of the trailer

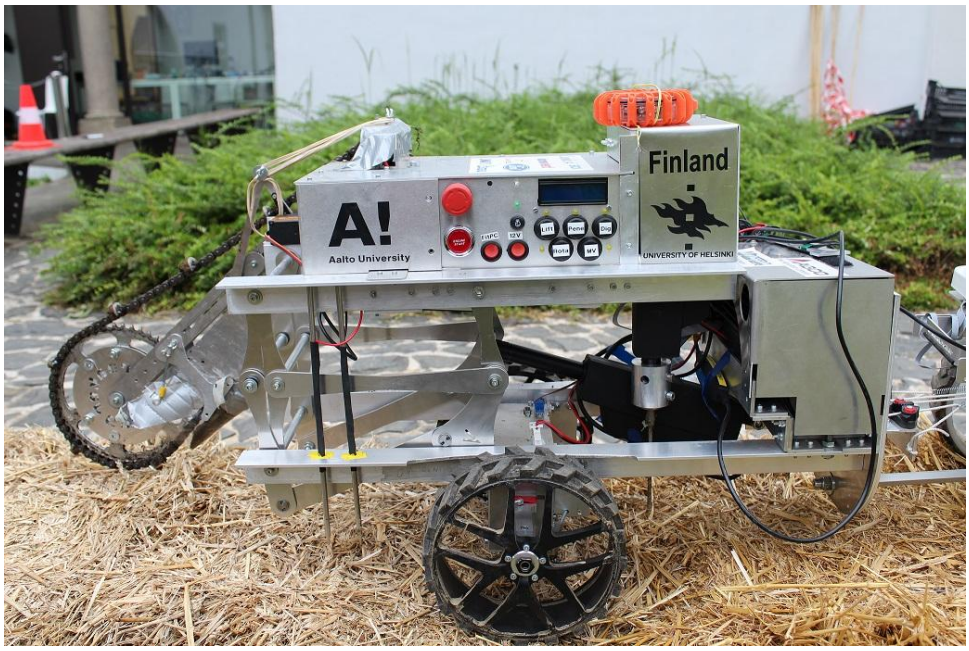


Figure 39 The trailer in Slovenia at the Field Robot Event 2015.

The two spikes on the left in the figure 31 are soil conductivity sensors which can measure the amount of moisture in the soil. The digger is on the back side of the trailer.

The soil sampler (below the "Finland"-label) requires plenty of force to penetrate the soil. This meant that the trailer must be heavy for the penetrometer to work. The weight of the trailer was increased intentionally by including three lead acid batteries, which are housed in the box on the lower right of the trailer.

There is also a weight distribution mechanism which transfers weight from the main robot in to the trailer. If the soil is very hard, this keeps the trailer on the ground when taking the penetrometer measurement. The strings and a spring connect the trailer to the main robot as shown in Figure 40.

Figure 40 Picture of the weight distributor.

5.2 Electronics

The following electrical components were used in the trailer:

- Linak LA12 linear actuator with position sensors, two pieces, lift mechanism and penetrometer
- Digger chain motor, RS555 type motor fitted with P60 Gearbox, ratio 132:1 (www.banebots.com) Savöx SV-0235MG, Servo for moving the digger arm
- MCLENNAN 1271-12-392 MOTOR, GEARED, 12V, 7RPM, 392:1, the carousel motor
- MULTICOMP MCPIP-T12L-001 SENSOR, M12, PNP, inductive sensor
- FitPC, fit-PC2i 2GB/1.6GHz, For machine vision processing, accompanied with a Kvaser Leaf Light HS CAN bus analyser (www.kvaser.com)
- Genius WideCam 1050 (webcam)
- Crumb128-CAN with Atmel AT90CAN128 (embedded microcontroller)
- Verbatim Ultra-Slim Portable Power Pack 98450, 4200mAh, USB, power distribution 5V, two pieces
- Sealed lead acid battery 12V/5Ah, three pieces (FitPC, Can hub, Actuators)
- CRX317 Charger for lead acid battery, 12V/1A, 3-Phase, automatic
- USB Charger for power pack charging, 5V/2,1A

5.2.1 Trailer electrics diagram

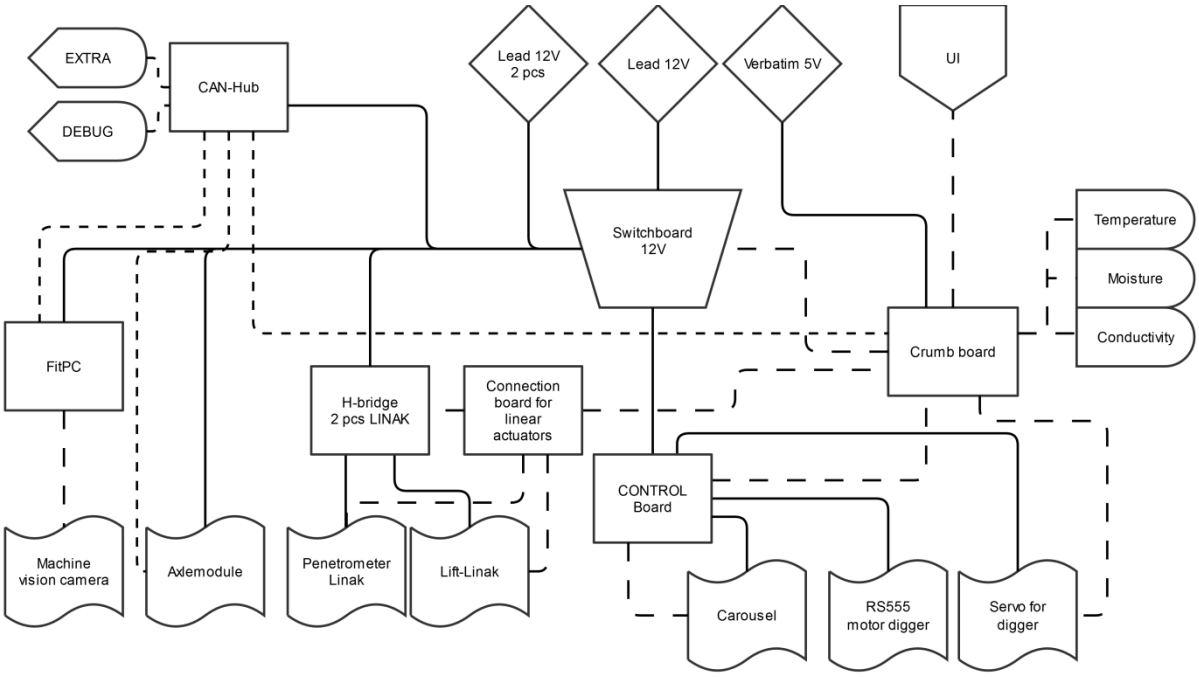


Figure 41 Continuous lines correspond to the power lines. Dashed lines mark the digital or analog signal wires.

5.2.2 Printed Circuit Boards in the trailer

The trailer PCBs were designed with the KiCad EDA software. The schema, layout, printing and drilling maps were all done with the KiCad. The positive resist coated, double sided PCBs were used in the process. The circuits were transferred from transparent paper onto the photoresist with ultraviolet light. Natrium hydroxide was used to remove all of the excess photoresist and etching was done with hydrochloric acid. The holes were drilled on to the PCB with a CNC drill. Holes larger than 3mm were drilled by hand. PCBs were cut to final size and inspected thoroughly before soldering. All components were hand soldered. Data connectors between boards were IDC10 cables. CAN hub had two connector types RJ11 and DB9 for data connections. Power was mostly transferred through screw terminals and Faston connectors.

A Crumb128-CAN module with an Atmel AT90CAN128 was used as the main microcontroller of the trailer. There are seven IDC10 connectors, an RJ11-connector for the CAN bus and a reset switch on the microcontroller board. A Verbatim Ultra-Slim Portable Power Pack powers the microcontroller through a USB connector.

Microcontroller is used for the folowing functions:

- Buttons, lights, LCD and sound
- Linear actuators and digger servo control by PWM
- Relay information to the user via CAN bus, which can then be used to visualize the measured values. A tablet was used during the competition to demonstrate the soil compactness for the judges.
- Communicate with the main robot through the CAN bus
- Read the digger motor's encoder signal
- Use the microcontrollers ADC for:
    - Position signals of the linear actuator
    - Penetrometer current measurement
    - Digger motor current measurement
    - Temperature measurement
    - Soil conductance

5.2.3 Control Board

The control board has two relays for the DC motors and a voltage regulator to power the digger servo. The relay that drives the carousel motor is oversized so that it can be used to power the digger motor if its own relay burns. ULN2003 is used to drive the relays and is controlled by the microcontroller board. The carousel has an iron position marker which is monitored by an inductive sensor (MCPIP-T12L-001). The microcontroller uses this information to drive the DC motor of the carousel correctly. The digger motor had an encoder attached, but it was not used.

5.2.4 Servo voltage regulation and power distribution

Servo, Savöx SV-0235MG, operates on 7,4V which is generated with a linear regulator(LM338) from the lead acid battery. The linear regulator proved to be

inefficient and required a hefty heatsink. Everything worked in the test like it was supposed to, until we attached this particular servo model and stressed it a bit. For some unknown reason the voltage rail started to rise. The problem did not occur with other servo models. With the help of a larger capacitor, 1000 µF instead of 1 µF, the servo voltage was stabilized to an acceptable level. Luckily the servo was not damaged and nothing else was connected to the voltage rail.

5.2.5 Carousel motor

A geared DC motor (MCLENNAN 1271-12-392) was used to power the carousel. The motor is small enough to fit easily in the center of the plate, but a little bit more torque would have been appreciated.

5.2.6 Axlemodule

The trailer utilizes the same axle modules as the main robot, this increases the amount of spare modules available for the main robot and decreased the design work. The trailer did not have to be steerable so this module was not equipped with a steering servo.

5.2.7 Linear actuators

The trailer has two Linak LA12 linear actuators to power the lifting mechanism and the penetrometer. The linear actuators are controlled by two H-bridges that are nearly identical to the ones used in the axle modules. This decision simplified the design work and meant that there were plenty of spare H-bridges available for the competition. The penetrometer H-bridge uses a more sensitive current measurement IC which provides more resolution for the soil compactness value. The H-bridges are controlled by the microcontroller board using two PWM signals and the current measurement is read using the AT90CAN's built-in A/D converter.

A separate breakout board was designed that combined the H-bridge signals together with the linear actuator position feedback signals. These signals were then routed to the microcontroller board using a ribbon cable which made the wiring much more reliable.

5.2.8 User interface board

The user interface board does not contain a microcontroller but serves as a common attachment point for the LCD, LEDs and buttons. The microcontroller board is responsible for controlling the UI elements. The UI is also attached to the switchboard's ULN2003 which controls the strobe light and a speaker. The power input for the UI was taken from a Verbatim 5V battery.

Buttons in the UI	**Description**	**LED's in the UI**	**Description**
µC ON/OFF	rocker switch controls FitPC and Can-hub	µC LED inside rocker switch	ON/OFF
12V ON/OFF	rocker switch controls 12V Actuators	12V LED inside rocker switch	ON/OFF
STOP (Emergency)	Cuts power from actuators		
START	Initializes selected program and pauses the program	START LED	ON all the time while 5V battery line is connected and OK
Program select buttons:	Program is processed next time program sequence is initialized.		
LIFT	Selects/deselects	LIFT LED	Program ON/OFF

	Lifting mechanism task		
PENE	Selects/deselects Penetrometer task	PENE LED	Program ON/OFF
DIG	Selects/deselects soil sampling task	DIG LED	Program ON/OFF
ROTA	Selects/deselects Carousel task	ROTA LED	Program ON/OFF
MV	Selects/deselects Machinevision task	MV LED	Program ON/OFF
		LED µC	ON/OFF
		LED Blinkalive	µC OK

Figure 42 Table of buttons and led operations in the GroundBreaker trailer UI.

5.2.9 CAN hub PCB

The CAN hub acts as the common connecting point for the CAN bus. Six devices can be connected with the option of powering the connected devices through the CAN bus.

5.2.10 Switchboard

The power input and distribution consists of three lead acid batteries. There are two batteries for the actuators and one for the computers. The loads are connected with the Faston connectors or screw terminals and protected with the blade fuses.

The power flow is controlled with two relays (SRKE-1AT-SL-12VDC). The relays are controlled by the UI buttons through an ULN2003 darlington array. The ULN2003 also

handles speaker and sound controls which were operated by the trailer microcontroller via a UI board connection. The LEDs (Light Emitting Diode) were attached to monitor the power distribution state on both relays.

The lead acid batteries were charged with a CRX317 charger with a fused plug. In order to avoid accidents with reverse polarity, the T-connectors were used for the batteries.

5.2.11 Computers

The FitPC was used for machine vision. The FitPC has a Windows XP operating system and a MATLAB program which sent message through the CAN bus after successful identification. The computer utilizes a Kvaser Leaf Light HS (USB to DB9 cable) to connect to the CAN bus.

A webcam, Genius WideCam 1050, was used for machine vision. The camera has a wide angle lens which makes it suitable for monitoring the ground. The camera was connected via USB to the FitPC.

5.3 Software

As the trailer was designed to work autonomously, it does not require any input from the main robot. The main functionality of the trailer is controlled by a Crumb128CAN-microcontroller. The code is written with the CodeVision AVR program and the programming language is C. The microcontroller controls all the actuators in the trailer except the axle module, which is controlled by the main robot.

The trailer also has a machine vision system for positioning the trailer during the freestyle demonstration.

5.3.1 Main control program

The main loop of the control program is divided into five sections, which can all be enabled and disabled from the UI. The sections are:

- Machine vision

- Lift mechanism
- Penetrometer
- Soil sampling
- Sample carousel

A detailed info of the machine vision is located in the next section.

The lift mechanism uses a linear actuator which has an internal sliding potentiometer for the position feedback. The position is read once every iteration of the program loop and if it exceeds the set limit, the actuator is stopped. If enabled, the upper platform is lowered before any other actuators are used, and the platform is lifted after all the other functions are done.

The penetrometer functionality is similar to the lift mechanism with added current measurement function. By measuring the current drawn by the linear actuator, we can calculate the power and subsequently calculate the force used to penetrate the soil. The dimensions of the tip of the penetrometers rod combined with the force measurement allows the calculation of the soil pressure. The microcontroller sends the pressure readings and the location of the penetrometer tip to the computer which draws a graph that displays the soil compactness.

The soil sampling is done with a DC motor and a servo which lowers and rises the digger head. The motor is started and the digger head is lowered. When the servo reaches the set limit, the digger stays in place for five seconds and then rises up. At the set upper limit both the servo and digger motors stop.

The last action in the sequence is turning the sample carousel. After a sample has been taken, the carousel is rotated so that an empty container is available for the next sample. A position feedback for the carousel is achieved by using an inductive sensor.

5.3.2 Machine vision

The purpose of the machine vision of the trailer was to detect tennis balls with different colours. The tennis balls were used to indicate the borders of the sampling-zone. The

colours that were used were red, orange and blue. Each colour holds a certain command for the robot (Figure 43).

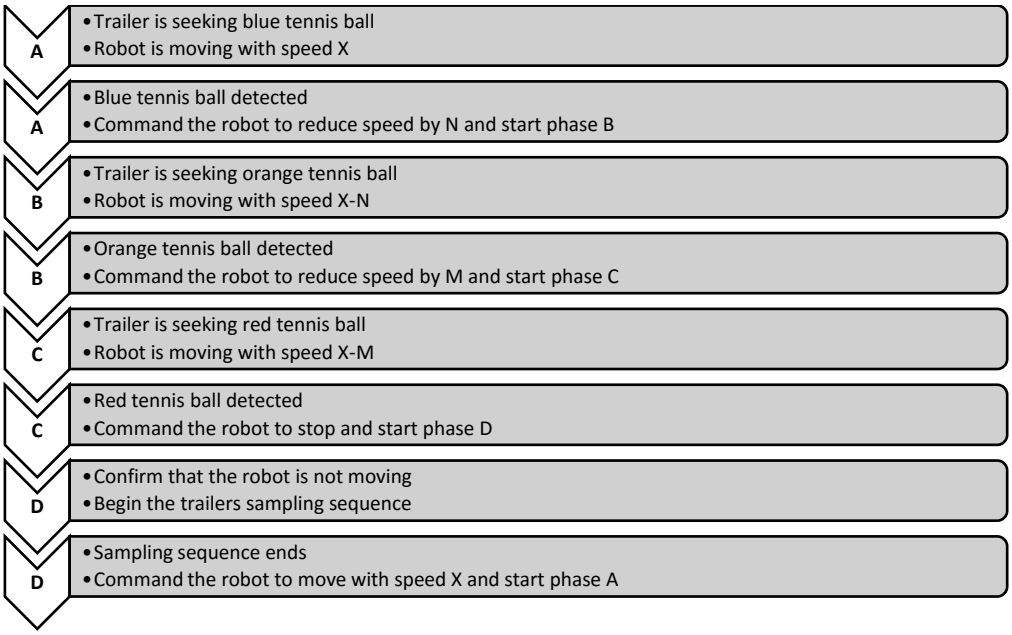


Figure 43 The idea of the trailers machine vision

The ECCI transform was used to find the target colours which were blue, orange and red. The ECCI transform has been used in previous field robots and it has proven to be easily tuneable and very efficient.

The ECCI transform is based on the EGRBI transform. In the ECCI transform, the RGB colour space is projected into a rotated colour space where one of the principal vectors is in the direction of the target colour, one in the direction of intensity (the white colour) and the last one is selected so that the base remains orthonormal. At first the algorithm divides the original image to the red image, green image and blue image and each image is used to create three new images: EC image, CC image and Intensity image. The equations 1, 2 and 3 are used to form EC, CC and Intensity channels [1].

$$EC = [Red \quad Green \quad Blue] * EC_{target} \qquad (1)$$

$$CC = [Red \quad Green \quad Blue] * (EC_{target} \times I_v) \qquad (2)$$

$$I = [Red \quad Green \quad Blue] * I_v \qquad (3)$$

, where Red, Green and Blue are the red, green and blue components from the original image. I_v is the intensity value 1/3. The EC_{target} is calculated by equation 4.

$$EC_{target} = \frac{1}{2*(R^2-(B+G)*R+B^2-B*G+G^2)} \begin{bmatrix} 2*R-G-B \\ 2*G-R-B \\ 2*B-R-G \end{bmatrix} \qquad (4)$$

, where R, G and B are the RGB values of the target colour scaled for 0 to 1 range. The target colour values for each colour were determined from the original image using data cursor or impixel-function in Matlab.

After ECCI-transform the EC, CC and Intensity channels were thresholded separately and combined to a result image using &-operator in Matlab (Figure 44). The correct threshold parameters for each channel had to be determined manually.

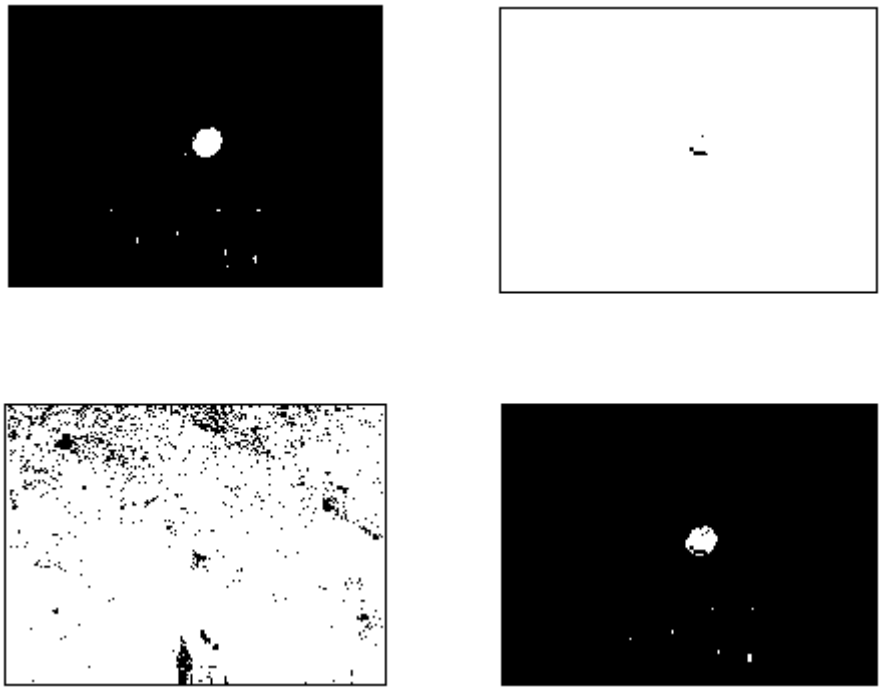


Figure 44 The threshold images and the combined image. On the top left is the EC channel and on the top right is the CC channel and on the down left is the Intensity channel after threshold. On the down right is the result image where EC, CC and Intensity channels are combined.

The result image contained some noise, so morphological operations were performed. The areas containing fewer than 50 pixels were removed (bwareaopen) and then the holes inside the white areas were filled (imfill). After these operations a closing operation was performed (imerode and imdilate). After smoothing the image, object detection was implemented with the Hough transform for circles in Matlab (imfindcircles). The Hough transform requires the edges of the object so a canny filter was used to detect the edges (edge ('canny')). Finally the found tennis ball was visualized in the original image (viscircles). The smoothing results, edges and visualization of the found tennis ball can be seen in Figure 45.

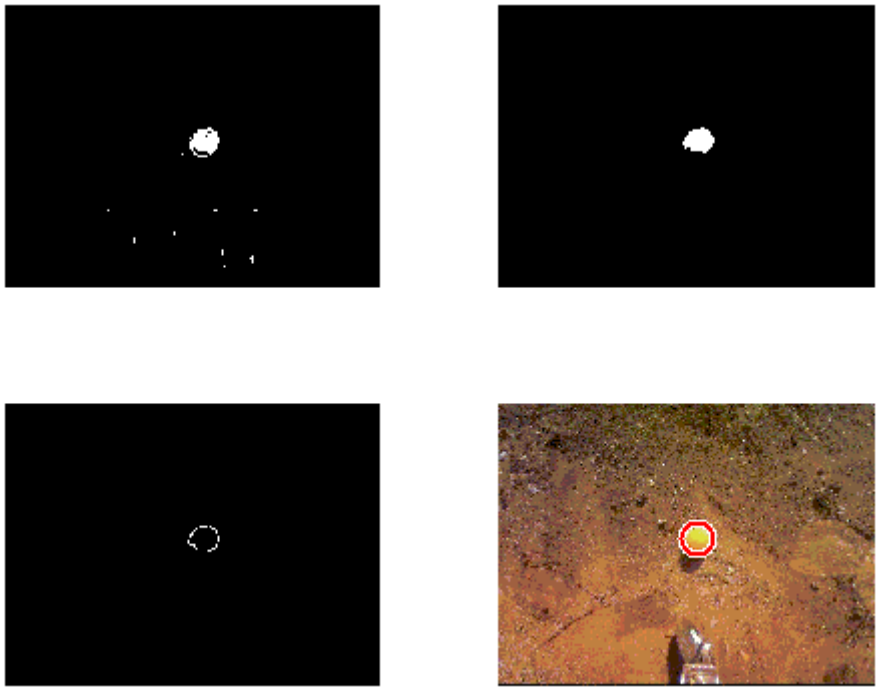


Figure 45 The result image, morphological operation, edge detection and the found tennis ball visualized. On the top left is the result image where the EC, CC and Intensity channels are combined. On the top right is the result image after the morphological operations and on the down left is the result image after the canny edge detection. Finally, on the down right is the original image where the result of the Hough circle detection is visualized

6. Discussion and Conclusions

During the project we learned about the importance of getting things done in the planned time. We didn't anticipate that a working robot built from scratch would require so much time. The amount of work was realized quite late in the project and led to failing schedules and many long evenings and nights. This also resulted in features being dropped even after many hours were already dedicated to them.

The mechanical design of the robot was a partial success. The new frame design resulted in a shorter robot with a working suspension system and the quick attachment

for the axle modules allowed for easier maintenance. However some problems were encountered with breaking axles and an insufficiently sized steering servo. Steering servos would stop working intermittently and embedded software could not recover from this because of software bugs. Using 1/5th scale axles instead of 1/8th scale parts could have solved the problem but would probably have resulted in extensive changes to the whole drivetrain. Current construction requires cutting RC drive shafts in order to lengthen them, this creates a weak link. The drive shafts were quite strong but almost all of them broke during the competition because the embedded software in the axle module contained numerous software bugs. These bugs caused uncontrolled acceleration that was too much for the axles.

The electronics of the robot worked without major flaws but multiple production cycles were required to achieve that result. Some minor bugs still remained however. The battery cut off could be switched on by the back EMF generated by turning the steering servo and some late installation jump wires were required for turning off the robot.

The biggest successes in the software side of the project were related to the working trailer code, the CAN communication as well as the good positioning and navigation algorithms. However the remote UI code was never quite finished, the plant detection code failed to work at the competition and we never had the time to write the planned GPS code. The competition results could have been highly improved by fixing the axle module code and increasing the drive speed. A fight for the first three places of the competition would not have been impossible.

7. References

[1.] Maksimow, T. et al., 2007. Department of Automation and Systems Technology - Wheels of Corntune. [Online] Available at: http://autsys.aalto.fi/en/attach/FieldRobot2007/FRE2007_WheelsOfCorntune.pdf [Accessed 1.8.2015]

[2.] Vepsäläinen, J. et al., 2014 FinnFerno final report. [Online] Available at: http://autsys.aalto.fi/en/attach/FieldRobot2014/FinnFerno2014_FinalReport_web.pdf [Accessed 25.8.2015]

[3.] Piirainen, P. et al., 2013 Double trouble final report. [Online] Available at: http://autsys.aalto.fi/en/attach/FieldRobot2013/DoubleTrouble_FinalReport.pdf [Accessed 25.8.2015]

[4.] ISO 11783-7. 2002. Tractors and machinery for agriculture and forestry – Serial control and communications data network, Part 7 Implement messages application layer. International standard.

PARS - Gaziosmanpaşa University, Turkey

Mustafa TAN (Captain)[1], Mehmet Metin ÖZGÜVEN[1], Muzaffer Hakan YARDIM[1], Cemil KÖZKURT[2], Mustafa ÖZSOY[3], Sefa TARHAN[2]

[1] Gaziosmanpaşa University, Department of Biosystem Engineering
[2] Gaziosmanpaşa University, Department of Mechatronics Engineering
[3] Gaziosmanpaşa University, Department of Electrical and Electronics Engineering

Gaziosmanpaşa Üniversitesi,
Taşlıçiftlik Yerleşkesi,
60250 TOKAT / TÜRKİYE
mustan79@gmail.com

1 Introduction

Since 2003, several teams from the different universities of the world have participated into “Field Robot Event” organized internationally by the prominent universities of Europe every year; and they compete with the robots they developed. We would like to participate into the competition as Gaziosmanpaşa University.

The competition reveals the future vision of precision agriculture. In today’s world, the noticeable changes prove that agricultural robots will be used in several fields of agriculture in near future. The agricultural robots run full-automatically in the fields, and this competition proves that will be real. It is the unique agricultural robot competition actualized in open area in the world. The competition also provides opportunities for international cooperation.

The Field Robot Event 2015 competition that will be hosted by University of Maribor this year will be held in Hoce / Slovenia between 16 and 18 June, 2015.

Our robot called “PARS” was created with a strong team including the graduate and postgraduate students and lecturers from Biosystem, Mechatronics and Electric-Electronic Engineering in order to provide participation into international field robots (Field Robot Event 2015) competition.

2 Mechanical Design

2.1 Chassis

Our robot be designed at a mechanic order that would fulfill its purpose as the simplest. In this sense, “Mad Force Kruiser VE” mechanic chassis accepted through its power and durableness would be used (Figure 1,2). The robot would be built upon this chassis.

Figure 1 Mad Force chassis

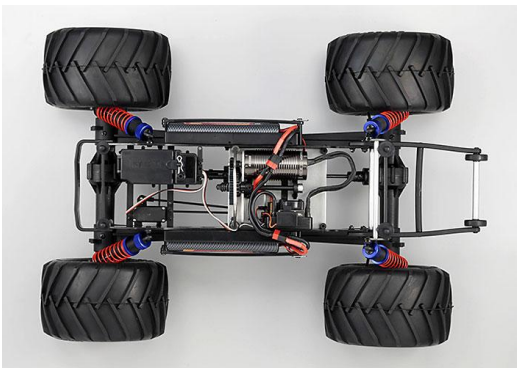


Figure 2 Mad Force Chassis top view

As the engine, 2 “Servo motors” with 15 kg/cm torque used. While choosing this motor, the factors such as the engine speed rate, turnover voltage of the engine, the current drawn by the engine, and size of the engine were considered. In Figure 3 and 4, the engines were presented, and in Figure 5, the power transmission drawings to wheels were shown.

Figure 3 Brushless DC Servo motor and ESC transmission

Figure 4 Servo motor power

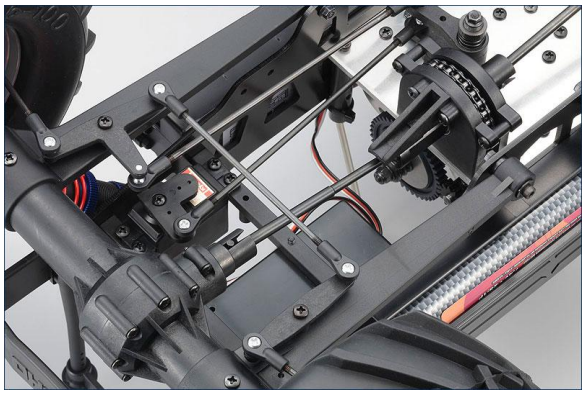


Figure 5 Power transmission (bottom) and drivetrain

Figure 6 Suspensions and differential gear case

3 Electronic

3.1 Controls

The electronic circuit should collect all data transmitted from all sensors, should transform to microcontroller and drive the engines for practicing the decisions made in the microcontroller. Consequently, there should be a microcontroller circuit as the main circuit, and sensor circuit, motor drive circuit and a voltage regulator circuit as the other peripheral units for a robot. Those would be used in ready-to-use modules. “Arduino series of DUE (Figure 7)" embedded system would be used as the main circuit. And the other peripheral units would be connected to this card.

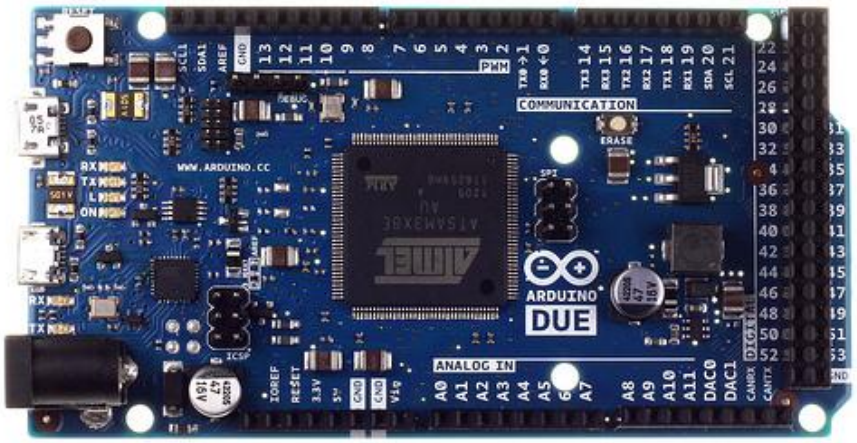


Figure 7 Arduino DUE

3.2 Battery

The main idea in battery selection was easy-changeable battery use at equal sizes. For that reason, we decided to use lithium polymer battery (LIPO). These batteries are fairly light when compared with lead acid batteries, but have some differences. One of the most important difference was battery voltage's being between 3.0 and 4.2 volt/cell. Therefore it is necessary to use shear circuit for preventing discharge in low battery voltage. The charger should also be "smart." The charger starts to charge through the constant current set by the user, and the when voltage reaches to 4.2 volt per cell, it starts to charge with the constant current. Misuse of batteries can cause fire.

3.3 Sensors and Cameras

The robot should have sensors providing it to have peripheral communication in order to make decisions in accordance with its purpose. The robot include ultrasonic, color and movement detecting sensors as the leading.

3.4 Software

While writing the software, the algorithm we organized, and I/O pins of the microcontroller in our electronic circuit should be considered. C/C++ will be used as the

programming language, and PROTEUS used as the simulation. The code we wrote converted into machine language through ARDUİNO compiler, and transmitted into microcontroller through the programmer.

3.5 Real Time Image Processing Board

Beagle Board xM used to making decisions using video streams acquired from webcam that connected to Beagle Board xM. The portable minicomputer board, including 1 GHz processor, 512 MB memory and many superior capabilities, is suitable to run real time process algorithms such as image processing. The board is supported Angstrom Linux, Ubuntu, Android and XBMC operating systems. The algorithms of computer vision based agricultural tasks can be coded in some languages like OpenCV, SIMULINK, C++ or Python. Designed algorithms would be coded and embedded to the board's memory to run as real time on PARS to perform visual tasks.

4 Algorithms and Methods

This is one of the most important steps, maybe the most important step, in robot design. Algorithm is the logical listing of the things to do, in brief. It is necessary to decide on what our robot should perceive, which sensors it will use and which process is more important. For that reason, the main algorithms and methods of the robot will be Line perception, Line break perception, Return skills and Plant determination.

The study plan will be as such. Firstly, a simple and easy-to-use method will be developed. Then, a more improved method will be developed providing the input and output interfaces of the algorithms to remain as the same. Doing such, it will be easier to learn what the robot should perform. So, we will analyze the aforementioned algorithms and methods, and choose the method providing the optimum result.

5 Conclusion

Our robot which has the outer casing that done by us have originality. And it was an advantage. In addition we use two different embedded system board, and we still had the advantage of using two different cameras.

But before the trip to participate in the competition to forget an important material in the preparation, all these advantages turned into disadvantages.And therefore we could not participate in the last two stages we can be ambitious.

Nevertheless, during the competition trying to find solutions, make good use of your time and learn new information and ideas from other competitors, it was the most important.

TALOS - University of Hohenheim, Germany

David Reiser[1], Robert Hübner[1], Hans W. Griepentrog[1]

[1]University of Hohenheim
Institute of Agricultural Engineering
Garbenstr. 9, D-70599, Stuttgart, Germany
dreiser@uni-hohenheim.de

1 Introduction

Moving in agriculture from big machines to small, autonomous machines, can have a key role in solving actual challenges in crop production like soil compaction, machine transport, security or human labour costs. For that competitions like the Field Robot Event can help to speed up development of mechanics and algorithms for this tasks. Detecting rows is mostly sufficient for robot navigation in semi-structured agricultural environments, as most of the current crops are planted in row structures. Detecting line structures is topic of many researches, performed by camera images ([1],[2]), light detection and ranging (LIDAR) laser scanner data ([3],[4],[5]), or other types of sensors. But uncertainty in environment still makes it challenging to detect the values in noisy sensor data, like it is typical in agricultural sites [5] and could be seen in the Field Robot Event. For this robot we used a Lidar based Random Sample Consensus (RANSAC) [6] line detection algorithm for the navigation, assisted by encoders and a inertial measurement unit (IMU) for headland turning.

Beside of just navigation, there is always a need for robotic research to show the use and ability for future applications, what we tried in the free style competition. For that our team focused on the potentials for detailed mapping of metal objects on a field. Every year a bunch of metal pieces on fields can be found, what cause damage on wheels and machines. To detect, map and remove this objects, could help to avoid repair costs.

2 Mechanics

As basic vehicle a small 4-wheel autonomous robot with differential steering named "Talos" was used (see Fig. 1,a). The size of the robot platform was 500 x 600 x 1100 mm. The weight of the robot is 40 kg and it is equipped with four motors with a total power of 200 W. Maximum driving speed is 0.8 m/s and a maximum static motor torque of 4 x 2.9 Nm.

For the Freestyle Event there was an attachment trailed, what was carrying 6 metal sensors and 6 magnets and was able to shift them up and down (see Fig 1/b). For the trailer a Linak motor (2362002002XXXG4) with 1500N maximum load was used.

A B

Fig. 1. Robot mechanics (A) and the attachment used in the freestyle event (B)

3 Hardware

3.1 On board computer & controller

The robot is controlled by an embedded computer, equipped with i3-Quadcore processor with 3.3 GHz, 4 GB RAM and SSD Hard drive. For energy supply, two 12V/48Ah batteries are providing an operating time of around 4-5 h, depending on the load torque, task and additional weight of equipment placed on the robot platform.

For the control of the freestyle attachment a separate Raspberry Pi 2 was used, equipped with an analog digital converter (ADC0838CCN), a Motor driver chip (LD2930) and a two contactors. The Motor driver chip was connected to the Linak linear motor. The computers communicated via Ethernet, using ROS on multiple machines. The four motors had been controlled by the embedded computer by two motor controllers (Roboteq, SDC2130). For the remote control a standard X-Box Joystick was connected with a Bluetooth dongle. For activating the magnets, of the attachment a standard contactor was used. The data flow diagram can be seen in Fig. 2.

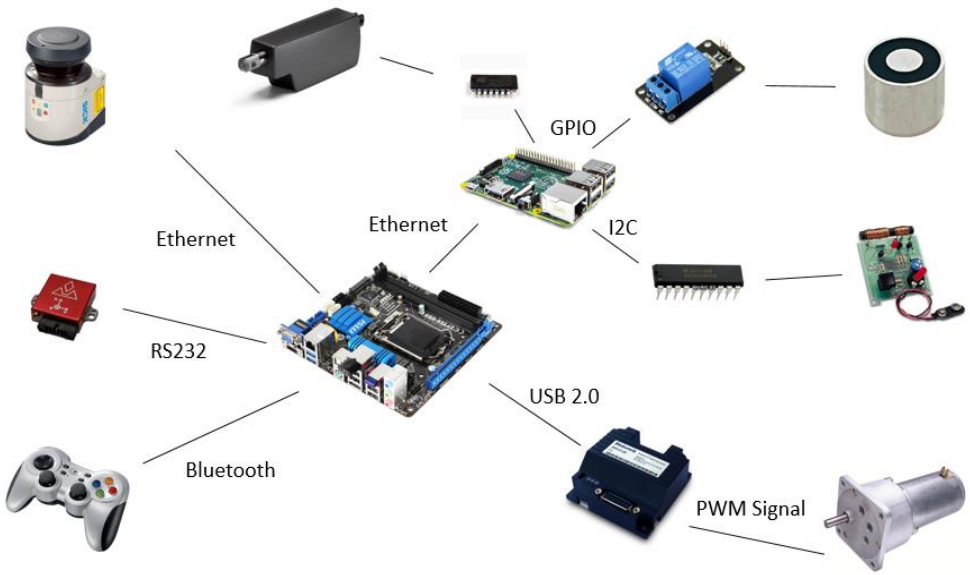


Fig. 2. Data flow diagram of the robot sensor setup

3.2 Sensors

The robot system is equipped with wheel encoders connected directly to the motor controllers, a VN-100 Inertial Measurement Unit (IMU) (VectorNav, Dallas, USA), two LMS111 2D-LIDAR laser scanners (SICK, Waldkirch, Germany). The laser scanners were mounted horizontally at the front and the back of the robot at a height of 0.2 m above the ground level.

For the freestyle Event additional 6 inductive metal detector sensors had been attached to an ADC converter connected to GPIO of the raspberry pi 2.

4 Software

The robot computer runs by Ubuntu 14.04 and use the Robot Operating System (ROS-Indigo) middleware for the data recording. The Raspberry Pi 2 runs by Raspbian-Linux and got also the ROS-Indigo middleware installed. All software components had been programmed in a combination of C++ and Python programming languages.

For the navigation software the data of the two laser scanners had been used. For that the data was first transformed into a 2D point cloud and was filtered in the two regions of the lines. The LIDAR data was collected with an average of 25 Hz and a resolution of 0.5 degree. For the fine calibration the filter values could be changed via runtime. Out of the two separated point clouds the RANSAC row detection algorithm could be applied to each row separate. The driving direction was afterwards resolved by adding the angles of both detected lines together. As soon as the defined headland region did not detect any objects, the headland turn was started. For evaluation of the algorithm outcome, the lines, the laser scanner data and the expected headland region had been visualized with markers. These could be shown in real-time in the RVIZ visualisation tool (see Fig. 3)

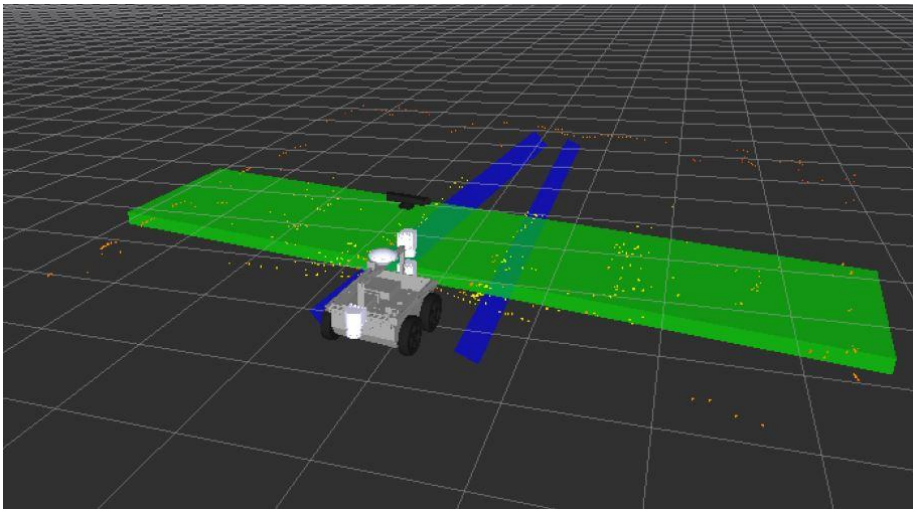


Fig. 3. Rviz visualisation of the detected lines (blue), the laser scanner data (dots), and area for headland detection (green square)

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. For the change between the modes, a mode-changer [7] was implemented, what made it possible to overwrite the autonomous mode with a user joystick input, and also to separate the program code to different tasks. In general 3 modes had been used: User Mode; Headland Turn; In Row navigation. For each mode, the motor controller subscribed to a separate speed topic, provided by the joystick, the laser scanner or the odometry with a point to point navigation.

For every task there was a different programs written to provide this speed information. For the freestyle event the odometry was used to provide georeferenced for the sensor values received.

5 Conclusions

The project helped to gain a huge amount of experience in the use of sensors under outdoor conditions. The results of the field robot event showed the problematic in using algorithms in outdoor environments and the high change in the variability of plants and surrounding. The system worked well under known circumstances but failed under high and unexpected changes. This caused to think about more options to avoid malfunction of sensors and have more options to check the correct performance of the robot before starting autonomous tasks.

The freestyle helped to find a relevant application for the robot what could help to make work in agriculture more efficient. Also could the vehicle be used for a lot of different other applications with the use to map analog sensor values together with a robots help. Especially for Precision Farming there could be a huge field of applications for autonomous robots in future.

Acknowledgments

The project is conducted at the Max-Eyth Endowed Chair (Instrumentation & Test Engineering) at Hohenheim University (Stuttgart, Germany), which is partly grant funded by the DLG e.V.

References

[1.] J. Marchant and R. Brivot, "Real-Time Tracking of Plant Rows Using a Hough Transform," *Real-Time Imaging*, vol. 1, no. 5, pp. 363–371, 1995.

[2.] G. Jiang, C. Zhao, and Y. Si, "A machine vision based crop rows detection for agricultural robots," in *Proceedings of the 2010 International Conference on Wavelet Analysis and Pattern Recognition*, 2010, no. July, pp. 11–14.

[3.] S. Hansen, E. Bayramoglu, J. C. Andersen, O. Ravn, N. A. Andersen, and N. K. Poulsen, "Derivative free Kalman filtering used for orchard navigation," in *13th international Conference on Information Fusion*, 2010.

[4.] O. C. Barawid, A. Mizushima, K. Ishii, and N. Noguchi, "Development of an Autonomous Navigation System using a Two-dimensional Laser Scanner in an Orchard Application," *Biosyst. Eng.*, vol. 96, no. 2, pp. 139–149, Feb. 2007.

[5.] S. A. Hiremath, G. W. A. M. van der Heijden, F. K. van Evert, A. Stein, and C. J. F. ter Braak, "Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter," *Comput. Electron. Agric.*, vol. 100, pp. 41–50, Jan. 2014.

[6.] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography." 1980.

[7.] H. W. Griepentrog, B. S. Blackmore, and S. G. Vougioukas, *CIGR Handbook of Agricultural Engineering Volume VI: Information Technology*. Michigan: American Society of Agricultural Engineers, 2006.

Zephyr – University of Siegen, Germany

Klaus Müller, Charam Ram Akupati, Felix Graf, Yuwei Guo, Sven Höhn, Jan-Marco Hütwohl, Thomas Köther, Samir Nezir Osman, Goetz Poenaru, Saeid Sedighi, Jan-Friedrich Schlemper, Whangyi Zhu, Jan Kunze, Klaus-Dieter Kuhnert

University of Siegen, Institute of Real-Time Learning Systems (EZLS)
contact: Hölderlinstr. 3, 57076 Siegen, Germany.
jan.kunze@uni-siegen.de

Abstract

This year's team Zephyr of the University of Siegen was the successor of the last year's winning team Phaeton. Our goal this year was to create a new robot from the scratch, learning from our first robot Phaeton.

1 Introduction

The 13th Field Robot Event in Maribor, Slovenia was the third competition for the team of the University of Siegen. After the last year's Field Robot Event, where our Team Phaethon was able to be the overall competition winner, this year we managed to win the overall competition again by scoring a first place in Basic Navigation and a second place in Weed Detection.

The team consists of eleven students from six different nations, of which four students have experience from the preceding project group. The students are studying computer science and electrical engineering in diploma and master degrees.

Fig. 1: ***Team Zephyr.*** *(left to right) (in the back) Yuwei Guo, Charam Ram Akupati, Jan-Marco Hütwohl, Sven Höhn, Goetz Poenaru, Jan Kunze, Klaus Müller, Samir Nezir Osman, Thomas Köther, (in the front) Jan-Friedrich Schlemper, Saeid Sedighi, Felix Graf, Whangyi Zhu.*

The current robot is based on the last year's robot Phaeton and was built anew. Our aim was to improve the problems of the last platform but also to reduce the costs by using a more cost efficient design and parts that are easier to buy and don't need to be imported from America or Asia. The robot body was improved to make it easier to repair and to exchange and access parts but also to make it more resistant to water and dust. Besides our approved sensors, Zephyr included a Kinect XBOX One [kinect] sensor for plant detection. Additionally, the driving algorithms and the plant recognition was reworked.

2 Mechanics

2.1 Updated steering system

To improve the steering system of the last robot so that it is able to steer in a shorter distance, it was necessary to change the components of the powertrain and substitute them with more flexible axles. The usage of universal joints lead to a better steering

behaviour of the robot. By calculating the range of the angle of each individual wheel in full steering statement, we chose the best fitting axle for it.

2.2 Motors

The robot has two powerful *LRP Vector K4 Brushless-Truck* motors with a maximum power of 80 watt each. They are mounted to the axles to lower the robot's center of mass. Additionally, the motor comes with three hall-sensors. The sensors are connected to the motor controller for precise power and throttle setting and also to the embedded board for measuring the rotational speed and direction. The motors are waterproof, fully adjustable and replaceable. [motor]

2.3 Servos

For the steering control Phaethon has a *HS-7980 TH High Voltage Ultra Torque* servo by *Hitec* on each axle. It is controlled with pulse-width modulation signal. Depending on the input voltage the pulling load differ between 360 Ncm and 440 Ncm with a regulating time from 0,17 sec/60° and 0,21 sec/60°. [servo]

The servos can be steered independently. Based on that the robot is capable of three different steering modes: single Ackerman mode, double Ackermann mode and crab-steering mode.

2.4 Robot body

To have an all new product the very first step is to design a new shape for it. In the Zephyr project group we tried to design and build a gorgeous, practical and light body. This body must be water and dirt proof to cover all sensors, control components and battery packages. Getting access easily to all components for maintaining them and even changing them was one of our first priorities when designing this part.

2.5 Laser scanner protector

The main scanning step of the surrounding area of the robot is done by a laser scanner. This sensor is mounted in front of the robot for efficiency, the downside of using the laser scanner in front of the robot is that there is the danger of damaging it due to collisions. To reduce this danger we designed a special cage for this part with an adjustable mechanism to be able to adjust the position of the sensor. The cage has also a cooling function for the laser scanner and is dust proof.

Fig. 2: Custom wheel model

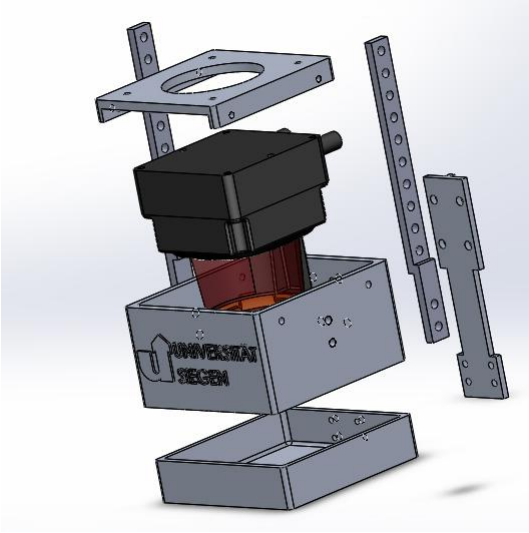


Fig. 3: Laser scanner protector model

2.6 New wheels

Because of new mechanical specification of the robot, we had to change the size of the wheels to have a higher efficiency. The performance of the robot had a high priority for us, so we decided to design and build our own wheels. The wheels were created then with the use of CNC machines in the university's workshop.

2.7 Chassis

Because of all new designed components, the dimension of the chassis had to be changed for mounting all the new parts on it. The new chassis has been designed and built by CNC machines, too.

Fig. 4: Robot Zephyr

3 Hardware

3.1 On board computer & controller

Zephyr contains three types of boards. The first one is a self-developed energy board, which provides several power sources for different types of sensors, motor- and servo controller and others. It also measures the voltage of the two accumulators and has the possibility to switch to the accumulator with the best state of charge. Two LED indicate which battery is actually in use. Another self-developed board is the UDOO Connector Board (UCB), which connects all sensors with the UDOO Board. The main feature of the UCB is signal conditioning. Furthermore, it converts the voltage for the different logic I/O level. A mini-itx mainboard with an Intel I7-4785T is used because a lot of power is needed for image processing like OpenCV. All demanding programs and

algorithms are executed on this processor. The UDOO Board provides signals like pulse-width modulation for steering, I²C communication and analogue digital conversation

3.2 Sensors

The robot hosts five types of sensors.

Laser Scanner

The most important sensor is a *Hokuyo UTM 30LX* laser scanner which has a range of up to 30 meters. It covers an area of 270° with 1024 rays and a frequency of 40 Hertz. But the accuracy of 30 mm can significantly deteriorate in direct sunlight. The laser scanner is the main sensor used in every task, for a reliable scan of the environment in front of the robot. [laserscanner]

Ultra-sonic sensors

If the robot needs to drive backwards, two ultra-sonic sensors at the rear assist the robot. For plants and equal structures this method works fine and no plants were harmed.

Inertia Measurement and Hall sensors

Equally important is the inertia measurement board which is mounted on the top of the UCB. It provides information of positive or negative acceleration for three axes, movement directions like pitch, yaw, roll and has a magnetometer on board for orientation. At least each of the two electric motors have hall sensors to measure the motors' speed.

OBE Sensor

At the rear bottom of the robot there is an OBE sensor. With this sensor it is possible to measure the real speed, the direction and the distance driven by the robot.

Kinect ONE

We use a Kinect ONE for the detection of normal and marked/unhealthy plants. The kinect is not only a video-camera, but also includes a depth-camera. For weed detection we use the kinect's depth-function to limit our observation space, so that only objects/plants in a certain area will be detected and computed while objects that are too close or too far away (e.g. plants in the next row) won't be observed. We create a point-cloud which consists of points with depth and color information. The color information is used to recognize marked/unhealthy plants.

4 Software

4.1 ROS

The ROS (Robot Operating System) is a software framework for robots. In ROS there are packages which contain small programs, so-called nodes. These nodes communicate with each other, by using ROS topics. A topic is a message-bus which may contain several publishers or subscriber. The ROS-nodes are used with all our sensors, including the laser scanner and the Kinect ONE.

The ROS-Core manages this message stream by registering those publishers and subscribers and assigns them to the respective receivers. ROS-Messages support all standard data formats, such as strings and vectors, are predefined and provide time stamps. That ROS topics can be sent over networks, which enables easy and fast communication between individual components. With this, we are able to split the modules, so that the complex algorithms run on a powerful mainboard, while other modules (e.g. the sensor modules) run on an embedded PC. Based on this approach there are more low level modules which communicate with the sensors and actuators, either with help of libraries for the given interface or directly.

4.2 Qstate

QSTATE was designed, to coordinate the execution of the navigation and object detection nodes. It implements a state machine, which specifies exactly when each node has to perform a certain task. When the state machine sends a message one of the navigational nodes, this starts with the entrusted task. As soon as this task is completed, it sends back a message to the state machine. The message contains a flag, which shows if the task was successfully completed, or not. Depending on the case, the state machine can respond differently.

4.3 Dead Reckoning

The position package implements the dead reckoning. Therefore, it uses the gyroscope and the optical movement sensor to calculate the position. This is realized inside a class which can be used in every part of the system.

To calculate the position, the movement detected by the optical movement sensor is rotated around the actual angle of the robot (given through the gyroscope) and finally the result is added to the last calculated position.[reck] So we're able to keep track of the position relatively to a start position(which is usually the beginning of the calculations).

4.4 Navigation in rows

The navigation in rows in all tasks was done by the same algorithm. Navigation relied mainly on laser scanner data. A state machine switched between the different behaviors of driving in a row and change rows. Following steps are made:

1. Get the laser scan data as a pointcloud data type via ROS

2. Lay a grid over the pointcloud. With a threshold divide between grid-cells with a scan point number greater than the threshold, then marked as full, and cells with less scan points, marked as empty. This way, outliers are eliminated.

3. Build clusters of the grid-cells with a simple clustering over the euclidean distance.

4. For each cluster do a line fitting and build a vector and a line equation

5. For each cluster compute the intersections of its fitted line with the line of each other cluster; Compute the absolute value of each vector of a cluster;

6. Determine the cluster that is with the highest probability a row (of plants) regarding the distance from the robot to the intersection (higher is better), absolute value of the vector (bigger is better) and the history (more frames visible is more reliable).

7. Drive alongside the best cluster and avoid collision.

8. If no clusters are visible for some time → change state of state machine

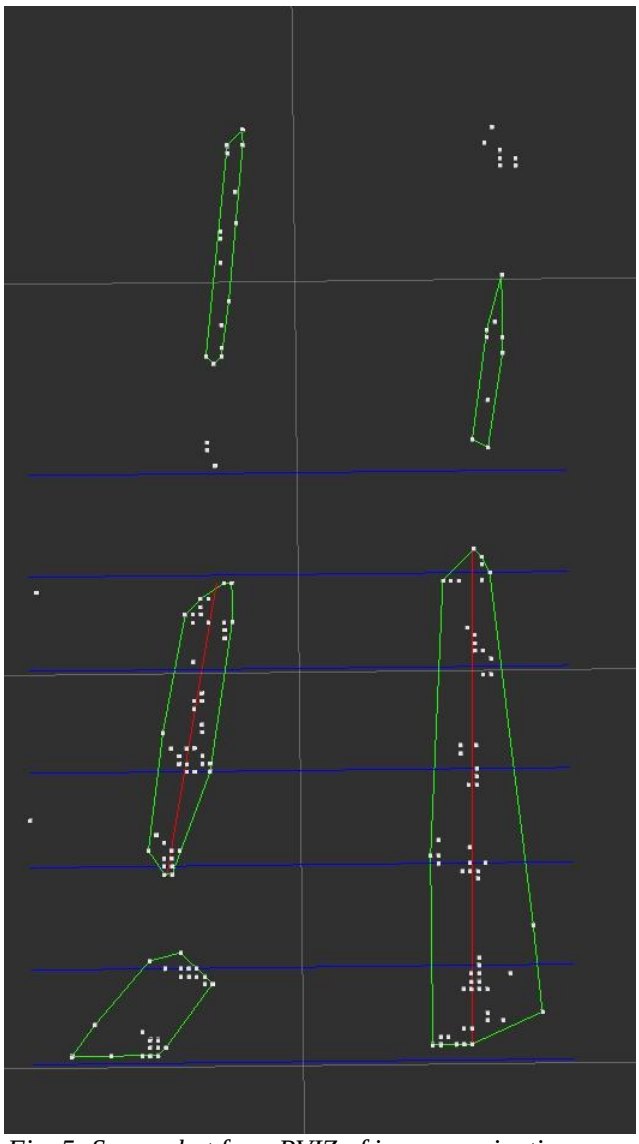


Fig. 5: Screenshot from RVIZ of in row navigation White Dots→Pointcloud; Green Line→Cluster; Red Line→Vector from line fitting for the two best rows; Blue lines→Local coordinate system; White lines → Global coordinate system; Not shown: Grid-cells and intersections

4.5 Changing Rows

Changing row also relies strongly on the laser scanner data. When triggered by the state machine the algorithm works as described below:

1. The robot steers out of the row with maximum angle of turn and stops when it is positioned orthographic to the rows.

2. The robot detects the end of the rows with the laser scanner and counts them until he passed the given number of rows.

3. Robot turns 90 degree into the row.

4.6 Plant detection

Plant detection was done using the depth and the image data from the Kinect ONE sensor.

1. Get a new point-cloud, which has depth and color information from the Kinect ONE.

2. Narrow/Limit the observation space by defining an observation region in space, skipping all points from the pointcloud that are outside.

3. Do segmentation and classification of the data with the use of color as a feature.

4. Determine position of a detected plant in each frame to determine if it is a new plant or still the old plant.

5. If the plant is new and classified as marked/unhealthy, a ROS message is sent.

5 Conclusion

We were able to successfully build the new robot platform Zephyr based on our experience with the former robot model Phaeton. Its high speed, the ROS software framework and the laser scanner could all be ported over to Zephyr. We managed to improve the batteries for longer driving duration and also for supporting more

computing power using an Intel 7 processor. The turning angle was slightly improved. However, the Kinect ONE proofed to be not as ideal as we hoped. Compared to the rest of the robot it is big and heavy and influences the balance of the robot. It also has a high power consumption while it's detection results were acceptable but leave room for improvement. The steering needs a better controller on the software side. In addition, the chassis frame showed some hardware weaknesses during the competition leading to downtimes for repairs. Nevertheless, the whole system continued the success of its predecessor and got the highest overall score winning the contest and will be building the base for further developments at our institute.

6 References

[1.] [kinect] *Xbox One, www.xbox.com/One [17.08.15]*

[2.] [laserscanner] ***UTM-30LX*** | https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html [12.07.14]

[3.] [motor] *Vector K4 Brushless Motor – Truck.* https://www.lrp.cc/en/products/electric-motors/brushless/sport/produkt/vector-k4-brushless-motor-truck/details/ [18.08.15]

[4.] [reck] Sekimori, D. , Myazaki , F.: *Precise dead-reckoning for mobile robots using multiple optical mouse sensors.* In: *Informatics in Control, Automation and Robotics II* (2007), Springer

[5.] [ros] *ROS.org | Powering the world's robots.* https://www.ros.org [12.08.15]

[6.] [servo] *Servo HS-7980TH.* *http://hitecrcd.com/products/servos/ultra-premium-digital-servos/hs-7980th-mega-torque-hv-coreless-titanium-gear-servo/product* [18.08.15]