ADIR ROW CHANGE & UI

Developing partial solutions for an autonomous spraying application

Deborah Schrag Lukas Marti

September 13, 2023 to January 7, 2024

FHGR, Fachhochschule Graubünden, University of Applied Sciences of the Grisons

Abstract

Task definition: Implement a GUI for generating presets for the robot.
Create an on-board UI for user interaction and allow the selection of pre-installed values through the UI.
Developing a row turning algorithm for autonomous plant spraying applications.

Goals: The preset creation tool needs to be a GUI, which should be able to take at least six values into account and write them into an ROS YAML file. This file is then transferred to the robot for later use.
In regards to the OBUI, the user can choose between five presets and is able to switch between auto mode and manual mode via a push button.
The row changing algorithm has to work in the test environment of the FHGR, where it has to turn into the next row of plants without hitting anything and not going over the allowed turning space. With rows that are open at the and and rows which are closed at the end.

Approach: While waiting on the robot's arrival from Hamburg, we concentrated our efforts on setting up a Gazebo simulation while working in parallel on the OBUI and PCT. Once the ADIR arrived, the focus shifted to the row change, prioritizing in field testing.

Outcome: The on-board UI and preset creation tool work as intended.
In the simulation, both of the row changes work well and reliably. In the real world however, the outcome depends quite heavily on the light conditions. Additionally, a different planner needs to be implemented, in order to be able to perform a backwards row change in unknown spaces.

Contents

1 Abbreviations

AMCL	Adaptive Monte Carlo Localization
CAD	Computer Aided Design
DC	Direct Current
ESP32	Espressif System 32
FHGR	Fachhochschule Graubünden
GUI	Graphical User Interface
ID	Identification
IMU	Inertial Measurement Unit
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LIDAR	Light Detection and Ranging
OBUI	On Board User Interface
PCT	Preset Creation Tool
ROS	Robot Operating System
RTAB-Map	Real-Time Appearance-Based Mapping
SLAM	Simultaneous Localization and Mapping
SSH	Secure Shell
SSH -X	Secure Shell with X11 Forwarding
TF	Transform Tree
UI	User Interface
URDF	Unified Robot Description Format
VNC	Virtual Network Computing
VO	Visual Odometry

XRDP	Open-Source Remote Desktop Protocol
YAML	Yet Another Markup Language

2 Records of change

Record of Changes

Version	Reason for Change	Initials	Date
1.0	Creation of this document	DS	07.11.2023
2.0	Work on project completed	DS & LM	31.12.2023
3.0	Final version of documentations	DS & LM	07.01.2024

3 Introduction

3.1 Context

The customer of this project is Ant Robotics. They are currently situated in Hamburg. Their goal is to build robots to help in the agricultural sector. This project is about programming a PCT, building an OBUI and developing a row change algorithm.

3.2 Assignment

The preset creation tool (PCT) is a GUI, with which users can set up a preset in YAML form, without having to write the file themselves. The presets then get loaded onto a robot to inform it about its surroundings. The PCT has to work on Linux Debian and has to take at least six values into account, it needs to have a GUI with pictures on it to, make it easier to understand for the user.

The OBUI is wired up on a PCB breadboard and a functional prototype of it is built. With this, the user can choose between five presets. During operation, it is possible to pause the robot, stop the program and put the robot into a manually controlled mode.

A row turning algorithm has to be designed for planters with open and with closed endings. The aim is to have it turn in a set space with as little damage to the ground as possible.

3.3 Approach

First, a specification requirements document has been written to use as a base for the different concepts.

For each task different concepts were made and compared against each other to determine the approach, which bring the most value to Ant Robotics. Important factor was also, whether they could be completed in the available time frame.

As part of the project, we were invited to visit Ant Robotics in Hamburg. There we would configure the robot together and visit a farm, where the application would take place. Afterwards, the ADIR is sent to Chur.

While waiting for the robot to arrive, a Gazebo simulation will be set up for first testing. The OBUI and PCT will be developed in parallel. Once the robot arrives, it is most important to do as much in-field testing as possible. For that, a testing environment is set up. To test approaches in real life has much more value to Ant Robotics for their particular use case.

4 Clarification of the task

In this chapter, the most important section of the specifications are shown as well as the system boundary and the project description. For the whole specifications, see attachment 5.

4.1 System boundary

For the scope of this project a system boundary diagram was made. Ant Robotics will provide us with the necessary ROS wrappers as well as a the end of row message. On our side we will develop the PCT, OBUI and row changing algorithm.

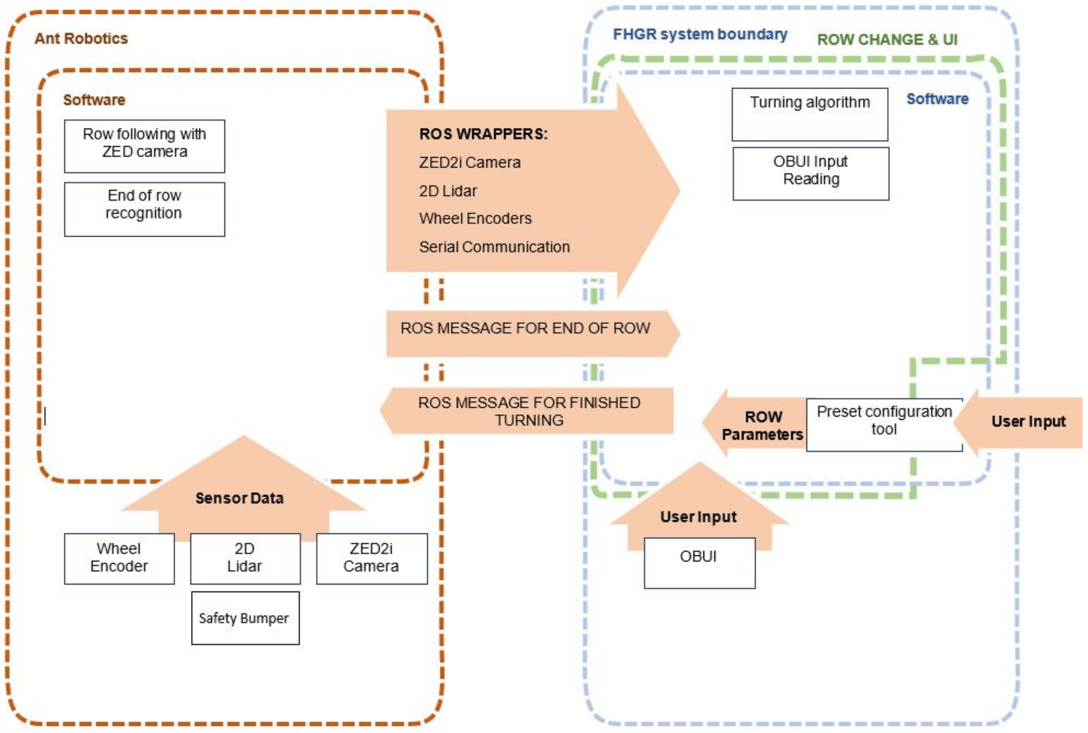


Figure 4.1: system boundaries

4.2 Project description

For the scope of this project three different sub-projects needed to be completed. The PCT which takes user input and provides us with row parameters and has to take at least six parameters into account and compiles a ROS YAML file. The presets are then used in a later component of this project.

The OBUI serves as an interface for preset selection on top of the robot. Additional functionality includes pausing the program and switching into a manual control mode. All of the inputs get sent over CAN bus to the ADIR, which is outside the scope of this project.

The row change turning algorithm takes a ROS message to start and sends out a ROS message when it is finished. The algorithm always has to make sure that the robot turns forward into the next row in a limited turning space. Turning backwards out of a row and turning forwards out of a row have to considered.

4.3 Requirements

In the following table are the most important must have requirements listed. For the full list see Attachment 5.

ID	Description
ID 1	The robot starts the row change forwards after receiving a ROS message with a boolean ”true”.
ID 2	The robot starts the row change backwards after receiving a ROS message named ”NAME” on the topic ”TOPICNAME”.
ID 3	The robot ends the row change after sending a ROS message on a topic.
ID 4	The robot drives forwards out of the row and forwards into the next row.
ID 5	The robot drives backwards out of a row and drives forward into the next row.
ID 6	The robot changes to the next row given a predetermined space.
ID 9	The preset creation tool takes 6 parameters into account.
ID 10	The output datatype of the file generated from the PCT is ”ROS YAML”.
ID 20	The user can choose between 5 presets.

ID 21	The user can press a push-button to switch between auto-mode and manual-mode.
ID 22	In the On Board UI, an Arduino or ESP32 microcontroller is used.

Table 4.1: Must-have Requirements

5 Concepts

The following sections provide an overview on concepts chosen by our client Ant Robotics. Splitting the projects in three parts:

- PCT
- OBUI
- Row change

For each of them different kind of concepts were put together and evaluated. The concepts were chosen in a concept decision meeting and additional input from Ant Robotics was added to the concepts. For a full list of all concepts see Attachment 9.

5.1 Preset Creation Tool

Functionality

Six values can be taken into account in the preset creation tool. If additional values are needed, they can be added later. The user interface lets the user input all the values into their respective boxes, give the file a name and save it.

While entering the values the user will have 2 pictures on the UI to guide them through the process. Every input value can be seen on a picture to minimize errors and misunderstandings.

Value Constraints

All values have predefined minimum and maximum limits, as well as specific data types. If any of these conditions are not met when entering values, the tool will trigger an error message.

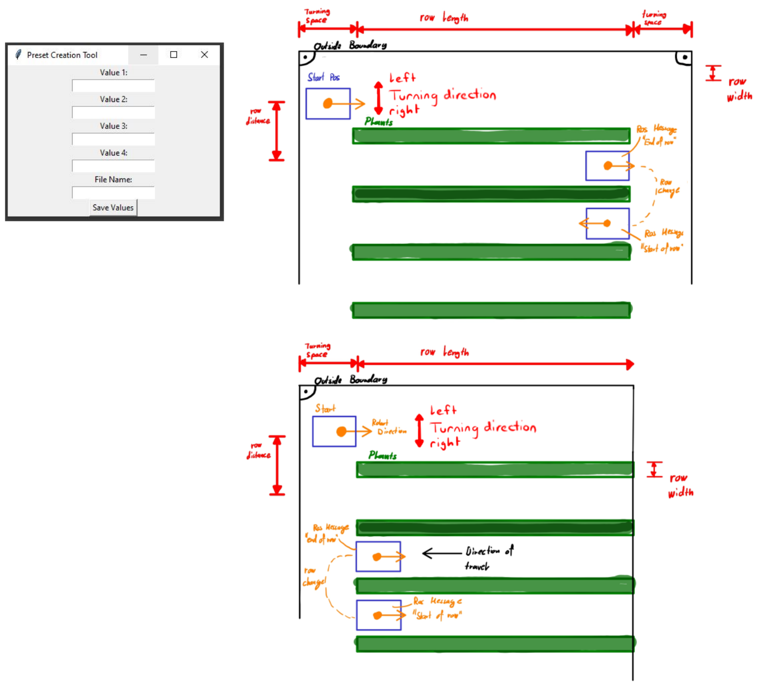


Figure 5.1: PCT concept design 1

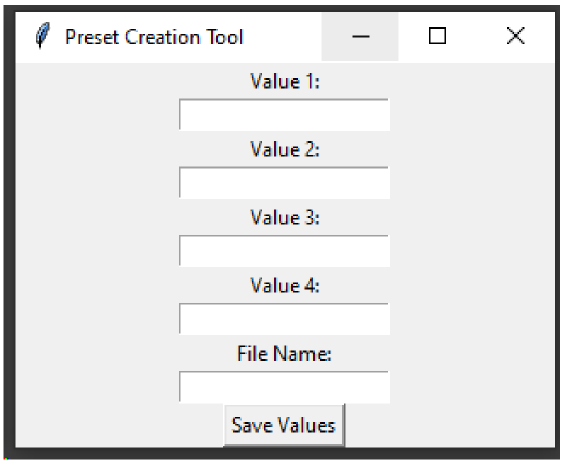
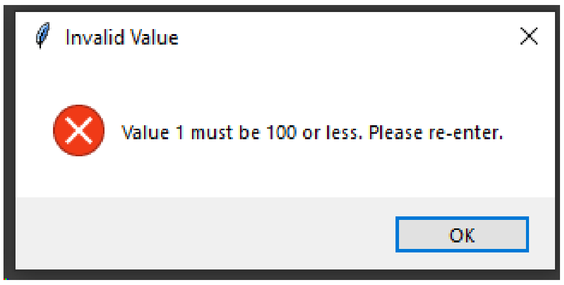
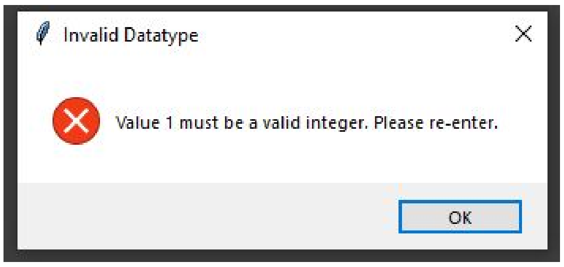


Figure 5.2: PCT concept design 1 - input window


Invalid Value

Value 1 must be 100 or less. Please re-enter.

Figure 5.3: Invalid value case


Invalid Datatype

Value 1 must be a valid integer. Please re-enter.

Figure 5.4: Invalid datatype case

Saving Presets

To save a preset, the user can provide a unique name for it. After entering the values and naming the preset, simply press the "Save Values" button to complete the process. The presets are saved as a "ROS YAML" file.

5.2 On-Board-UI

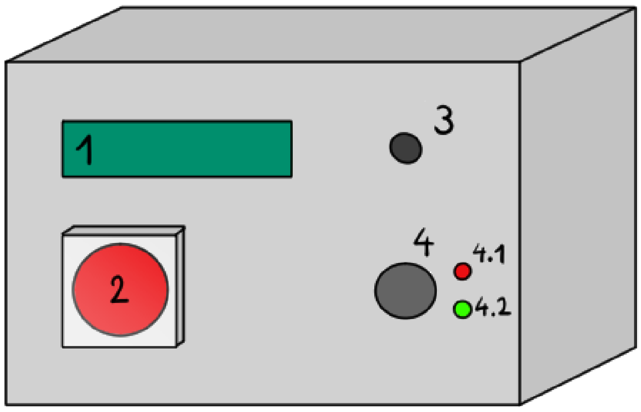


Figure 5.5: OBUI concept design 1

Functionality

Selecting Presets

- The LCD screen (1) displays the selected preset.
- To choose a preset, simply scroll through the options using the rotary encoder (3).
- After choosing a preset, it can be launched by pressing the auto on/off button (4).
- The status LED (4.1) will turn on, indicating that the robot is in auto mode, while the other LED (4.2) will turn off.

Pausing the Machine

- If the user wants to pause the machine, they can press the rotary encoder (3) once.
- The LED (4.1) will start blinking, indicating that it is paused.

No.	**Concept**	**Description**
1	LCD Screen	• A LCD screen to display the name of the selected preset. • It can show 10-15 characters.
2	Emergency Stop	• Interrupts the power source.
3	Rotary Encoder	• Turning the encoder will scroll through the presets. • If the robot is moving autonomously, pressing it will pause the robot, pressing it again will make it continue.
4	Auto ON/OFF	• A push-button, pressing it after choosing a preset will put the robot in auto mode. • Pressing the button again while in auto mode will put it in manual mode.
4.1	LED	• Red status LED, indicates that the auto mode is on.
4.2	LED	• Green status LED, indicates that manual mode is on.

Table 5.2: OBUI interface - concept 1

- Pressing the rotary encoder (3) again will turn the LED (4.1) back on permanently and the robot will resume its operation.

Emergency stop

- Pressing the button (2) will turn off the machine immediately.

Hardware list

- 1 LCD Screen
- 1 Technical Emergency Button
- 1 Incremental encoder
- 1 LED red
- 1 LED green
- 1 Push-button

Pros & Cons

Pros

- Selected preset can be seen on screen.
- Incremental encoder is easy to use.
- LEDs indicate the current mode.
- Visible in bright sunlight.
- Cost efficient LCD display.

Cons

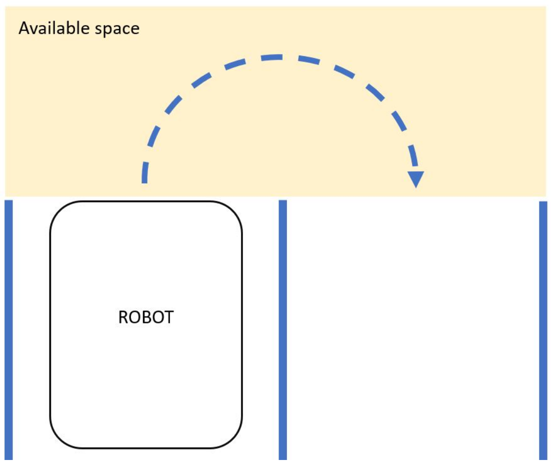
- Only one preset at a time can be seen.
- Eventually time consuming to find preset.

5.3 Row Change

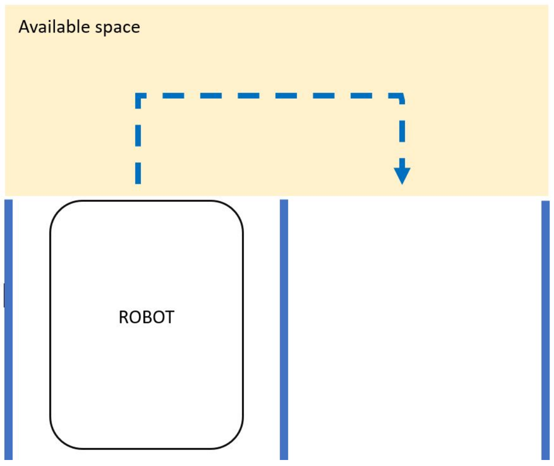
The robot calculates an ideal path from the parameters generated in the PCT. Depending on the available space, the robot chooses one of several path types.

- Continuous curve into the next row.
- Two Ninety-degree turns on the spot.

To increase the amount of available data even further, a stereo camera can be used in conjunction with the odometry.



(a) Simple turn



(b) Right angle turn

Pros

- Distance to objects at different heights can be measured.
- Object recognition included
- If already used for row following, it would be an inefficient use of resources to not use it for the row change too.

Cons

- Even higher cost
- Uses more processing power
- Additional time to get familiar with the technology is required

Used sensors and cost

- Incremental wheel encoders
- IMU
- Stereo camera ZED 2i
- $\rightarrow$ Cost: $\approx$ 500 €

6 Preset Creation Tool

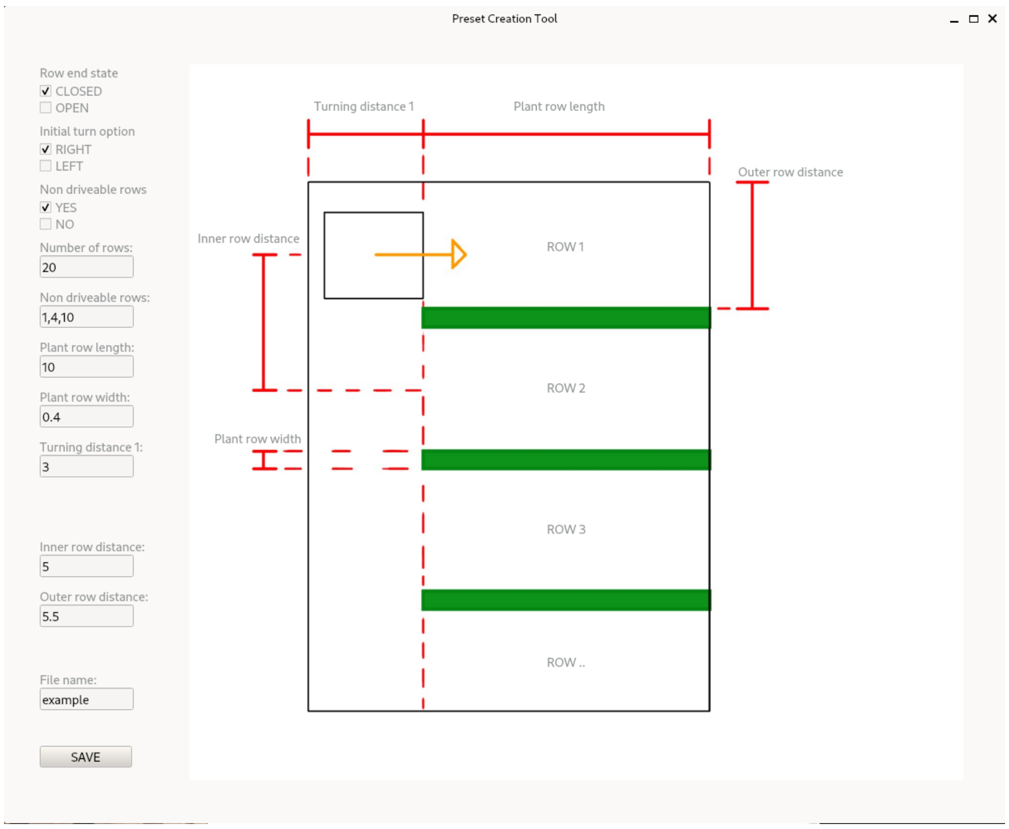
The Preset Creation Tool is a Python-based application designed to assist users in defining the properties of the surroundings they want to pass through with the ADIR. The tool provides a graphical user interface (GUI) using the PyQt6 framework. Following parameters are taken into account.

6.1 Parameters

Row end state	Is the end state of the row, if it is open or closed. Open means, there is an open space on the opposite side of the start, closed means to robot has to drive backwards out of the row in order to turn.
Initial turn option	Is the turning direction at the beginning, if the first turn goes to the right or to the left.
Non driveable rows	Gives information which rows are not able to be driven through by the robot.
Number of rows	Is the number of rows the robot has to pass through before completing its task. This does include the non driveable rows.
Plant row length	Is the length of the plant rows.
Plant row width	Is the width of the plant rows.
Turning distance 1	Is the available turning depth at the beginning of the row.
Turning distance 2	Is the available turning depth at the end of the row.
Inner row distance	Is the distance from one middle of one row to the next. It is thus the space between two plant rows as well.
Outer row distance	Gives information about the distance from the wall to the first plant row.

Table 6.1: PCT parameter list

6.2 Design


Preset Creation Tool

Row end state
[✓ CLOSED] OPEN

Initial turn option
[✓ RIGHT] LEFT

Non driveable rows
[✓ YES] NO

Number of rows: 20

Non driveable rows: 1,4,10

Plant row length: 10

Plant row width: 0.4

Turning distance 1: 3

Inner row distance: 5

Outer row distance: 5.5

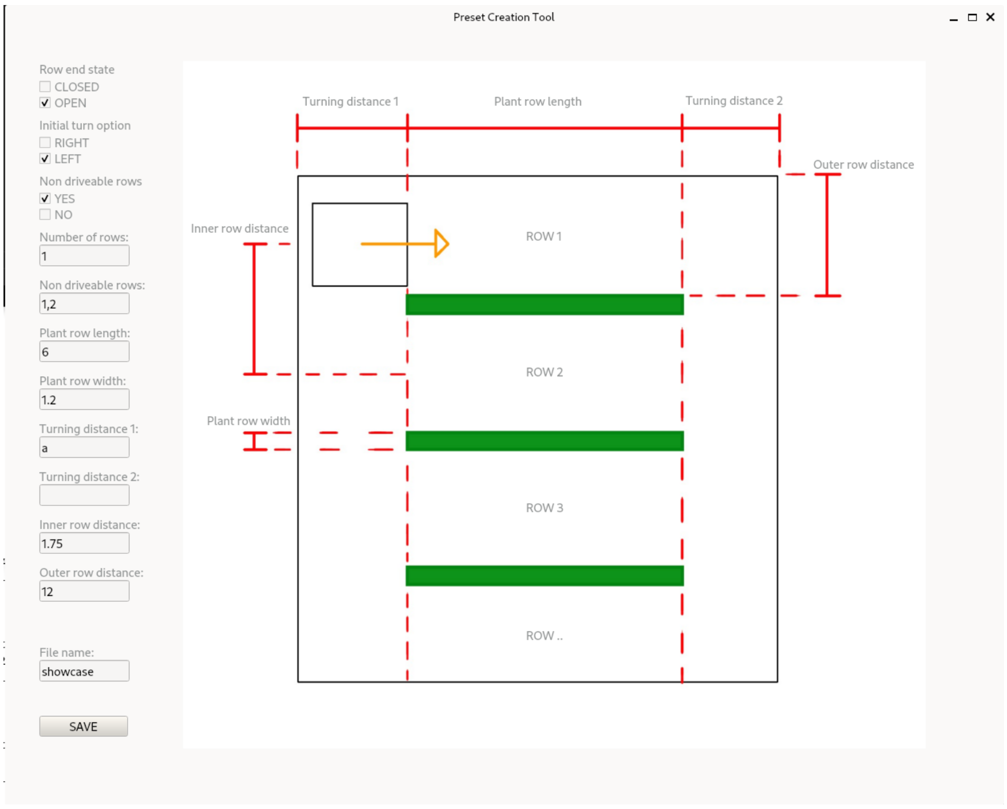
File name: example

Figure 6.1: PCT design with closed rows

Figure 6.1 and 6.2 show x a screenshot of the finished tool in an example configurations.

On the left side, the user is shown all of the available parameters as either check boxes or text input fields. For the undriveable rows, the row numbers are separated by commas. All of the measurements are interpreted in metres.

The image on the right shows a sketch of a field of plant rows. It updates its look depending on the selection of the row end state checkbox. Every time one of the options is checked, the corresponding sketch is shown. To aid the user in finding the correct parameters, all of the measurements are drawn as well.


Preset Creation Tool

Row end state
☐ CLOSED
☑ OPEN

Initial turn option
☐ RIGHT
☑ LEFT

Non driveable rows
☑ YES
☐ NO

Number of rows: 1

Non driveable rows: 1,2

Plant row length: 6

Plant row width: 1.2

Turning distance 1: a

Turning distance 2:

Inner row distance: 1.75

Outer row distance: 12

File name: showcase

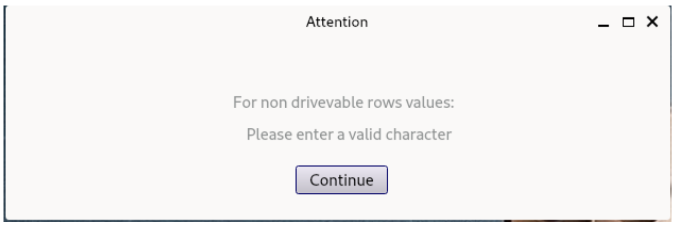
Figure 6.2: PCT design with open rows

6.2.1 Error handling

Upon saving the user input will get validated. The three following error catching instances are implemented:

- The user enters an invalid character
- The user enters a non valid datatype
- The user does not choose a checkbox

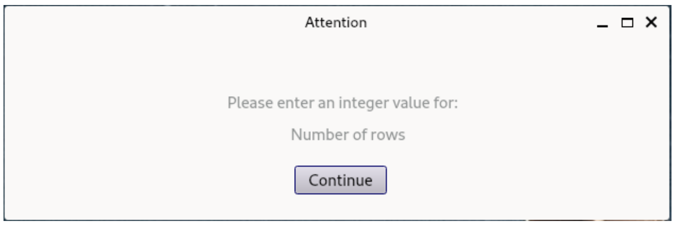
A minor fourth error catching is implemented to beware the user of entering accidentally values that are too high or too low. The value will be set to either the minimum or maximum value. Those three error instances are enough to catch every user error and guarantees the generation of a valid YAML file.


Attention

For non drivevable rows values:

Please enter a valid character

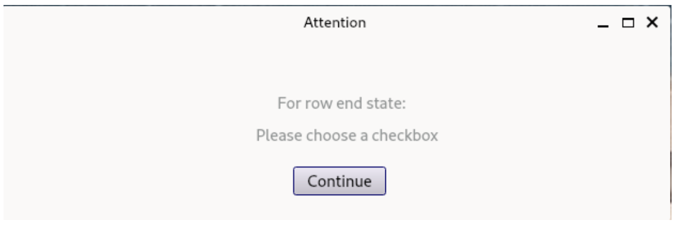
(a) character check


Attention

Please enter an integer value for:

Number of rows

(b) datatype check


Attention

For row end state:

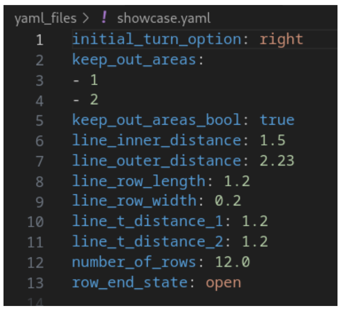
Please choose a checkbox

(c) checkbox check

Figure 6.3: Error windows for all of the possible unusable user inputs

6.2.2 Usage

Upon launching the application, users are presented with an graphical user interface. Users can interact with checkboxes and input fields to define various parameters such as row distances, widths, lengths, and configuration options. Checkbox interactions allow to choose turning options, row end states (open or closed), and driveable row configurations. Upon entering the desired settings, users can save their configurations as YAML files as seen in Figure 6.4



```yaml
initial_turn_option: right
keep_out_areas:
- 1
- 2
keep_out_areas_bool: true
line_inner_distance: 1.5
line_outer_distance: 2.23
line_row_length: 1.2
line_row_width: 0.2
line_t_distance_1: 1.2
line_t_distance_2: 1.2
number_of_rows: 12.0
row_end_state: open
```

Figure 6.4: Generated YAML file

6.2.3 Functionality

Most of the interactions with the tool take place in the main window. When saving the file, the tool will start to check for any incorrect inputs. First the checkbox values will be looked at and handled accordingly, to see whether they are are any missing user inputs. The text inputs are handled separately with different criteria for errors for each of the input fields. When the values pass all of the checks, they will be combined and exported as a YAML file. Figure 6.5 shows, in which window each of the actions happen.

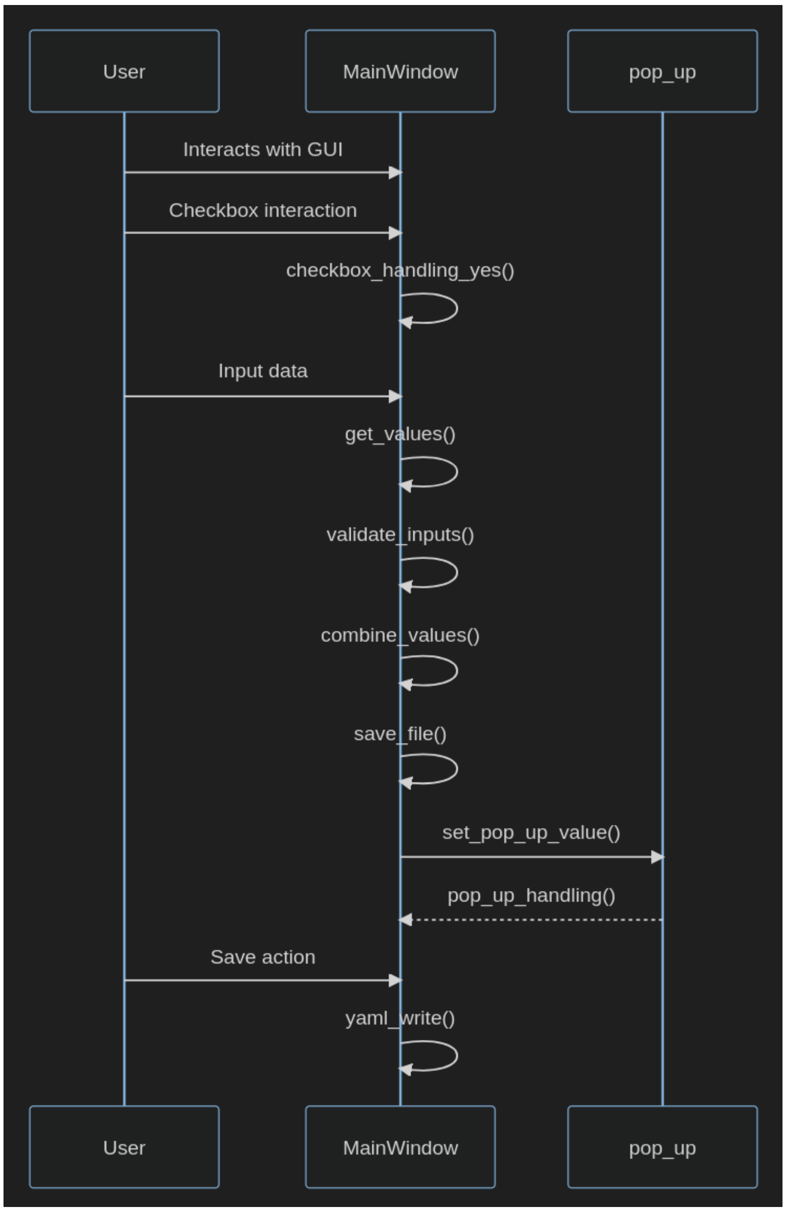


Figure 6.5: Sequence diagram

7 On-board-UI

The On-Board UI, henceforth OBUI, is used for easy control of the robots presets in the field by potential customers.

7.1 Construction

The OBUI consists of the following parts:

- 2 red LEDs
- Incremental encoder
- Push button
- LCD display
- Arduino Uno R4 minima

Since the ADIR was already equipped with an emergency button as seen in Figure 8.1, the emergency button on the OBUI will be left out.

The LEDs are both the same colour, because the shipment of the green ones got cancelled.

To house all of the parts in one unit, a simple panel was designed with mounting space for all of the components. Afterwards, the panel was extended into a box with an additional backplate to protect the electronics from the environments. Figure 7.1 shows the construction in Fusion360.

7.2 Electronics

The following sections dissects the schematic of the OBUI. To create it, the open-source software KiCad was used.

7.2.1 Arduino

The electronics consist of an Arduino Uno R4 minima as the centerpiece, see Figure 7.2. All other components are connected directly to it and draw their power from there as well.

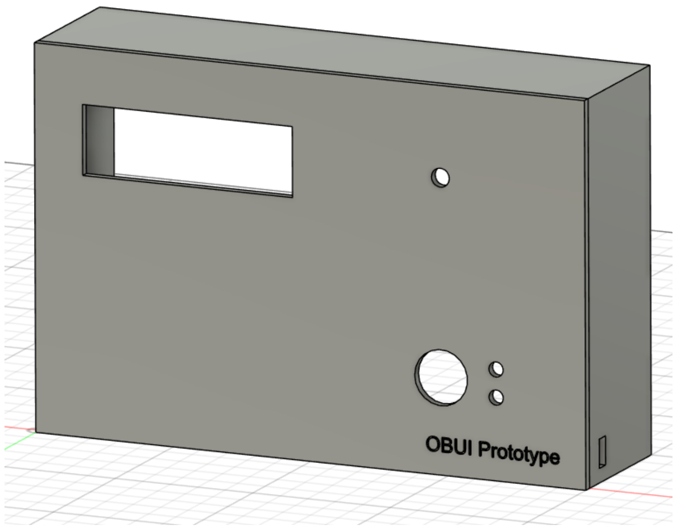


Figure 7.1: OBUI case in CAD

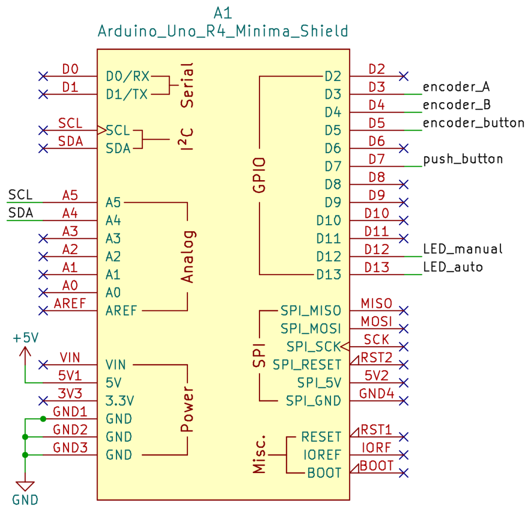


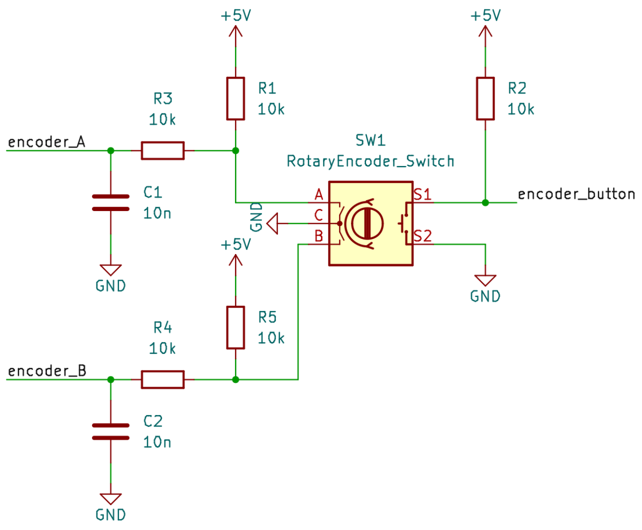
Figure 7.2: Schematic part containing the Arduino and all of its connections

The choice fell on this particular model, since it is relatively cost-effective compared to other official Arduino offerings. Additionally, it offers greater performance compared to most Arduinos and supports the CAN protocol natively.

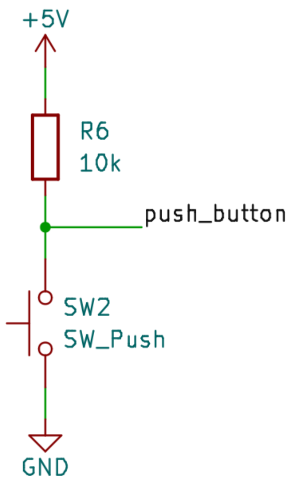
With this approach it is possible to power the whole OBUI through USB or an external 6 V to 24 V supply. This leaves many approaches open to connect the assembly to the rest of the robot.

7.2.2 Inputs

The input options consist of an incremental encoder with built-in push button, as well as a separate push button.



(a) Encoder with recommended filter circuit



(b) Push button

Figure 7.3: OBUI inputs

Figure 7.3a shows, that the encoder has been implemented with the recommended filter circuit according to the datasheet [14]. The push button on the encoder, as well as the separate one (7.3b) get a pull up resistor.

7.2.3 Outputs

To be able to see the currently selected preset, the choice fell on a simple LCD display [8]. It consists of two rows, which are able to display 16 characters each. Communication happens over I^{2}C. Figure 7.4 shows the schematic part as well as the necessary pull-up resistors.

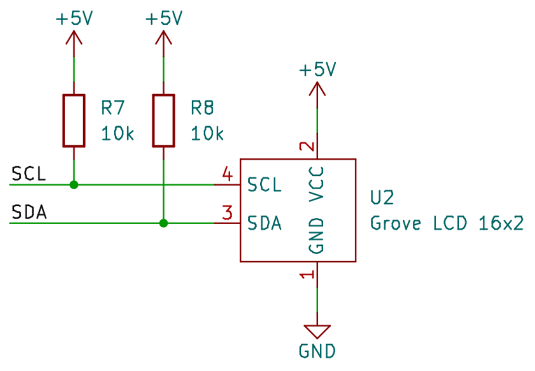


Figure 7.4: Schematic part containing the LCD display and the corresponding I^{2}C pull-up Resistors

Two additional LEDs show the current mode of the robot at a quick glance. They are driven with a simple transistor amplifier, calculated according to the datasheets from the LEDs [7] and the transistors [2]. Figure 7.5 shows the completed circuit.

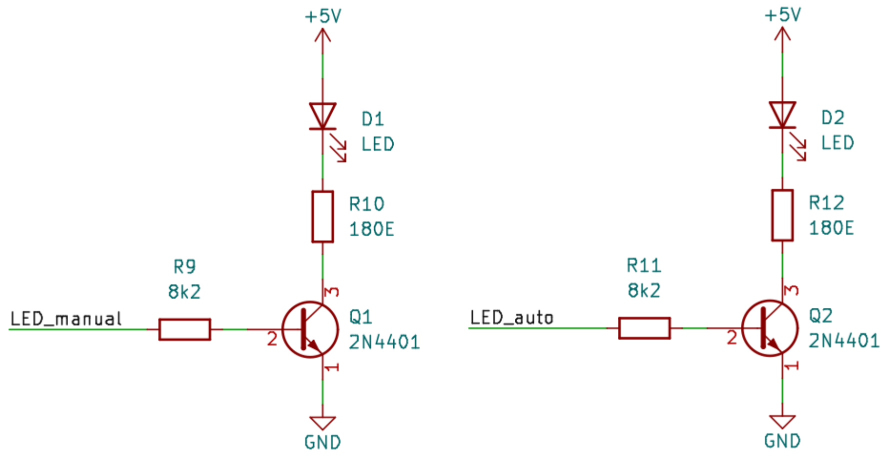


Figure 7.5: Schematic part containing the two LEDs, run by a simple transistor driver

7.3 Software

For quicker development, pre-existing libraries were used whenever possible. Debouncing the push buttons, reading the encoder and controlling the LCD are all solved via external libraries.

7.3.1 State machine

The OBUI software consists of a simple state-machine, consisting of four different states. One for selecting a preset, a second one when the preset is running, a third one when the preset is paused and the final one for pausing the program while running. Figure 7.6 shows the state event diagram of the program.

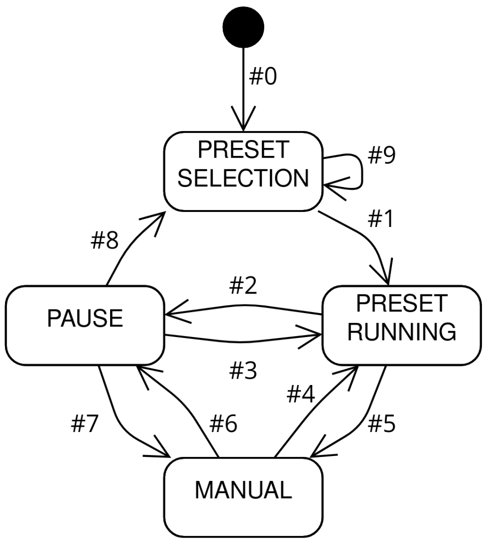


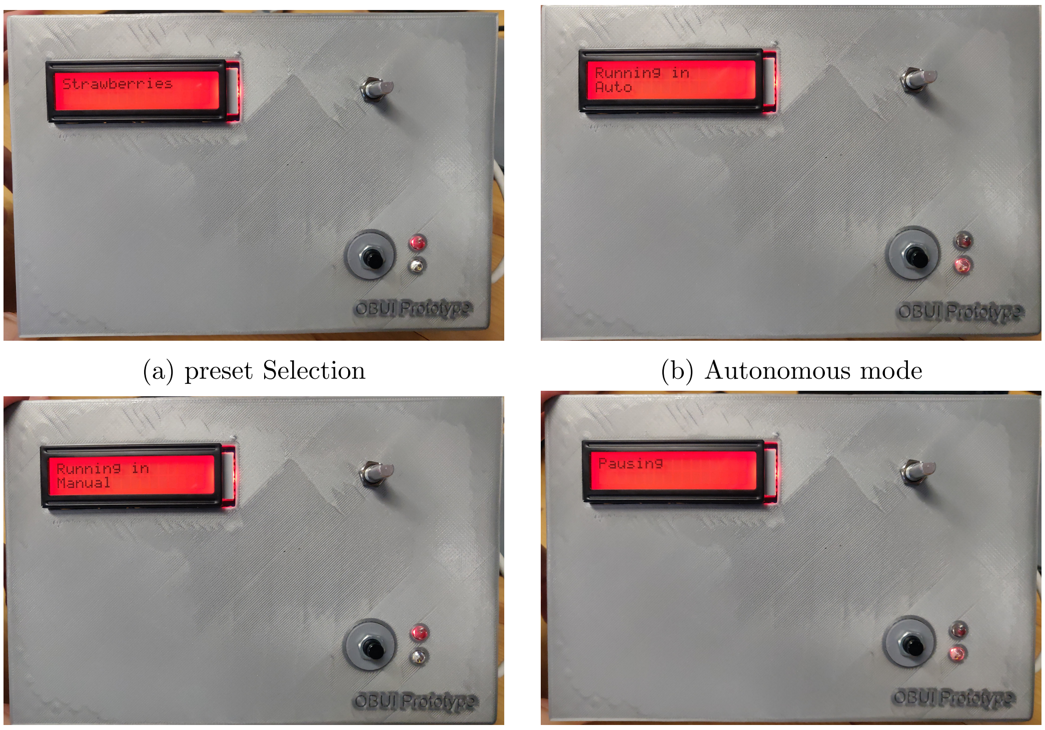
Figure 7.6: State event diagram of the OBUI

When the program starts, the user is able to use the encoder as a scroll wheel and select a preset with the push button button. After that, the autonomous mode starts automatically. The LED responsible for auto mode lights up.

During autonomous operation, the push button can be pressed in order to put the robot into manual mode. The other LED will turn on and a corresponding message is shown on screen. Another push will return the auto mode.

At any time, the incremental encoder button can be pressed, which sends a pause signal to the robot. The lower LED starts blinking. When the encoder button is pressed again, the program will be stopped and the user is greeted with the preset selection screen again. Pressing the push button instead will return the program to its previous state.

Figure 7.7 shows all the different screens during operation. Also visible is the corresponding LED, as described above.



(a) preset Selection (b) Autonomous mode

(c) Manual mode (d) Paused

Figure 7.7: Screens of all the different states

7.3.2 Robot communication

Since the communication to the robot via CAN bus is not in the scope of this project, placeholders were put in the source code. For every necessary message there is a simple function declaration. The functions are already called in the state machine wherever needed.

Example of such a function based on the pause function:

```
void can_pauseProgram()
{
  // Send pause over CAN bus
}
```

With a corresponding call in the state machine during manual mode:

```
else if(encoder_button.falling())
{
    can_pauseProgram();
    ...
    state = PAUSE;
}
```

8 Row Change

The end goal of the application is a system, which is able to spray rows of plants completely on its own. Our part of this project was implementing an autonomous row change algorithm.

8.1 Platform overview

The working platform we were provided with is the ADIR, a mobile robot from ANT ROBOTICS, as seen in Figure 8.1. It has a massive steel chassis which can be equipped with different sensors. On the front of the robot is an emergency stop- and a general use push-button. In a later revision, the emergency stop is connected to a safety bumper in the front. There is enough space on either side to mount a double wheel configuration for additional traction. Our configuration came with single wheels.

Movement type	Differential drive
Number of Wheels	4
Weight	160 kg
Payload	400 kg
Width	100 cm
Length	110 cm
Height	60 cm
Number of motors	2
Voltage supply	Dual 25.8 V batteries
Top speed	$4\,\frac{\text{km}}{\text{h}}$
Interfaces	CAN
On board controls	Push button and emergency button

Table 8.1: ADIR specifications

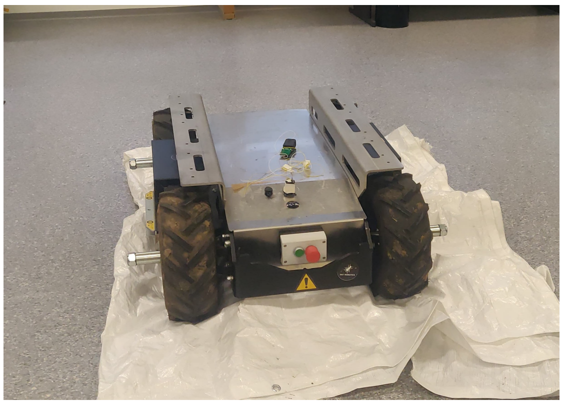


Figure 8.1: ADIR mobile platform

8.2 ADIR hardware setup

Since testing the robot remotely was not possible in its current base state as seen in Figure 8.1, a solution which allows us to work remotely on the ADIR had to be found. The remote connection for working on the robot is necessary, because being right next to it, while testing was not an option, since its weight of 160kg and sudden movement posed a high risk for accidents. For the scope of this project setting up the ADIR for remote work was not part of it, so a makeshift solution had to be made up. We had the following options available at the time:

- Remote desktop connection with XRDP
- Remote desktop connection with VNC
- Remote desktop connection with SSH
- Video transmitter and receiver

Establishing a direct connection via a remote desktop, connection via XRDP was not possible on the NVIDIA Jetson. VNC did work but required a display plug at the NVIDIA Jetson at all times. Connection via SSH worked but allowed for terminal only usage which did not allow us to monitor the ADIR in RVIZ. With an additional -X flag, windows could be transmitted as well, but performance was incredibly lacking.

Using a display transmitter and receiver was left as the last option which we then opted for after we were able to use hardware, which was available at the school. For mouse and keyboard input we were using a mouse and keyboard setup which used a wireless transmitter allowing us to work up to 8m remotely on the ADIR. Table 8.2 shows a precise overview of all the different components.

Computation	NVIDIA Jetson Orin Nano
Power supply regulation	Traco power DC/DC Converter to 12 V
Power supply plug	Designed and 3D Printed
Video transmitter	Speaka professional
Mouse and keyboard transmitter	TRUST Transmitter, included with wireless mouse and keyboard
LIDAR	YDLIDAR TG15
Stereo camera	ZED 2i
Wheel encoder	ADIR built in encoders

Table 8.2: ADIR on board hardware

In order to power all the necessary hardware, the voltage from the batteries was converted to a usable 12 V via a DC-DC converter.

To be able to connect everything in time, a power plug was designed and 3D printed as seen in Figure 8.2 and 8.3 instead of waiting for a supplier.

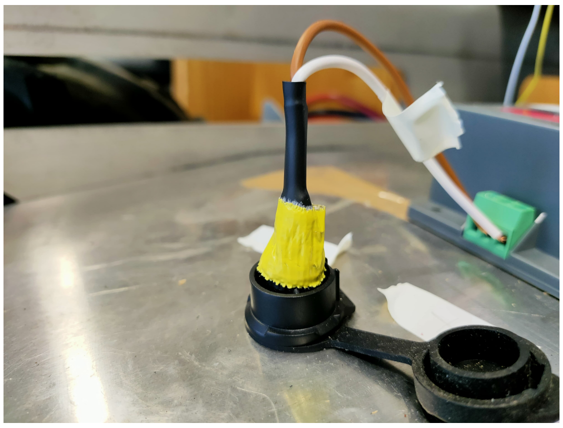


Figure 8.2: 3D printed plug

The plug connected the batteries to the DC-DC, which in turn supplied the NVIDIA Jetson module with its high input voltage range. All of the other components are powered through the USB connections. Figure 8.4 shows the complete connection diagram, Figure 8.5 shows what it looked like when connected. Note that the LIDAR is not connected, since it was only used occasionally.

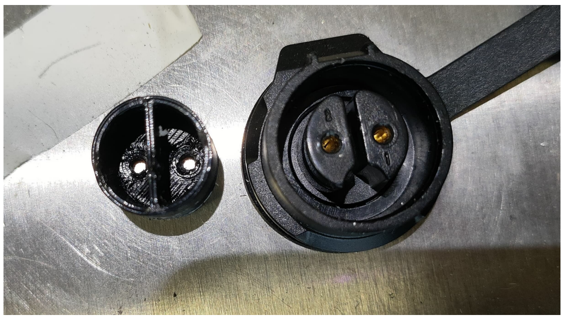


Figure 8.3: Plug next to ADIR socket

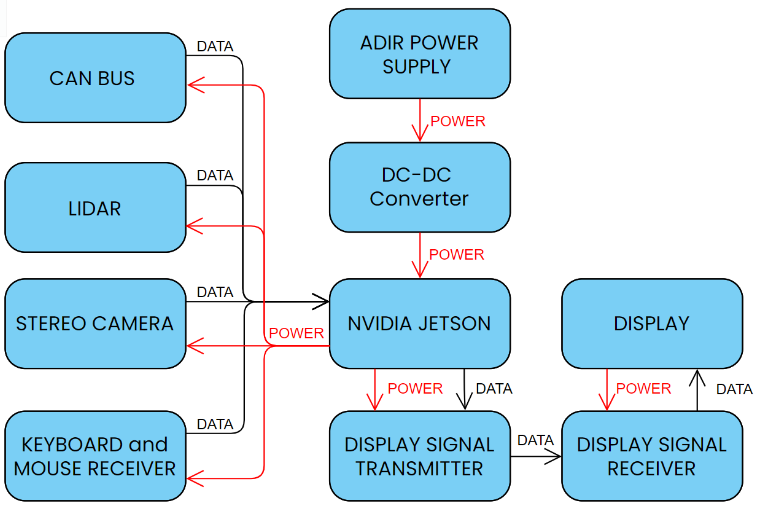


Figure 8.4: Connection diagram

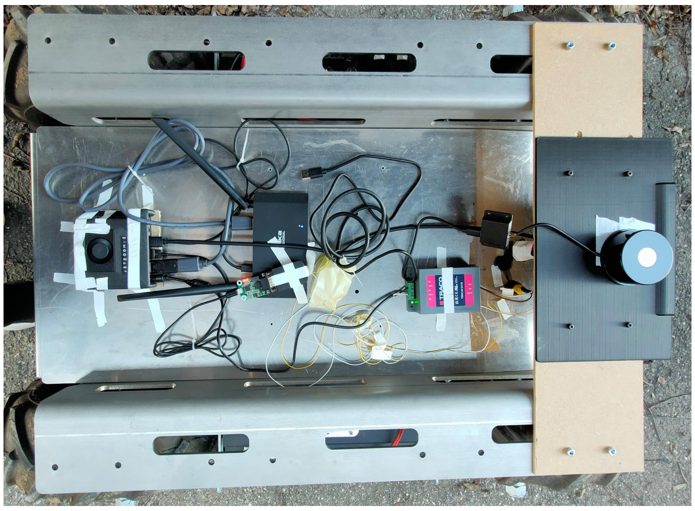


Figure 8.5: ADIR hardware setup

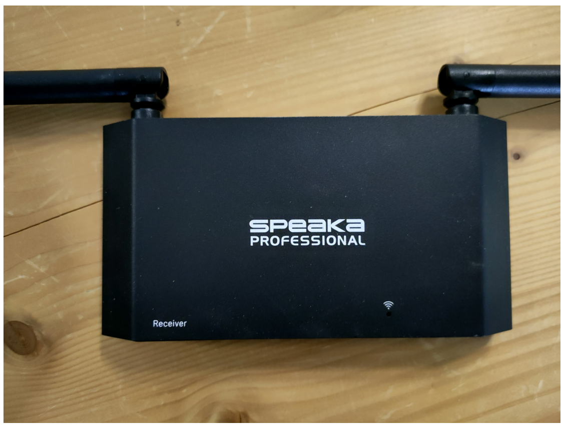


Figure 8.6: video receiver

8.3 ADIR software setup

8.3.1 Robot description

In order to be able to do anything relevant for this project, including setting up a simulation environment, the first step in development was to create a virtual model of the robot and all of its components.

For that, a so-called Unified Robotics Description Format (URDF) file is used. It contains information about the different parts of a robot and how they are connected. For example, where the camera is positioned in relation to the center of the robot.

Robot model

In order to create the URDF file for our purposes, a 3D model of the robot, provided by ANT ROBOTICS, was used to create it automatically. For this purpose, a plugin [10] for the popular CAD modeling software Fusion360 was used. Only a little bit of preparation was needed on our end, to set the origin of all the parts according to the URDF specifications. This consists partly of setting the rotational axis for the wheels and setting the viewing direction for the camera.

Figure 8.7 shows the view in Fusion360 with all of the axis. Note that one of them is rotated slightly upward. This is because earlier in the project, an upwards angled LIDAR was used in order to detect objects higher up than the mounting spot on the robot. This approach was later scrapped.

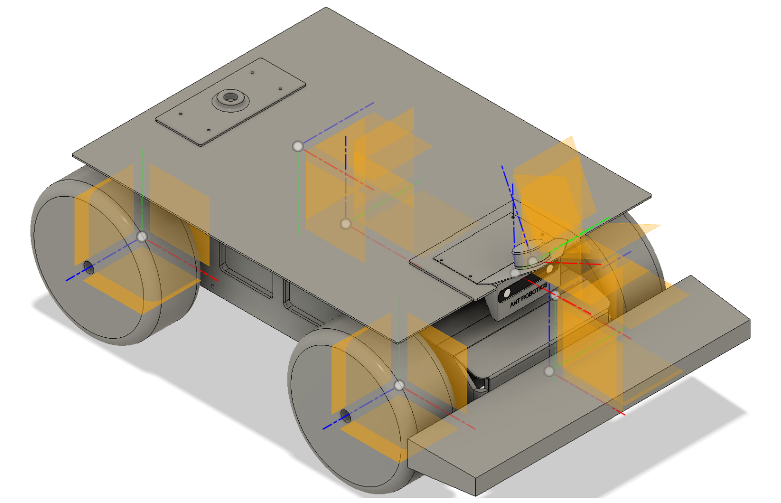


Figure 8.7: ADIR in Fusion360 with all the different origins visible. Red corresponds to X-, Green to Y and blue to the Z-axis

This URDF file was then expanded to include additional plugins for simulation purposes with Gazebo. More on that can be found in Section 8.4.

8.3.2 Map converter

Overview

An earlier approach was to write a map converter. The general concept was to generate a map from the preset values. The map is generated in the ”map converter” Afterwards the map is given to the robot.

Map Converter

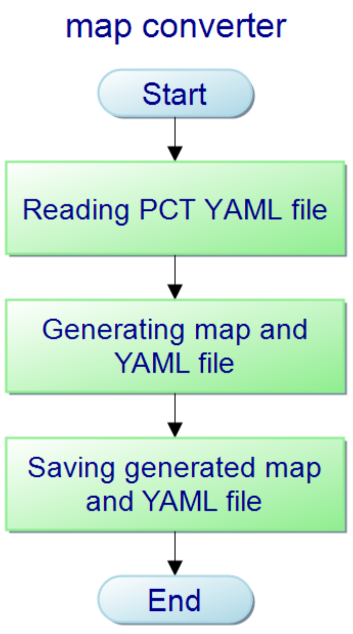

map converter
Start
Reading PCT YAML file
Generating map and YAML file
Saving generated map and YAML file
End

Figure 8.8: Flow diagram map converter

The task of the map converter lies in reading the generated YAML file from the preset-creation-tool and generating a map according to the YAML file parameters. The generated map is an occupancy grid. After generation the map, it was supposed to be loaded into the map server. In the future it could be implemented into the PCT. The user would be able to see directly with what kind of map he would be working.

8.3.3 Runtime overview

As seen in Figure 8.9, the initial stage happens independently from the rest of the program. The user generates a preset with our preset creation tool and loads the corresponding YAML files on the robot.

When the robot starts its autonomous operation, the aforementioned preset file gets loaded as individual constants into the ROS parameter server. These values will later be used to calculate the correct path to the next row. Additionally, a ROS node gets launched, which will return a row change path upon request.

When this node receives the message, that a row change should be executed, it takes the parameters from the server, obtains the current robot position and calculates a smooth path around the rows. This path then gets published as a series of waypoints as well as a continuous path. Specifically with a series of geometry_msgs/ PoseWithCovarianceStamped messages or a single nav_msgs/ Path message respectively.

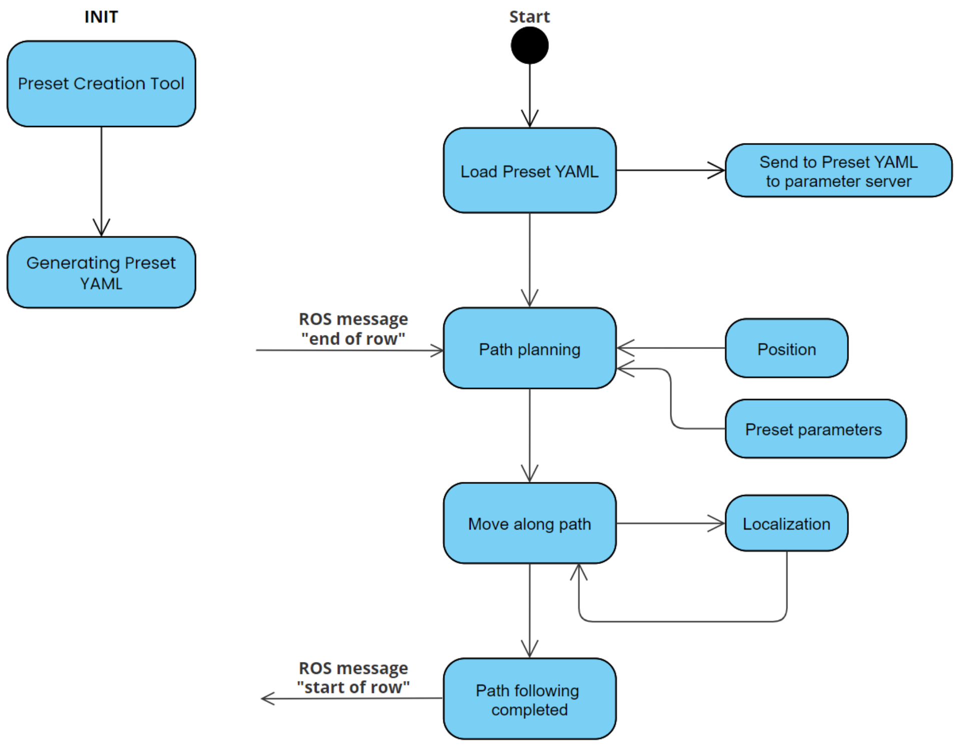


Figure 8.9: Row change program overview

For our project the following packages from the respective sources were used:

- Ant Robotics
  - adir_control
  - ros_can
- Open source ros packages

  - move_base [22]
  - rtab_map [9]
  - robot_localization [23]
  - follow_waypoints [24]
  - zed-ros-wrapper [29]
- Our packages
  - row_change_trajectory
  - map_converter

As seen in 8.10 the green colored box is the package we have written, containing the row change trajectory. From the row change trajectory the directional vectors are published as set of waypoints to the package follow_waypoints. They will afterwards be sent as individual waypoints to the move_base, which in turn sends velocity commands to each waypoint.

The yellow colored boxes are the code Ant Robotics provided us. The blue colored enclosures are open source ROS code. Sensor data is gathered from the wheel encoder, and the stereo camera, for the stereo camera the ros package zed-ros-wrapper is used. From the stero camera we get VO and IMU data. For generating the map, we use the RTAB-Map package which gets the point cloud from the stereo camera. The different transforms (tf) are used to tell the nodes and packages the physical relation to each other.

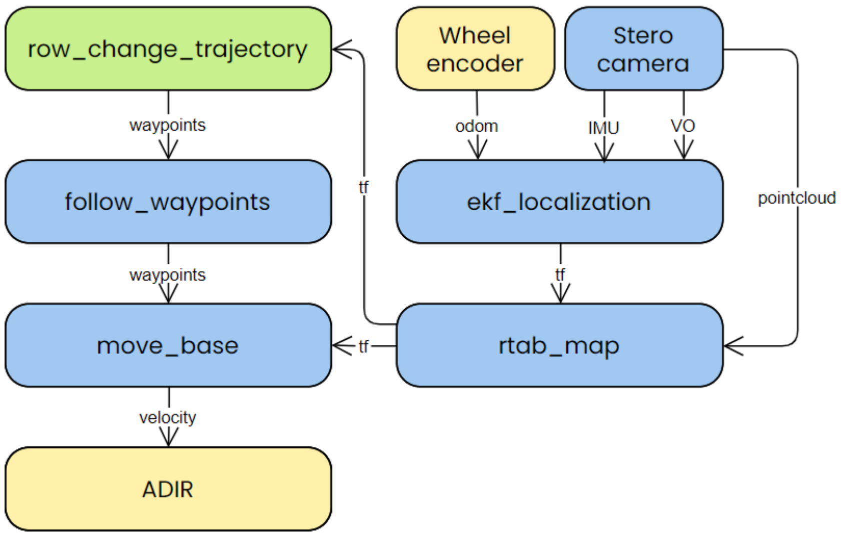


Figure 8.10: Ros packages connection

For a full overview of all topics and nodes and their corresponding connection see Attachment 6.

8.3.4 Sensor fusion

In order to increase the accuracy of the odometry, multiple sensor sources are fused together with an extended Kalman filter (ekf) approach. In this case there are three odometry sources:

- Wheel odometry coming from the robot platform. Returns current pose estimate as well as translational and rotational speed.
- IMU, integrated in the stereo camera. Returns rotational velocity and translational acceleration.
- Visual odometry, calculated by the stereo camera software. Returns pose estimates.

The first package tested for sensor fusion was robot_pose_ekf [28]. It natively supports all three of the above-mentioned sources. Problem was, that the ZED software does not publish a velocity estimate as part of its odometry, which made it not usable for the real-life ADIR. The drift was simply too high after a short while. In simulation however, without visual odometry, it still works reliably.

Since the pose estimate with robot_pose_ekf was not accurate enough for the row change application, another package was tested. The ROS package robot_localization [26], based on the research by T. Moore and D. Stouch [4], is an ideal choice for this scenario. It is possible to add any number of odometry and IMU inputs. Additionally, it is possible to select, which of the 15-dimensional states should be taken into account from which source.

The fifteen dimensions are:

- Pose: X, Y, Z
- Rotation: $roll$, $pitch$, yaw
- Translational velocity: $\dot{X}$, $\dot{Y}$, $\dot{Z}$
- Angular velocity: $\dot{roll}$, $\dot{pitch}$, $\dot{yaw}$
- Translational acceleration: $\ddot{X}$, $\ddot{Y}$, $\ddot{Z}$

We selected the following dimensions from each of the inputs:

IMU:	$\ddot{X}$ and $\dot{yaw}$
Wheel odometry:	$\dot{X}$, $\dot{Y}$ and $\dot{yaw}$
Visual odometry:	X, Y and yaw

$\dot{Y}$ was not included in the IMU, since this increases the drift in Y-direction. The rest of the rotational axis are also not included, since the robot only moves in a two-dimensional plane.

Any pose estimations were excluded from the wheel odometry, because it did not proof accurate enough in testing.

Concerning the visual odometry, all the available information was used.

8.3.5 Localization

The localization is done by the open-source ROS package rtabmap_slam [9]. Real-Time Appearance-Based Mapping (RTAB-Map) is a SLAM algorithm designed to map out a new area based on depth camera, point cloud or LIDAR information. In this application, rtabmap_slam is used, which is a ROS implementation to be used in 3DOF situations.

Initially, a ROS package with an adaptive Monte Carlo Localization (amcl) [18] was used. It was easier to use, since it is a part of the ROS navigation stack, which we used too. But in order to use it, the point cloud coming from our stereo camera had the be converted into a single, two-dimensional laser scan and thus loosing information. Additionally, it was using more computational power compared to rtabmap_slam.

Using rtabmap posed an additional advantage over amcl. Since we are able to take height of the obstacles into account, the robot is able to drive under obstacles where there is enough space. This is very much beneficial, when the robot is able to execute the row change in a tighter space when it knows that it is able to drive underneath the planters or other obstacles as well.

Important for this application is also the fact, that rtabmap_slam is able to do simultaneous localization and mapping (SLAM), instead of only localization. The robot does not have have a pre-generated map of the area but must instead rely on its current measurements of the surroundings.

8.3.6 Navigation

For navigation, a collection of ROS packages called navigation_stack [23] was used. Centered around move_base[22], it contains packages for localization, mapping and path planning.

Move_base is used to let an autonomous robot drive to a goal located on a map. If only the parts from navigation_stack were used to navigate to the next row, the taken path could differ every time and the chance of unwanted movements is too high. With an additional package, follow_waypoints[24], we can publish multiple goals one after another. This allows us to loosely follow a pre-calculated path.

Within the navigation_stack, there are multiple options concerning planners. They are divided into global and local planners.

Global planners are responsible for finding the most efficient way in a map to the goal. This gets published as the global_path for the local path to follow.

Local planners are confined to the immediate surroundings of the robot. They watch out for smaller obstacles along the way and try to control the robot, so it does not drift too far away from the path generated by the global planner.

Move_base implements different planners as plugins, which can be separately specified.

Global planner

Using the navigation_stack provides three different global planners.

- global_planner
- navfn
- carrot_planner

For our approach, we have been using the global_planner[20]. It is a more flexible version of the navfn planner, which is the standard in the navigation_stack. More parameters can be adjusted for a better result of the global path.

Since we are sometimes driving backwards out of the row, and the sensors are only attached at the front, it is important to have a planner that allows for planning in unknown space. Even when driving forwards around to the next row, the stereo camera is not able to take all the necessary information in at once and thus creates temporarily unknown spaces. The other choices do not support such a behaviour.

Local Planner

The following local planners have been tried:

- base_local_planner
- dwa_local_planner
- teb_local_planner

The first tried local planner was the base_local_planner [22], since it is the default choice when using move_base. For this use case, this plugin could not be used, since only forward motion is supported, which would only allow for the forward row change.

The dwa_local_planner[27], based on the work by Fox D, Burgard W and Thrun S [1], was tried out next. Dwa is short for dynamic-window-approach. What made it attractive was an extensive tuning guide, written primarily for this plugin [5].

In the simulation as well as in the field, the dwa_planner proved to be quite reliable in the forwards row change. It is not as smooth as hoped for, since it always stops at a waypoint before going to the next one. This includes occasional turns on the spot in order to match the angle given by the current goal. Backwards row changes were sadly still not possible. The planner never realized, that driving in reverse was a viable option, even though all the parameters were set to allow this type of movement.

The last tested option was the teb_local_planner[25], based on the time-elastic-band approach [3] and additional work in trajectory optimization [6]. The many available parameters allow for a well-tuned system for a specific application.

The teb_local_planner is also able to follow a path directly, instead of relying on goals alone, which would allow the robot to follow the path to next row much more precisely.

8.4 Simulation setup

Since it is not easy to spontaneously test our algorithm anywhere on the university perimeter, a simulation environment was created.

The simulation software of choice was Gazebo, the widest spread simulator for ROS applications. We used Gazebo-classic, since this is the version that ships with our ROS version, noetic.

For the robot itself, the URDF file also contains references to the individual 3D modelled parts, which allows for a very precise footprint for the simulation. Figure 8.11 shows what the model looks like when loaded into gazebo. Note the slightly floating LIDAR. It is slightly raised, so that the mounting bracket does not obstruct the view when experimenting with different upward angles without having to change the robot model every single time.

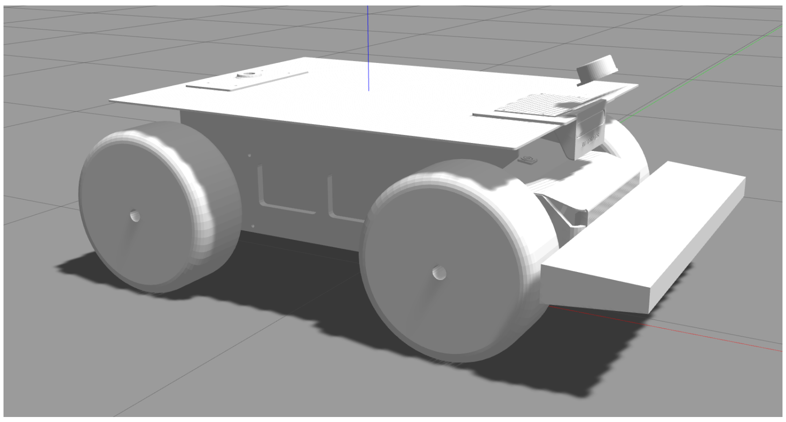


Figure 8.11: ADIR loaded into gazebo

8.4.1 Gazebo plugins

Included in the previously mentioned URDF file for the robot description are also a file for sensor and one for movement plugins. To find and configure them, the official Gazebo tutorial on this topic was used [19].

Sensors

Concerning the IMU, an imu_plugin was used for simulation.

For the LIDAR on top of the robot, a simple setup consisting of a laser_controller plugin and a ray type laser sensor was used. Since a YDLIDAR TG15 sensor was provided by Ant Robotics, the parameters for the simulated sensor were set as close to the parameters of the TG15 as possible. The only exception is the number of samples, which was set to 2000 instead of 2700. Reason for that being, that the plugin is not able to compute any higher number of rays than that.

Lastly, the stereo camera was simulated using a plugin, which is designed to emulate the Kinect sensor by Microsoft. Since its release, it is popular choice for low-cost depth camera applications in the robotics space. In our case, the parameters were tweaked to match the ZED 2i stereo camera. In later test it was reveled, that the resolution had to be lowered significantly in order to keep the simulation performance at an acceptable level.

Movement

Since our robot platform ADIR is based on a skid-steer design, the aptly named skid-steer-drive-controller was used. This plugin is not only able to control the robot, but generate wheel odometry as well. To set the correct parameters, the 3D model was used to get the measurements and the ADIR features page gave us the necessary information to calculate the torque [16].

When testing out the simulation, movement was initially very inconsistent and uncontrollable. The simulated robot was sometimes not able to turn at all, at other times it slid along the ground like it was on ice. The cause was later found in the friction parameters defined by the URDF file. $\mathbf{mu}_1$, set in positive X-direction of the robot needed to bet set higher, so that the robot could drive forward and backwards. $\mathbf{mu}_1$, set in positive Y-direction needed to be lowered significantly, which enabled the robot to slide slightly to the side, which in turn enabled a smooth rotation.

8.4.2 Simulation world

To get the simulation as close as possible to a real-life scenario, the plant rows were modeled in CAD. Ant Robotics provided an example of a plant row by Metasa [21] as well as additional measurements from a real farm.

The rows come in hanging and standing form. Both were modelled in Fusion360 in 10 m long segments. Figure 8.12a shows the completed model of a standing row. Figure 8.12b is the hanging row.

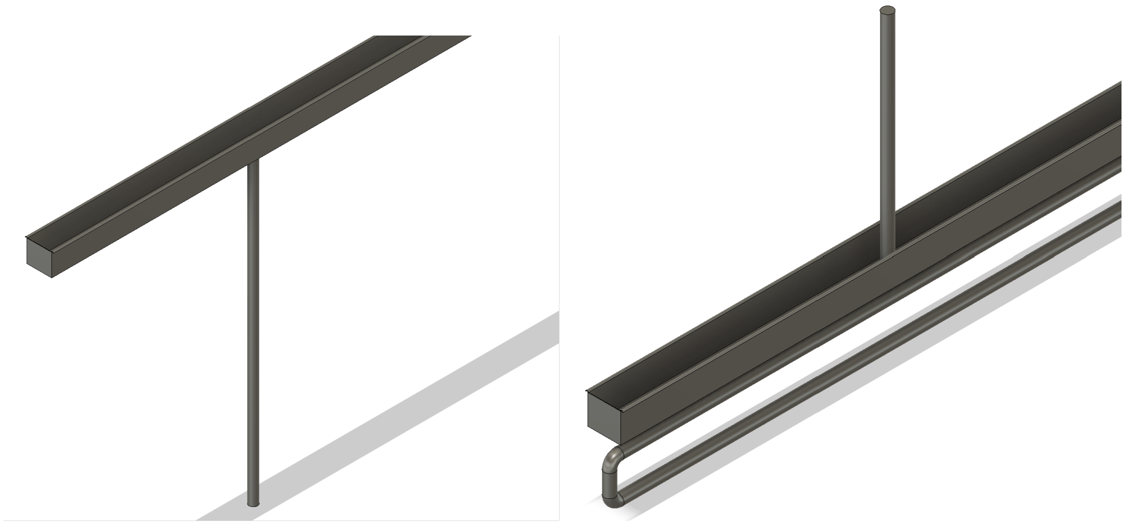

(a) Standing row (b) Hanging row

Figure 8.12: The two types of rows modelled in Fusion360

To create the simulation world, five of the standing rows were put next to each other, with the space according to measurements from the farm. At one end of the rows, there is an available turning space of 2 m, the other end has 2.2 m. The difference is there in order to test the tolerance in our turning algorithm. Figure 8.13 shows the completed Gazebo simulation setup with the robot in its starting position.

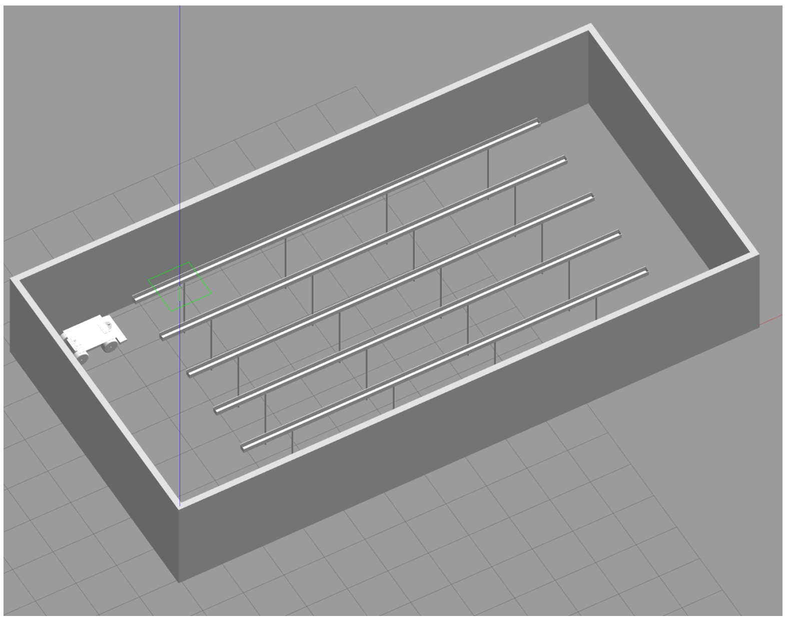


Figure 8.13: Simulation world in Gazebo containing the standing rows

8.5 Row change path

Since the ADIR has to move to the next row in a predetermined space, an approach to make use of the preset parameters is preferred. Using those, an ideal trajectory can be calculated to move around to the next row.

8.5.1 Trajectory function

The function responsible for calculating a smooth path is a segment of a stretched unit circle.

To get as smooth of a turn as possible, the robot should utilise a big part of the available turning space. For this, the following stretching factor in X-direction was chosen:

$$d_1 = \text{turning\_distance} - \frac{\text{ADIR\_width}}{2} - \text{offset}$$

Turning_distance is loaded from the parameter server when the program is running. Subtracting half the ADIR width would result in a curve that strafes the edge of the allowed space very closely. For this purpose, an additional offset is added. This contains an additional distance for reliability, as well as the additional distance lost by driving out from the row.

In order to reach the next row in a single smooth curve, the diameter of the circle has to be equal to the distance between two rows, the row_distance parameter from the preset. Thus, the stretching factor in Y-direction equals:

$$d_2 = \frac{\text{row\_distance}}{2}$$

These parameters can be combined into the following function to describe a half circle:

$$f_{trajectory} = \begin{bmatrix} d_1 \cdot \sin(\gamma) \\ d_2 \cdot \cos(\gamma) \end{bmatrix} \Bigg| \text{ With } \gamma \in [0, \pi]$$

To be able to send the correct direction at every point, the directions of the individual points can be calculated by taking the derrivative of the trajectory function:

$$v_{trajectory} = \dot{f}_{trajectory} = \begin{bmatrix} d_1 \cdot \cos(\gamma) \\ -d_2 \cdot \sin(\gamma) \end{bmatrix}$$

Figure 8.14 shows an example of a such a trajectory. Note: The forward direction of the map and thus the initial direction of the robot is in the direction of positive X.

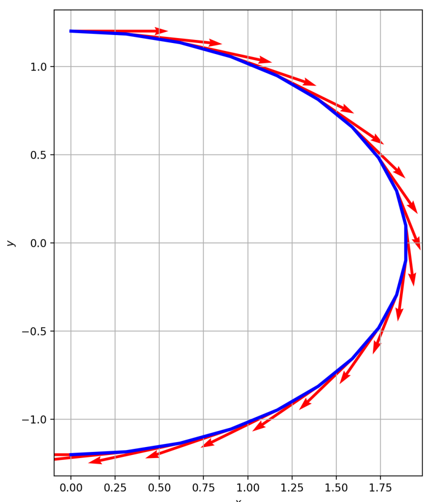


Figure 8.14: Row change trajectory example with row_distance of 2.5 m, calculated in a space of $\gamma \in [0, \pi]$

If the turn should be made in the other direction, the only change to the function is to change its start and end point to:

$$\gamma \in [-\pi, 0]$$

When only a ninety-degree turn is needed, the range of the function gets sliced in half.

8.5.2 Row change trajectory forwards

Looking at the row change forward trajectory, pictured in Figure 8.15, it is split into three parts:

- Driving slightly out of the row for clearance
- Executing the turn into the next row
- Driving into the next row

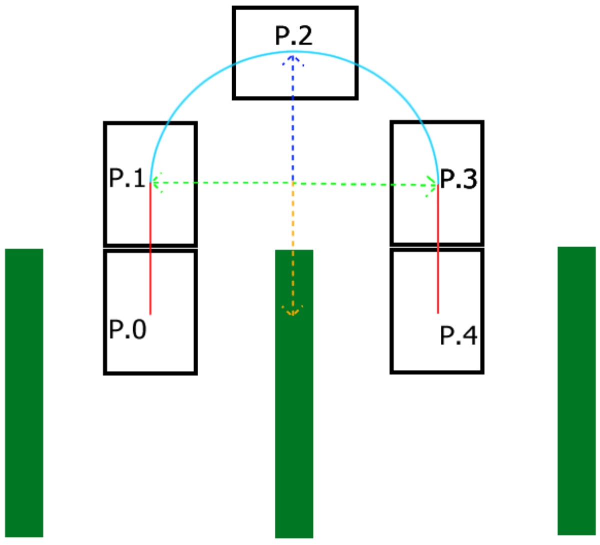


Figure 8.15: row change forward trajectory

Red line	ADIR length
Light blue line	Trajectory function
Dark blue dotted line	d_1
Green dotted line	d_2

Table 8.3: Row change forward trajectory table

The resulting trajectory then consists of the calculated half circle, which is offset by the distance used to drive out of the row.

Figure 8.16 shows an example of a calculated path in Rviz, set in the Gazebo simulation.

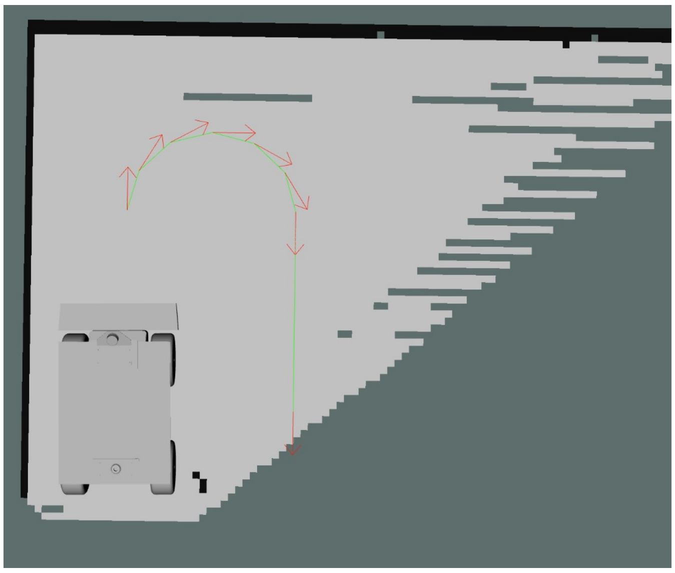


Figure 8.16: Forwards trajectory, as seen in Rviz

8.5.3 Row change trajectory backwards

The trajectory in the backwards row change is made of 5 parts. We got two additional actions compared to the forwards trajectory, reason for that is that the robot should always drive forwards into the next row, since the stereo camera and the emergency button are at the front. The following steps can be seen in Figure 8.17:

- Driving out of the row
- A quarter turn to line up the row change
- Moving to the next row
- Turning towards the row
- Driving into the row

Red line	ADIR length
Blue line	Trajectory function
Purple dotted line	d_2
Blue dotted line	d_1
Orange dotted line	ADIR length

Table 8.4: Row change backward trajectory table

Figure 8.18 shows an example of a calculated path in Rviz, set in the Gazebo simulation.

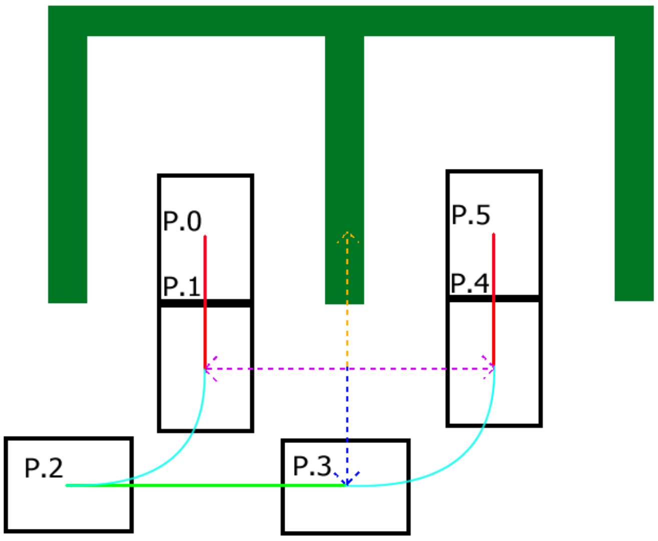


Figure 8.17: row change backwards trajectory

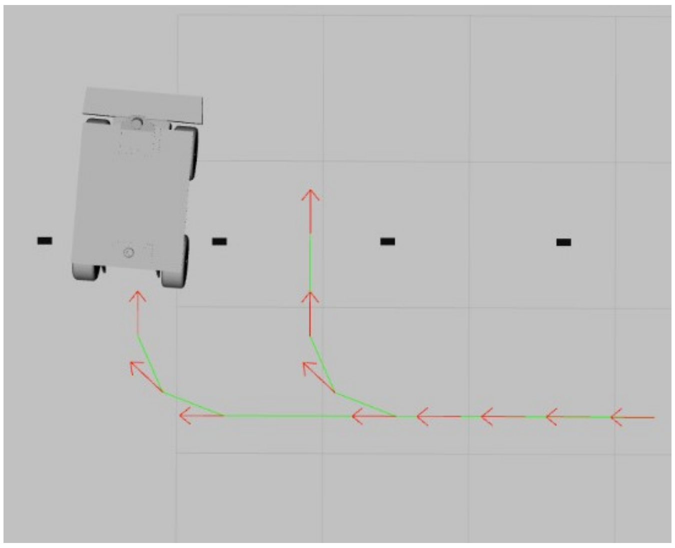


Figure 8.18: Backwards trajectory, as seen in Rviz

Publishing the trajectory

When the correct ROS message is received by the row change trajectory node, it calculates the path based on the parameters in the parameter server. Whether it should change rows forwards or backwards, how much space there is to turn and how big the length of the distance between two rows is.

Additionally, the current position of the robot needs to be taken into account. The path around the rows gets offset by the coordinates and half of a row_distance, in order to start the turn on the curve itself, instead of sitting in the middle of the circle.

If the rows of the current environment are open, the robot needs to alternate between turns in positive and negative X-direction. Depending on whether the turn is to the left or the right, the directions of the individual points needs to be reversed.

After all of the calculations are completed, the waypoints are published as individual poses, as well as a single path. This is to ensure compatibility with different ways of following the path.

9 Testing

9.1 Testing environment

The first testing environment was indoors in the FHGR mobile robotics lab, but testing could not proceed there. Reason for that were the streaks the ADIR left while making turns of any radius.

For further testing, it was necessary to find another location, where the condition of the floor does not matter. In the end, permission was received to use an abandoned industrial zone next to the university. On the premise, there is a shed which granted us a cold, yet weatherproof testing environment.

The testing environment is a enclosed space with a approximately 30 square meters area with natural lighting through a window front as seen in figure 9.1. An additional light source is made up of a single LED tube which omits light from the opposite site of the windows.

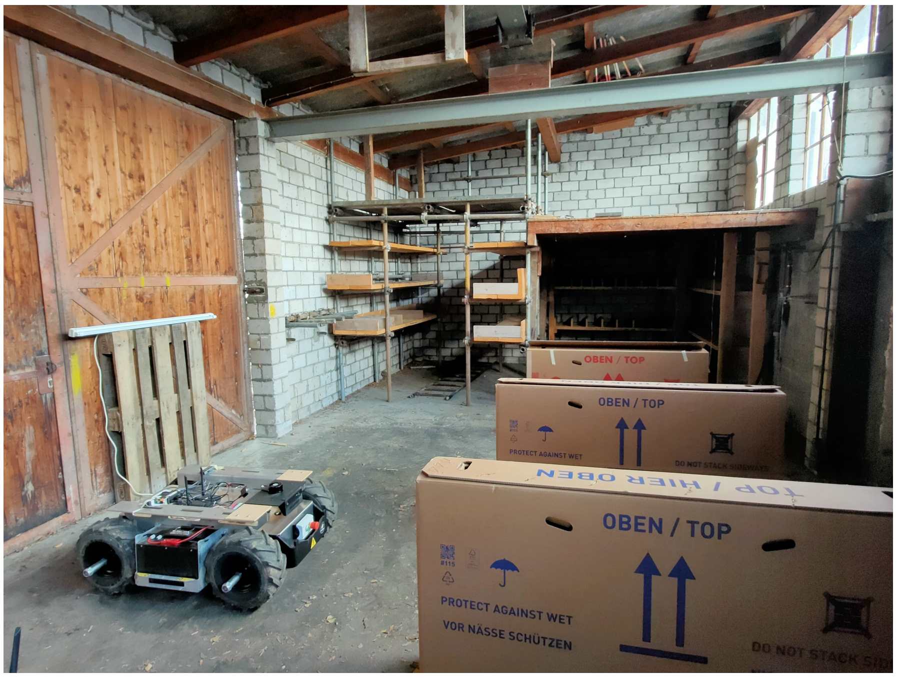


Figure 9.1: Testing environment with natural light

A screen for remote control was set up on a table in a distance of approximately 150 cm to the first row as seen in Figure 9.2.

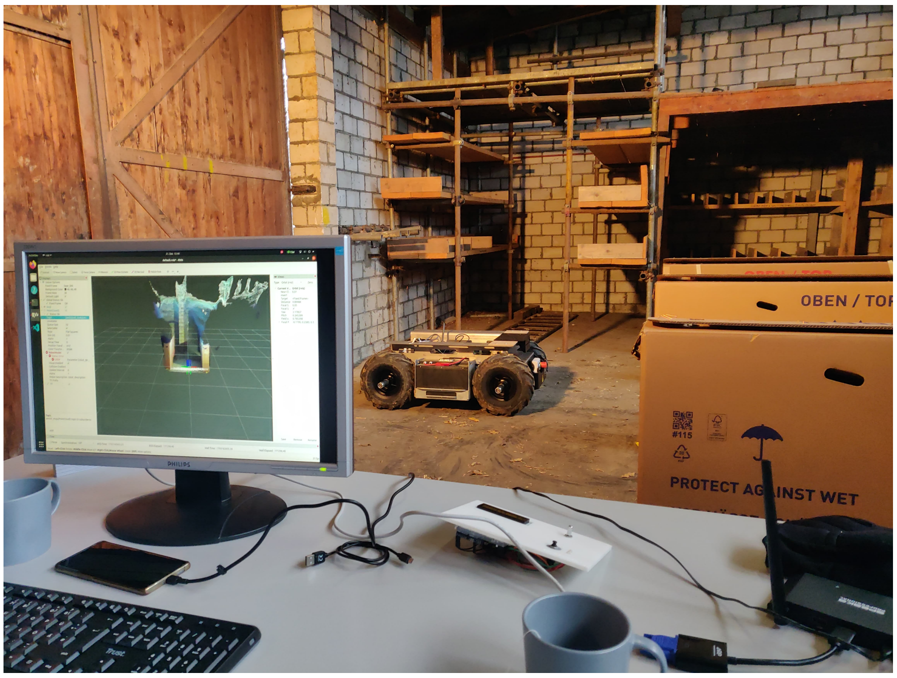


Figure 9.2: Testing environment with spotlight

For a steady power supply, a cable reel was pulled over from the school to the shed. Having a single power source limited the amount of watts we were allowed to use.

In regards to that, the light source had to be switched from a strong spotlight 9.2 to a LED tube 9.3. This sadly reduced the amount and quality of light we were able to get tremendously.

For the row replacement, multiple bike stores have been called to ask for bike cartons. Bike cartons offer the perfect size and limit the risk of damaging the ADIR if it should drive directly into the rows. See Figure 9.4

In the testing environment are three fake plant rows which are put up with a spacing of 150 cm. Because of the space constriction, only a closed-row setup would fit. Open rows were tested anyway, but could not be tested in both ways.

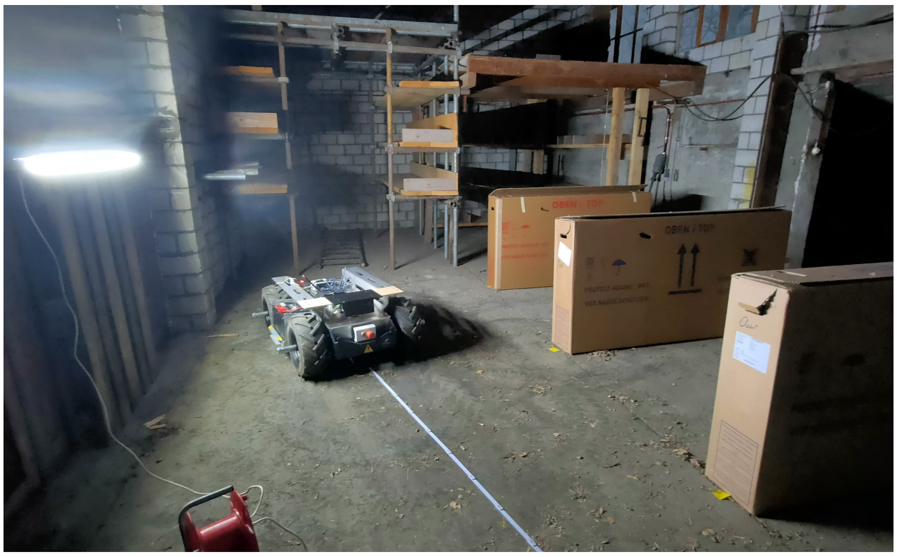


Figure 9.3: Testing environment LED tube

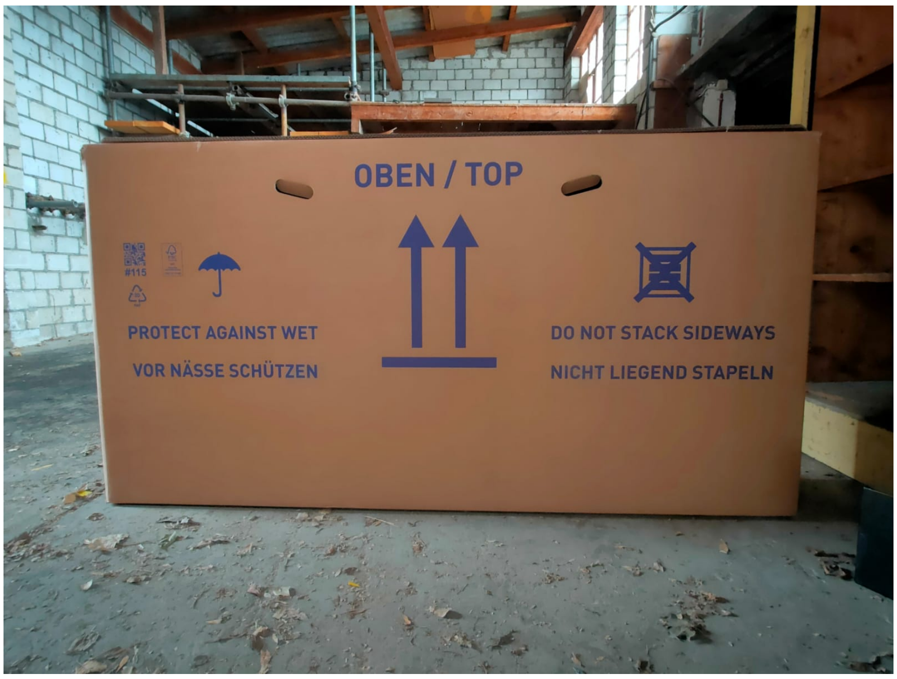


Figure 9.4: Carton row replacement

9.2 Forwards row change

The forwards row change has proven to be successful in both the simulation and in field testing.

9.2.1 Simulation configurations

For the forwards row change in the Gazebo simulation environment, the following parameters have been used in different configurations while testing:

Configuration 1

Global planner	navfn
Local planner	base_planner
Number of waypoints	11
Pre-generated map	Yes

Table 9.1: Testing parameters simulation forward row, configuration 1

At first, the approach had an additional step of generating a map of the environment from the preset file and using that for navigation. This did not prove to be useful in this use case, since the localisation algorithm was unable to pinpoint the current position of the robot. The map did not contain enough detail information.

Configuration 2

Global planner	navfn
Local planner	base_planner
Number of waypoints	11
Pre-generated map	No

Table 9.2: Testing parameters simulation forward row, configuration 2

So the pre-generated map was removed and instead used a SLAM algorithm now. This proved much more successful and the simulated ADIR was able to drive towards the waypoints. After the robot reached a few goals however, it would always get stuck right next to the path, unable to find a way to move sideways back on the path.

Configuration 3

Global planner	global_planner
Local planner	dwa_local_planner
Number of waypoints	7
Pre-generated map	No

Table 9.3: Testing parameters simulation forward row, configuration 3

Finally after switching out both the local and global planner and tuning their parameters, the robot was able to drive to each waypoint one after another and drive into the next row.

Simulation results

The results from the simulation were promising. Even though the robot had to explore its surroundings as it was following the waypoints, it now followed the path reliably.

9.2.2 In field configuration

For the forwards row change in the FHGR testing environment the following parameters have been used in different configurations while testing:

Configuration 1

Global planner	global_planner
Local planner	dwa_local_planner
Light condition	Daylight with closed shed
Number of waypoints	7
Pre-generated map	No

Table 9.4: Testing parameters in-field forward row, configuration 1

The first test in the shed were promising. Most of the time, the robot got a few waypoints in, before ultimately getting stuck somewhere along the way.

As it turned out, the light level was insufficient and because of that, the stereo camera and odometry occasionally reported the same obstacles again, but a few meters further away from the current position. This resulted in the trajectory strafing a non-existing obstacle and the robot not being able to get close enough to the waypoints.

Configuration 2

Global planner	global_planner
Local planner	dwa_local_planner
Light condition	Daylight with open shed and LED bar
Number of waypoints	7
Pre-generated map	No

Table 9.5: Testing parameters in-field forward row, configuration 2

After opening all of the doors, including the large garage entry, there was enough light for reliable mapping and localization. With some subsequent parameter tuning for the in-person ADIR, the row change was now working reliably. Even minor imperfections in the map did not pose an issue.

9.2.3 In field results

The results in the shed were very dependent on the current light level. Later on in the evening, even with multiple artificial light sources it was not possible to get accurate positional readings anymore.

On a single day during the available test phase, the weather was good enough to test our setup in broad daylight. The issue here was, that there was now a much larger amount of data for the stereo camera to process, since it was not in a crammed garage anymore. With this addition required processing, the Jetson could not keep up with the calculation and positioning was even less reliable than in the dimly lit garage.

9.2.4 Summary forwards row change

In both the simulation and the testing environment at the FHGR the ADIR did successfully complete the row change without damaging or hitting the rows.

In the real world, the amount of light and the processing power are however limiting factors.

9.3 Backwards row change

For the backwards row change, tests in a Gazebo simulation and the FHGR testing environment were done.

9.3.1 Simulation configurations

For the backwards row change in the Gazebo simulation, the following parameters have been used in different configurations while testing:

Configuration 1

Global planner	global_planner
Local planner	dwa_local_planner
Number of waypoints	31
Pre generated map	No

Table 9.6: Testing parameters simulation backward row, configuration 1

The ADIR showed no direct backwards motion, but started a rotary recovery, meaning the ADIR spun in circles hitting the rows. Using the dwa_local_planner for a backwards motion in a straight line has been working.

Configuration 2

Global planner	global_planner
Local planner	teb_local_planner
Number of waypoints	31
Pre generated map	No

Table 9.7: Testing parameters simulation backward row, configuration 2

A change from the dwa_local_planner as seen in configuration one, see 9.6, to the teb_local_planner was made. The ADIR did drive backwards but upon reaching each waypoint took a large amount of time to reach the exact waypoint. While trying to reach the waypoint the ADIR tried to position himself as best as it could, meaning it would drive forwards and backwards numerous time per waypoint. This configuration was deemed unusable.

Configuration 3

Global planner	global_planner
Local planner	teb_local_planner
Number of waypoints	11
Pre generated map	No

Table 9.8: Testing parameters simulation backward row, configuration 3

Keeping the planner from configuration two, see 9.7, the only change made was the amount of waypoints. The ADIR showed improved functionality and was still able to follow from one waypoint to the next without stopping at everyone to replace it self. The local planner can generate a longer path which helps the ADIR to redirect himself directly while driving. The row change backwards was successfully done in the simulation.

Configuration 4

Global planner	global_planner
Local planner	teb_local_planner
Number of waypoints	5
Pre generated map	No

Table 9.9: Testing parameters simulation backward row, configuration 4

Seeing as the performance has improved with less waypoints another test was done with the changing parameter being the number of waypoints which were reduced to 5. With a number as less as five the ADIR tried to turn around and drive forwards to the next waypoint which is not what it is pursued within this test. A number of five waypoints was deemed unusable.

9.3.2 Simulation results

In regards of testing in the simulation, configuration 3, see 9.8, was the most successful and has fulfilled a row change backwards without striving from its trajectory or hitting any of the rows.

9.3.3 In field configuration

For the backwards row change in the FHGR testing environment, the following parameters have been used in the configuration:

Configuration 1

Global planner	global_planner
Local planner	dwa_local_planner
Light condition	Dusk and LED tube
Number of waypoints	11
Pre generated map	No

Table 9.10: Testing parameters in-field backwards row, configuration 1

For configuration one the dwa_local_planner was used since the parameters already have been tuned for the forward row change with the help of the ROS navigation tuning guide, see [5]. The ADIR did not attempt a backwards motion but instead started a rotary recovery, meaning it crashed in the surrounding cartons. It is unclear if it would have worked in proper light conditions.

Configuration 2

Global planner	global_planner
Local planner	teb_local_planner
Light condition	Dusk and LED tube
Number of waypoints	11
Pre generated map	No

Table 9.11: Testing parameters in-field backwards row, configuration 2

Upon testing in the FHGR testing environment the test failed immediately since the robot did not know the environment behind him. The ADIR was not able to navigate into the next row without already knowing the environment.

Reason for this being a limitation in the teb_local_planner, which does not allow goals outside of a already known area.

9.3.4 In field results

Neither configuration has been able to execute a working row change.

9.3.5 Summary backwards row change

The backwards row change has only been partly successful.

Driving from one row into the next one in the simulation was only successful when a pre generated map was passed to the ADIR with about eleven waypoints. Using the backwards row change in an environment unknown to the ADIR did not work.

Testing the row change backwards in the testing environment at the FHGR has not been successful. Worth mentioning is, that the testing time was around 5pm where the light level was most likely not sufficient enough anymore for the stereo camera to work properly. Having only a limited time at the test facility, development was halted at this point.

10 Conclusion

10.1 PCT

Pros

- Easy usage
- Parameters can be added

Cons

- Difficult installation and usage for people unfamiliar with Python
- Window is not scaleable

10.1.1 Results

The PCT has been successfully structured and programmed. After getting feedback in the concept decision meeting, additional inputs were added, for a picture to be shown with all parameters on it. It is also easily possible to add additional parameters in the future, when necessary. It has a simple design which allows easy and straight forward usage.

10.1.2 Evaluation

The program fulfils all must have requirements from the specification list. At least six parameters are taken into account ID 9, the output of the preset creation tool is a ”ROS YAML” file ID 10. The PCT is programmed with python ID 11. The PCT only saves the preset if all parameters are filled out ID 12. The generated YAML file can be transferred onto the robot via USB ID 13.

For a full list of fulfilled requirements see Attachment 5.

10.1.3 Problems/Open points

Installing the program requires the installation of additional libraries. This could be managed by making a python executable which allows for even easier usage. The window of the application is not scaleable which looks either smaller or bigger depending of the display resolution.

10.2 OBUI

Pros	**Cons**
• Easy use • Low power consumption • Multiple voltages possible for power delivery	• Not waterproof

10.2.1 Results

The hardware for the OBUI was selected and meets the requirements. Besides switching the green LED for a red one since the supplier could not deliver it in time. The electronics are neatly packed inside a box which can be easily opened by four screws on the back. The emergency button was left out since it was already implemented on the ADIR.

10.2.2 Evaluation

The system does fulfill all must have requirements. The user can choose between five presets ID 20 and is able to push a button to switch between manual mode and auto mode ID 21 and an Arduino Uno R4 minima is used ID 22 which was specified. As a simple proof of concept it works reliably.

For a full list of fulfilled requirements see Attachment 5.

10.2.3 Problems/Open points

The microcontroller could be placed more on the side of the box for a easier cable plug in. Additionally the box is not waterproof and using it in field without waterproofing provides a slight risk, should there be any rogue droplets from the sprayer.

10.3 Row change

Pros

- Generated its own trajectory based on the environment.
- Detects sudden obstacles
- Utilizes stereo camera
- High precision odometry

Cons

- Slow row change
- Works unreliably in dark light conditions
- Can not detect obstacles while driving backwards
- Not always friendly to the ground

10.3.1 Results

The row change fulfills most of the must requirements. A row change forwards is possible in the FHGR testing environment and in the simulation. A row change backwards was not achieved in the FHGR testing environment. If it would have worked is unclear since the light conditions were not good enough for a proper test. In the simulation however, it managed itself quite well.

However, as mentioned previously, working in moderate to bad light conditions lowers the accuracy tremendously.

10.3.2 Evaluation

The must have requirements were almost all fulfilled. Requirement ID1, ID2 are fulfilled since the row change starts upon receiving said messages. ID3 work in the simulation but is not confirmed to work in the testing environment of the FHGR.

ID 4 is fulfilled since when the ADIR does a row change forward it drives into the next row forwards. ID 5 is not fulfilled since in the testing environment it was not confirmed if the ADIR would have driven backwards into the next row, it is worth mentioning that it has been working in the simulation. ID 6 is halfway fulfilled with the robot taking the parameters given to him and calculating a trajectory in a predetermined space is working with the row change forwards and backwards following the trajectory backwards is not confirmed in the testing environment.

For a full list of fulfilled requirements see Attachment 5.

10.3.3 Problems/Open points

The biggest problem is the use of the stereo camera in not sufficiently lit areas. It limits the usage of the ADIR in darker environments since the robot can not track its current position properly anymore.

A calculated path is made but can not be passed as standalone message to the local planner. To implement this, it is necessary to implement the row change trajectory planner as a move_base plugin, which was not possible in the given time frame.

Given a published path, the teb_local_planner should be able to follow the path closely instead of following the waypoints one after another.

Because of time constraints, the backwards turn to the left has been implemented, but not yet sufficiently tested.

With the current setup, the ADIR does sometime damage the ground in the form of steer marks. Because of the global planner, it often drives to the next waypoint, stands still and then proceeds to turn on the spot until the correct position is reached.

10.4 Outlook and Recommendations

There are still a few things left that can be improved. All the things that can be improved are listed below with a short description.

10.4.1 PCT

- Software
  - As of now the main window is not scaleable and does look different depending on the display resolution. Implementing the function scaleability is recommended.
  - The current installation of the PCT can be considered difficult for people who are not familiar with Python or Linux. A Python executable would provide the possibility to just use the software with a double click.

10.4.2 OBUI

- Box design
  - The box is as of now not waterproof. An application of a sealing agent to the shell is recommended.
  - The microcontroller could be placed differently to plug in the USB cable. As of now, one has to open the back panel.

10.4.3 Row change

- Row change trajectory

  - The current path is not directly used, only certain points from it. To make the most of it, a global planner could be written to pass the path to the local planner directly. The code for the path planner already exists, but it needs to be rewritten in c++ and integrated as a move_base plugin.
  - Since the robot has to drive a certain distance backwards to the next row, the turning direction could alternate during the program cycle. This would prevent the trajectories from hitting either wall at the sides.

- Obstacle avoidance on the rear side
  - Additionally, another stereo camera could be used to check for obstacles on the back. This would also enable for the current setup with teb_local_planner to work. Additionally, backwards row changes might not be needed at all.
  - Using a LIDAR is also an option, but the position could pose problems in detecting objects of different heights.
- Light conditions
  - Only using the ADIR in areas were it is well lit enough for the stereo camera to function properly.
  - It may be beneficial to limit the stereo camera's depth in order to save on processing power.
- Ground conditions
  - At the moment the ADIR does leave black skid marks on the ground while turning on the spot. A different type of tire material could prevent that, enabling easier testing indoors.
- Plug for NVIDIA Jetson power supply
  - If the NVIDIA Jetson should stay on the ADIR, a proper power plug would be recommended. The current one is 3D printed without reverse polarity protection.
- Hardware setup for remote operation
  - A general setup of components required for remote work on the NVIDIA Jetson is recommended for autonomous testing since being to close to it while testing provides a high risk for accidents.
- Placement of emergency button
  - The emergency button should be replaced so that it is accessible from every side of the ADIR. If the robot traps a user between him and a wall with its backside, it proved to be impossible to reach the front to press the emergency stop button.

11 Attachments

1. CAD Files
    - Standing rows model
      Description: CAD model of standing rows for Gazebo simulation.
      File: standing_row.step
      Date and Time: 07.01.24 15:51

    - Hanging rows
      Description: CAD model of hanging rows for Gazebo simulation.
      File: hanging_ros.step
      Date and Time: 07.01.24 15:51

    - Plug
      Description: CAD model of plug for power supply from the ADIR to the NVIDIA Jetson.
      File: Plug.step
      Date and Time: 04.01.23 18:57

    - OBUI
      Description: CAD model of the OBUI.
      File: OBUI.step
      Date and Time: 04.11.23 18:59

2. Calculations
    - Trajectory calculations
      Description: Pythonfile for calculating the trajectories.
      File: row_change_trajectory.py
      Date and Time: 22.12.23 18:13

3. Circuit diagram
    - OBUI circuit diagram
      Description: Circuit diagram for OBUI.

File: obui.pdf
Date and Time: 05.01.24 11:37

- OBUI BOM
  Description: Bill of materials OBUI.
  File: BOM.xlsx
  Date and Time: 07.01.24 18:44

4. Datasheets
   - Encoder
     Description: Datasheet of used encoder in OBUI.
     File: PEC11R.pdf
     Date and Time: 04.01.24 18:57
   - Transistors
     Description: Datasheet of used transistors in OBUI.
     File: 2N4401.pdf
     Date and Time: 05.01.24 11:47
   - Arduino Uno R4 Minima
     Description: Datasheet of used microcontroller in OBUI.
     File: ABX00080-datasheet.pdf
     Date and Time: 04.01.24 18:56
   - LEDs
     Description: Datasheet of used LEDs in OBUI.
     File: AV02_0373EN_DS_HLMP_ELxx_2018_03_12-1827869.pdf
     Date and Time: 04.01.24 19:17
   - LCD
     Description: Datasheet of used LCD in OBUI.
     File: JDH_1804_Datasheet.pdf
     Date and Time: 04.01.24 19:17
5. Requirement
   - Requirement specifications
     Description: Full requirements specification document.
     File: Requirements specifications v1.9.pdf

Date and Time: 24.10.23 15:34

- List of fulfilled requirements
  Description: Excel sheet of fulfilled requirements.
  File: List_of_fulfilled_requirements.xlsx
  Date and Time: 07.01.24 13:45

6. ROS diagrams
   - ROS node and topic graph
     Description: Rosgraph of nodes and topics.
     File: rosgraph.png
     Date and Time: 06.01.24 16:38

7. Software
   - OBUI
     - Description: Code for the OBUI.
       File: OBUI_akt.ino
       Date and Time: 05.01.24 23:12
     - Description: State event diagram for OBUI code.
       File: SED.png
       Date and Time: 19.12.23 17:36
     - Description: State event table for OBUI code.
       File: OBUI_SET.ods
       Date and Time: 19.12.23 18:06
   - PCT
     Description: Code for the PCT.
     File: pct.zip
     Date and Time: 07.01.24 14:17
   - Row change
     Description: Catkin workspace of row change.
     File: catkin_ws.zip
     Date and Time: 07.01.24 12:54

- Map converter
  Description: Code for map converter.
  File: ros_map_converter.zip
  Date and Time: 07.01.24 14:18

8. Timetable
   Description: Timetable with original planned timeline and added actual timeline.
   File: Timetable.ods
   Date and Time: 07.01.24 14:11

9. Concept Ideas
   Description: List of all concepts that were discussed.
   File: Concept_Decisions.pdf
   Date and Time: 07.01.24 16:18

Bibliography

[1] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997. DOI: `10.1109/100.580977`.

[2] On Semiconductor, *General Purpose Transistor NPN Silicon*, 2010. [Online]. Available: `https://www.farnell.com/datasheets/661741.pdf`.

[3] C. Rösmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," Jan. 2012, pp. 1–6, ISBN: 978-3-8007-3418-4.

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

[5] Kaiyu Zheng, "ROS Navigation Tuning Guide," Tech. Rep., Sep. 2016.

[6] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, Feb. 2017, ISSN: 0921-8890. DOI: `10.1016/J.ROBOT.2016.11.007`.

[7] Broadcom, *T-1$\frac{3}{4}$ (5 mm) Precision Optical Performance AlInGaP LED Lamps*, 2018. [Online]. Available: `https://www.mouser.ch/datasheet/2/678/AV02_0373EN_DS_HLMP_ELxx_2018_03_12-1827869.pdf`.

[8] Seeed, *SPECIFICATION OF LCD MODULE*, 2018. [Online]. Available: `https://raw.githubusercontent.com/SeeedDocument/Grove-16x2_LCD_Series/master/res/JDH_1804_Datasheet.pdf`.

[9] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. DOI: `https://doi.org/10.1002/rob.21831`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831`.

[10] T. Kitamura, *Fusion2URDF*, https://github.com/syuntoku14/fusion2urdf, 2020.

[11] mprprograms, *GitHub - mprograms/SimpleRotary*, Dec. 2021. [Online]. Available: `https://github.com/mprograms/SimpleRotary`.

[12] Seeed Studio, *GitHub - Seeed-Studio/Grove_LCD_RGB_Backlight: Seeedstudio, Grove - LCD RGB Backlight*, Oct. 2022. [Online]. Available: `https://github.com/Seeed-Studio/Grove_LCD_RGB_Backlight`.

[13] Arduino, *Product Reference Manual*, 2023. [Online]. Available: `https://docs.arduino.cc/resources/datasheets/ABX00080-datasheet.pdf`.

[14] Bourns, *PEC11R Series - 12 mm Incremental Encoder*, 2023. [Online]. Available: `https://www.bourns.com/docs/product-datasheets/pec11r.pdf`.

[15] MicroBeaut, *GitHub - MicroBeaut/ADebouncer: Advanced Debouncer Library for Arduino*, 2023. [Online]. Available: `https://github.com/MicroBeaut/ADebouncer`.

[16] Ant Robotics, *ADIR Features.* [Online]. Available: `https://www.adir.cc/index.php/features/`.

[17] *base_local_planner - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/base_local_planner`.

[18] Brian P. Gerkey, *amcl - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/amcl`.

[19] Gazebo classic, *Gazebo : Tutorial : Gazebo plugins in ROS.* [Online]. Available: `https://classic.gazebosim.org/tutorials?tut=ros_gzplugins`.

[20] *global_planner - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/global_planner`.

[21] Metasa, *Channel cart system (130 mm).* [Online]. Available: `https://www.metasa.de/en/produkte/rinnen-stellage-system-130-mm`.

[22] *move_base - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/move_base`.

[23] *navigation - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/navigation`.

[24] Slamcore, *slamcore/follow_waypoints.* [Online]. Available: `https://github.com/slamcore/follow_waypoints`.

[25] teb team, *teb_local_planner - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/teb_local_planner`.

[26] Tom Moore, *robot_localization wiki.* [Online]. Available: `http://docs.ros.org/en/melodic/api/robot_localization/html/index.html`.

[27] TU Dortmund, *dwa_local_planner - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/dwa_local_planner`.

[28] Wim Meeussen, *robot_pose_ekf - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/robot_pose_ekf`.

[29] ZED, *zed-ros-wrapper - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/zed-ros-wrapper`.

List of Figures

List of Tables

12 Declaration of authorship

≪I hereby declare that

- I have completed this work independently and have not used any other means than the specified sources and permitted resources. I have indicated all passages which have been derived from outside sources either literally or in their general substance and am aware that, should I have failed to do so, the University Administration is entitled to revoke the qualification or title granted to me on the basis of my work.≫

[illegible]

Marti Lukas

FHGR

Deborah
Schrag

Schrag Deborah

FHGR