ERF Hackathon 2022

Part of Tweedejaarsproject BSc KI

Derck Prinzhorn 13058207
Thijmen Nijdam 12994448
Jurgen de Heus 13162829
Juell Sprott 13101817

UNIVERSITY OF AMSTERDAM

FNWI
University of Amsterdam
Netherlands
July 3rd 2022

Contents

1 Introduction **2**
- 1.1 Client description . . . 2
- 1.2 Problem description . . . 2
- 1.3 Challenge description . . . 3
- 1.4 Previous research . . . 5
- 1.5 Product vision . . . 5

2 Implementation **5**
- 2.1 Proposed solution . . . 5
- 2.2 Provided hardware and software . . . 5
- 2.3 Navigation . . . 5
  - 2.3.1 Model-based approach . . . 5
  - 2.3.2 Behaviour-based approach . . . 10
  - 2.3.3 The algorithm . . . 10
- 2.4 Communication . . . 11
  - 2.4.1 Approach . . . 11
  - 2.4.2 Between Junos . . . 12
  - 2.4.3 Developer . . . 13
- 2.5 Features . . . 13
  - 2.5.1 Homing . . . 13
  - 2.5.2 Juno vicinity lights . . . 15

3 Conclusion **15**
- 3.1 Juno 1 . . . 15
  - 3.1.1 Problems . . . 16
  - 3.1.2 Future improvements . . . 16
- 3.2 Juno 2 . . . 16
  - 3.2.1 Problems . . . 17
  - 3.2.2 Future improvements . . . 17

4 References **17**

Appendices **17**

A ROS **17**

B Hardware **18**

C Software **18**

D Sensor hardware and software **18**

E Other open source ROS packages **19**

F diagrams **20**

1 Introduction

1.1 Client description

Innovations are found in a variety of places in society. This report focuses on innovations in the agricultural sector. There are two main clients involved. An agricultural innovating company and a robotics company. Those two companies are connected to us through our third client, Arnoud Visser. He is our supervisor from the UvA and a professor specialized in robotics.

Company introduction Lely

One of the pioneers in agricultural innovation is Lely, which is the client involved in this project. Lely offers solutions for almost all activities in the cowshed, to make agrarian life easier.

Company introduction RoboHouse

RoboHouse is a company with a network of robotics experts and industry partners, which believes new innovations are necessary to make work more healthy and empowering. It's located in the TU Delft Campus where innovative organizations can come together to develop and test their developed applications in a realistic lifelike environment.

1.2 Problem description

In a barn, cows need to be fed. One of the ways in which they are being fed is by grass-like food, for example hay. The cows reach through a feeding fence, to eat the feed but also slowly push the feed away from the fence. They can do this until they are unable to reach the feed. Therefore the feed needs to be pushed back to the fence by the farmer. This is very time-consuming which is the reason why Lely invented the Juno robot (Figure 1). The Juno is a feed pushing robot that pushes feed towards the feeding fence.

Figure 1: The Juno robot in a barn.

1.3 Challenge description

Lely has the ambition to create smarter robots and wants to challenge students to come up with creative ideas and solutions. This year Lely collaborates with Robohouse during the ERF, and organizes a hackathon. There are many challenges for robots, such as the Juno, because of the different factors in the barn which vary. The barn the Juno has to drive through could be dirty. The layout of the barn could be different, with people walking around. In addition cows could be sticking their heads through the feeding fence. The Juno has to ensure that the barn is a safe environment for the people and animals in it by being able to sense the environment and communicating about this to the farmer and other robots. In this challenge two Juno robots have to push the feed in a barn environment in as less time as possible. An example barn environment is shown in Figure 2. Junos must drive a route through the barn environment. Barns vary every day, but there will always be a narrow passage where only one Juno can pass at once. The Junos are not allowed to collide with each other and the farmer.

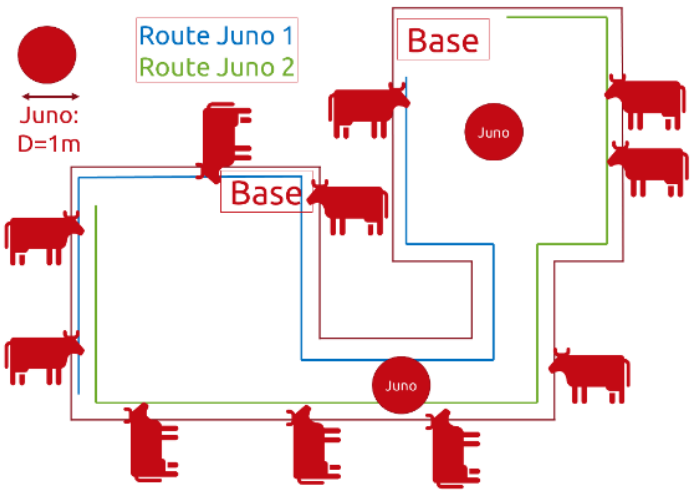

Figure 2: Example of a barn environment.

In addition to the principal problem statement, there are additional challenges to be completed to gain extra points in the Hackathon:

1. The Juno needs to (autonomously) inform the farmer that it is in the narrow passage where only one Juno can pass.
2. Pass through a plastic strip curtain. The curtain will be set up in an unknown location in the barn environment when the teams decide to tackle this challenge.
3. Be able to cope with parts of the feeding fence having been removed.
   (a) A farmer could cross the road at the section without the feeding fence.
   (b) Communicate to the farmer that he can cross.
4. Leave from/return to 'home' base (touch traffic cone) in the middle of the arena instead of a corner.
5. Leave from/return to 'home' base (touch traffic cone) at a random location in the arena instead of a corner or exactly the middle of the arena.
6. Drive around obstacle(s) (for example, mini wheelbarrows).
7. Count (our inflatable 'hot') cows and inform the farmer. The heat will be provided by infrared heating lamps in a protective cage. The approximate temperature will be 75 - 100 °C.
8. Deal with failing network during route (communication). The robot should still be able to perform his tasks without the connection to the cloud or second Juno.

9. Turn lights on if there is a Juno in the neighborhood.

1.4 Previous research

The current Juno robot is a stand alone robot, which uses a predefined route to navigate in the barn environment. The benefits reported by Lely themselves are firstly an increase in the food consumption of cows, which has multiple advantages. For example, the improvement of overall health, fertility and milk production. Secondly, Lely mentions that farmers report that they are saving 180 hours a year (half an hour a day), which can be spent doing other activities on the farm.

1.5 Product vision

The software for the Juno has to be implemented in very short time, less than two weeks. Therefore the goal is to create feasible robot software with at least three primary assets. The first asset is navigation. The Junos have to be able to navigate in the barn without any objects in it, following the route along the fence. The second asset is obstacle avoidance. While navigating in the barn the Junos have to avoid random object in the barn and continue it's route. The third primary asset is communication. The farmer has to be informed in special situations, for example when the farmer wants to cross the Juno. Furthermore the Junos have to inform each other, so they don't collide. In addition to the primary assets extra features, like counting cows and moving through a curtain, are possible secondary assets.

2 Implementation

2.1 Proposed solution

There are two Junos. One is the master Juno and the other is the client Juno.
The master Juno will follow a model-based approach.
The client Juno will follow a behaviour based approach.

2.2 Provided hardware and software

For this project, we have received a Mirte robot, a Turtlebot3 robot and a Juno robot during the 3 hackathon days. Each of these also make use of ROS, a framework that allows us to control these robots. More information about the hardware and software can be found in the appendix.

2.3 Navigation

2.3.1 Model-based approach

The first Juno, the master Juno, will use a model-based approach for navigation. This approach makes use of advanced techniques in order to generate a route

that can be used by the Juno to traverse the environment. It is very important that the model is representative for the environment in which the robot has to be able to operate. This approach requires a greater degree of ROS understanding, but allows for more diverse applications. When the model is accurate this approach has the potential to be a great fitting solution for our problem, as it can more accurately handle unique cases in our environment which other approaches will tend to struggle with. Since there are limitations on hardware, this approach is feasible on only one Juno.

This approach makes use of the ROS Navigation Stack. While this uses sensor inputs similarly to other approaches, it makes use of these sensor inputs to both generate a cognitive map for the robot to move in, as well as give the robot to orient itself inside this map. Using these inputs, it is then possible for us to generate a plan for the robot which it can then traverse. This approach has two large advantages. The first is that for our challenge a path must be traversed that requires us to traverse one side of the barn environment from x distance to the barn wall, and then return according to the same path from y distance to the wall. The navigation stack significantly streamlines the path planning involved in this task as it can simply pass path plans to our robot after which they will autonomously follow the given path. The second is that, due to its smart usage of sensor inputs, the robot can actively avoid obstacles without any interference from our end. This makes the model-based approach near perfect for our problem. However, implementing this approach has several downsides. For one, using this method requires some preparations for our robot, both hardware and software. Second, after first implementation and testing of the navigation stack, fine-tuning is required for path planning to achieve the desired route with high accuracy.

The ROS Navigation Stack has several key points, which will be discussed one by one:

- sensor streams
- robot odometry
- map creation
- path planning

Sensor software and hardware are further discussed in the appendix.

Robot odometry is an import aspect of the navigation stack, as it is used in combination with sensor inputs to provide improved map creation as well as active route planning when dealing with obstacle avoidance. Odometry provides an estimate of the robot's position relative to it's starting position using either additional sensor inputs or wheel encoders. In the case of wheel encoders, these record the position and orientation of a robot position over time. This is

done by by keeping track of the movement of the wheels and converting these in both positional and orientational coordinates. This information can be used to obtain localization. Localization can be used in tandem with a given map to estimate the robot's position relative to the given map, which will prove useful to both map creation as well as path planning. For our final implementation, two methods of localization are used. One for map creation and another for path planning. These are the SLAM algorithm and AMCL algorithm respectively.

In order for us to plan a path for the robot, a static map must be created which contains the entire environment in which the autonomous robot has to navigate. This map can then serve as a reference point for where static obstacles and walls could possibly be so it is possible to plan around them. The navigation stack uses the SLAM algorithm to perform map creation. Otherwise known as simultaneous localization and mapping. This technique makes use of LiDAR inputs in conjunction with the previously mentioned localization in order to actively create a static map while driving through the environment.

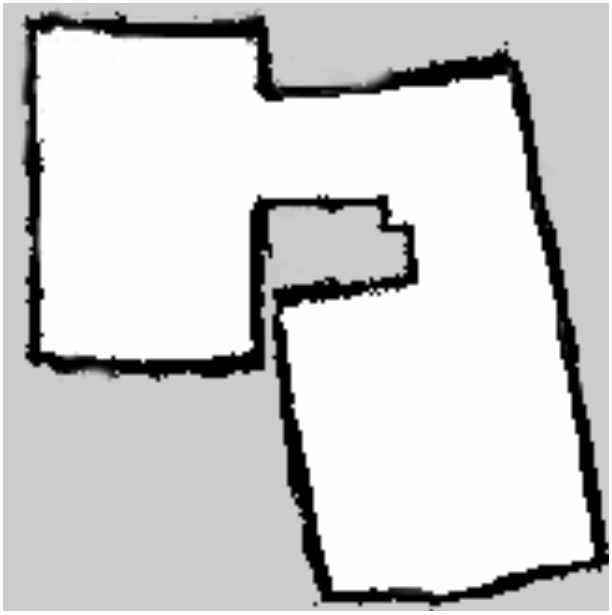

Figure 3: A map created of the barn environment using SLAM, visualized in Rviz.

The SLAM algorithm will use feature extraction on the laser inputs in order to obtain static walls and obstacles, which are then saved in a map. Simultaneously, the same laser scan data is used by SLAM to localize the robot within that same map. The method used for localization generally differs between each SLAM algorithm, however most algorithms generally use laser data or wheel odometry. Due to the fact that wheel odometry can be unstable in

certain environments – such as the barn environment – the decision has been made to use the Hector-SLAM algorithm, which is far less reliant on odometry data. In turn, the algorithm is far more reliant on the LiDAR. This algorithm has been optimized for usage with the LiDAR however and due to it's budget friendly nature compared to other algorithms, as well as better ease of use and more documentation, Hector-SLAM has ultimately been chosen for the final implementation.

Map creation, robot position and eventual map saving are all visualized within Rviz, a GUI made for ROS that supports the ROS navigation stack which takes in topic messages and visualizes them for improved ease of use and debugging.

With map creation out of the way, the next step is working with the core of the navigation stack. The move_base package provides the stack with the required code and nodes to get a functioning path planning setup. Move_base takes in the aforementioned sensor streams, robot odometry and static map to set up the navigation stack core. With this and a goal pose within the given map given by either the user or a separate program to then calculate the most optimal path towards said goal.

The move_base core consists of a global costmap, a local costmap, a global planner and a local planner. A global costmap consists of the normal SLAM-generated map combined with map layers that modify the map in various ways. By the default an inflation layer is placed on top of the costmap which inflates any objects on the map, resulting in any planners being used attempting to plan around these inflations, which ultimately provides safer path planning. The global costmap makes use of the map created with the SLAM algorithm to generate a path from the initial pose towards a given goal. This is done with the global planner, which generates the most optimal route from the initial pose to the goal using one of the algorithms that can be configured. For the challenge, a route that follows the walls is required with a focus on straight paths. Several configuration combinations have been tested on both Mirte and Turtlebot3 to achieve this goal, with the ideal configurations shown below:

```
use_grid_path=True
use_dijkstra=True
old_navfn_behaviour=False
```

The grid path straightens the path as a grid is overlayed over the map. Any planners used with this grid map will only follow lines on the grid. The $use\_dijkstra$ variable will determine whether Dijkstra's algorithm or A* algorithm is used for path planning. For both algorithms, each point in the coordinate frame of the map that does not fall within the inflation layer is used for calculating the optimal path.
With the global plan created, the next step is working locally with the local costmap and local planner. The local costmap uses both localization and Li-

DAR laser scans to generate a local costmap within the robot's vicinity. This local costmap works similarly to the global costmap, but uses LiDAR input data to actively change the map in order to account for obstacles not found in the global costmap. This way, even if the environment changes after creating the SLAM map, the robot can still cope with new obstacles. This is then combined with a local planner, which takes the global planner and attempts to follow the global plan while attempting to navigate past obstacles and adjust the orientation and position of the robot such that the robot does not run into collision problems.

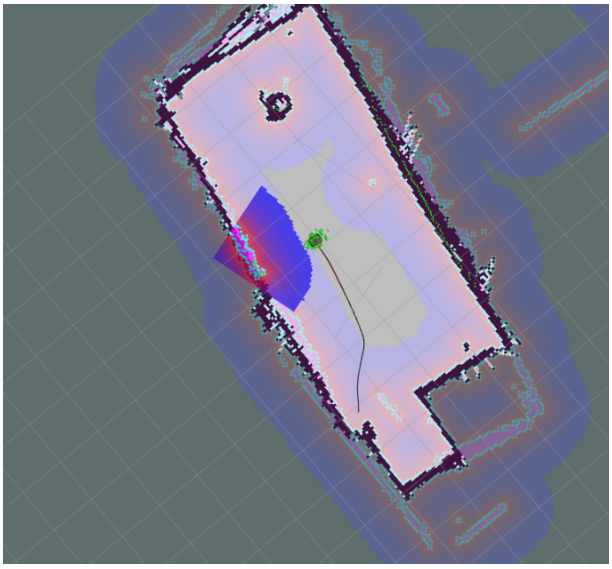

Figure 4: An example of a planned path.

Once the navigation stack is configured, the next step is to generate a route that moves along the walls in both directions. For this we create a finite state machine that takes in a set of goals – referred to as way-points from this point on wards – and sends these way-points to the robot one by one. After each way-point, a check is done if the final point is reached. If not, the robot will then obtain the next way-point and send the path for this point to the robot until each point has been reached. A finite state machine has been chosen as we can add additional features like communication that can run simultaneously with route navigation. The ROS Smach package perfectly integrates these functions as both our navigation and our additional features are ROS-based. This allows easier creation of the FSM and debugging and improving our FSM. More information about the ROS Smach package and a detailed FSM used by the Mirte, Turtlebot3 and Juno can be found in the appendix.

2.3.2 Behaviour-based approach

The second Juno, the client, will use a behaviour-based approach for navigation. This is the most simple approach. With few sensors and generalized behaviours, it is easy to create a robust robot. A behaviour-based robot is also very explainable since it follows clear logical rules about interaction with the environment. However, due to it's simplicity it is not expected to be the most optimal solution for our problem. Since there are two Junos and limitations on hardware (sensors), the behaviour-based approach seems to be a good baseline for both Junos, and possibly the best approach for a less equipped Juno.

The robot has the ability to navigate and avoid obstacles. behaviour-based systems (an example is shown in figure 5) consist out of different modules, called behaviours. A behaviour is a programmed response to a certain sensorial input and/or other system modules. The behaviours can be programmed with a set of functions and can even be designed like a FSM (Finite State Machine) which allows to have transitions through states within a behaviour (Jones and Roth, 2004). The behaviours send outputs to the effectors of the robot and possibly other behaviour modules.

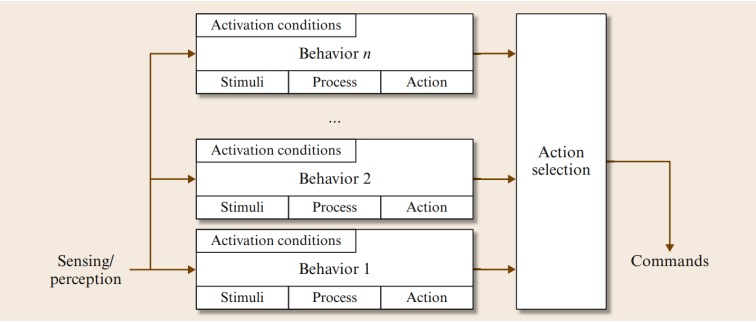

Figure 5: Schematic representation of a simple behaviour-based system. (Matarić and Michaud, 2008)

2.3.3 The algorithm

Navigation is the most important part of the robot. Without navigation, the robot is not able to do anything useful. The robot needs to follow the fence, for which a wall-following behaviour is implemented. In the code block below two cases are described in pseudo-code. The core of the behaviour based approach is that the current state of the robot is checked every iteration of a for loop. By choosing the right behaviour based on conditions with if statements, the robot can follow a wall for a longer period of time. If the robot is too close to the wall and the front is clear, adjust a bit to the right. However, if the front sensor detects something very close, a turn is required.

```
if right_wall is far and front_wall is far:
    adjust_to_right()
elif right_wall is far and front_wall is close:
    turn_left()
elif right_wall is good and front_wall is far:
    move_ahead()
elif right_wall is good and front_wall is close:
    turn_left()
elif right_wall is close and front_wall is far:
    adjust_to_left()
elif right_wall is close and front_wall is close:
    turn_left()
```

Figure 6: Visualization of the wall distance. R corresponds to being in the perfect distance to the wall, -R is close to the wall and +R to far.

2.4 Communication

Concerning the communication, the goal was to set up communication between the Junos and the developer, between the Junos and the farmer and to set up a multi-agent system involving the two Junos. Unfortunately, due to time limits, it was not possible to implement the communication channel between the farmer and the Junos. For this, it was planned to create a web-application using Azure.

2.4.1 Approach

The multi-agent system was implemented using a master and client-system. One Juno was intended to be the master and the other one the client. For this, the ROS-master URI of the client, which is normally just the IP-address of the client

itself, receives the IP-address of the master resulting in the client Juno becoming the client to the master Juno. Because of this, the client Juno participates in the ROS-process of the master Juno. This means that the master and client now share the same ROS-process and thus are connected to each other. Now, one Juno can subscribe to a topic to which another Juno publishes and thus they can communicate to each other. However, the problem of this is that when the Junos share the same ROS-process, they also share the same "cmd_vel" topic which sends the movement commands to the robot. This means that the two robots perform the exact same movements. To resolve this, namespaces had to be created for the topics for which a seperation between the two Junos was required.
For communication between Junos and the farmer, we work with the Azure services provided by Lely to create two resources. These resources are the IoT Hub and the Web App of Azure. For the IoT Hub we make use of the Ros Azure IoT Hub package which allows the robot to send messages and information to the Azure servers. This information can then be send to a web app on a mobile device to notify the farmer about necessary information for the challenge. Examples being route progress and hardware errors.

2.4.2 Between Junos

After this setup was created, the Junos were able to communicate with each other about their states and a multi-agent system was realised. This is primarily necessary for avoiding a collision between the two Junos in the narrow passage. As described above, this is accomplished by making one Juno the master and the other one the client. By making this distinction, this case can be implemented as a Juno-encounter behaviour. The master Juno always gets priority over the client Juno. The master Juno constantly detects whether it is inside the corridor. This is being realized by adding the minimum distance to a wall to the left to the minimum distance to the right and checking whether this distance is smaller then a certain threshold value, approximately the breadth of the narrow passage. The master thus constantly sends a boolean message of this to the client Juno which thus constantly receives this message and simply stops to driving to avoid a collision. Whenever the messages are true again the client Juno will continue it's wall following. Another more sophisticated approach was also implemented where the client itself also detects whether it's inside the narrow passage. Then if the master and the client are both inside the narrow passage, the client will know this and will move backwards out of the corridor and wait outside of it to avoid a collision. If thereafter, the master passed the corridor, the client will be aware of this and will continue it's wall following. However, due to difficulties and time limits, this approach was not used.

2.4.3 Developer

Besides the communication between the Junos, the developer should also be kept up to date about the processes of the two Junos. Because ROS mainly drives on the communication channels called topics, the data that is being sent through these topics is all the developer needs. Because of this, ROSboard is being used. This is a dashboard which presents all the data of the topics in a appropriate way. To realise this, only the required package has to be installed in the workspace. In figure 7 the ROSboard is presented during the processes of the Junos. For example, in the upper left corner, the scan data from the LiDAR is presented. Next to it, the boolean messages of whether the master is in the narrow passage are presented.

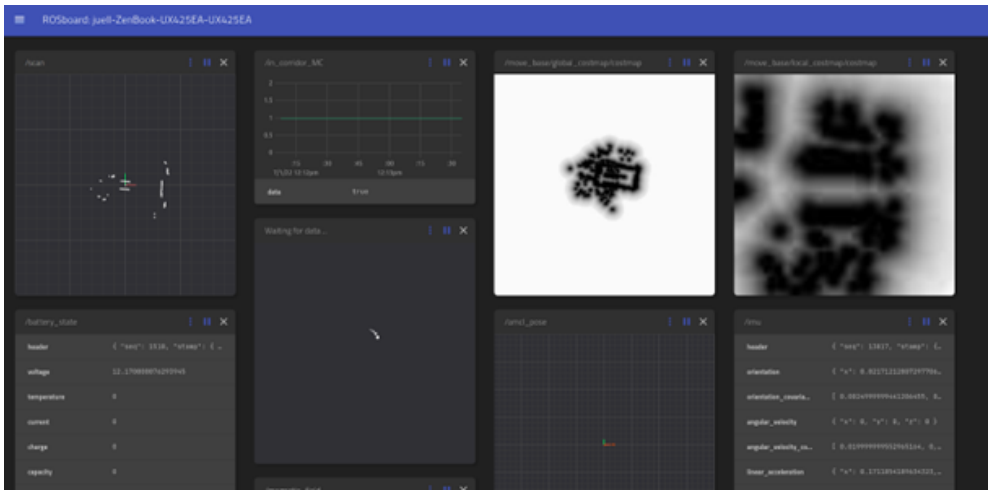

Figure 7: ROSboard during Juno processes

2.5 Features

To score more points, several extra features for the Juno robot were implemented.

2.5.1 Homing

The homing feature was developed using computer vision to detect the base of the Juno. In the challenge it was known to be a traffic cone, which is characteristic of its orange color. With this in mind, an orange blob detector was implemented. The final blob detected actually consists of more smaller blobs combined (Figure 8). To make our algorithm more noise resistant, a threshold is set for the size of the blobs which were allowed to be combined into the final blob (Figure 9).

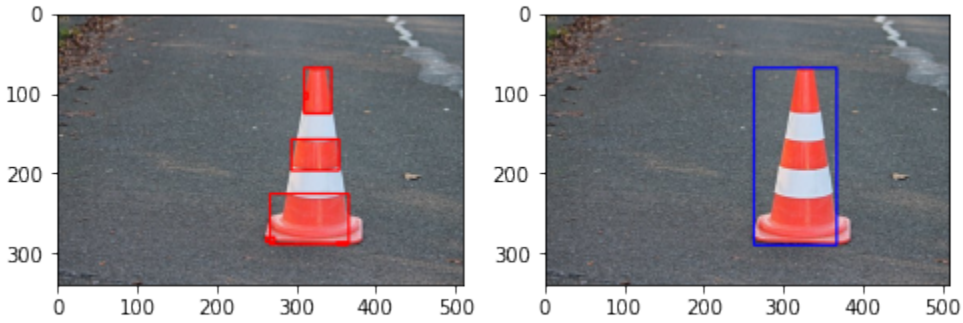

Figure 8: Detected blobs versus combined blobs

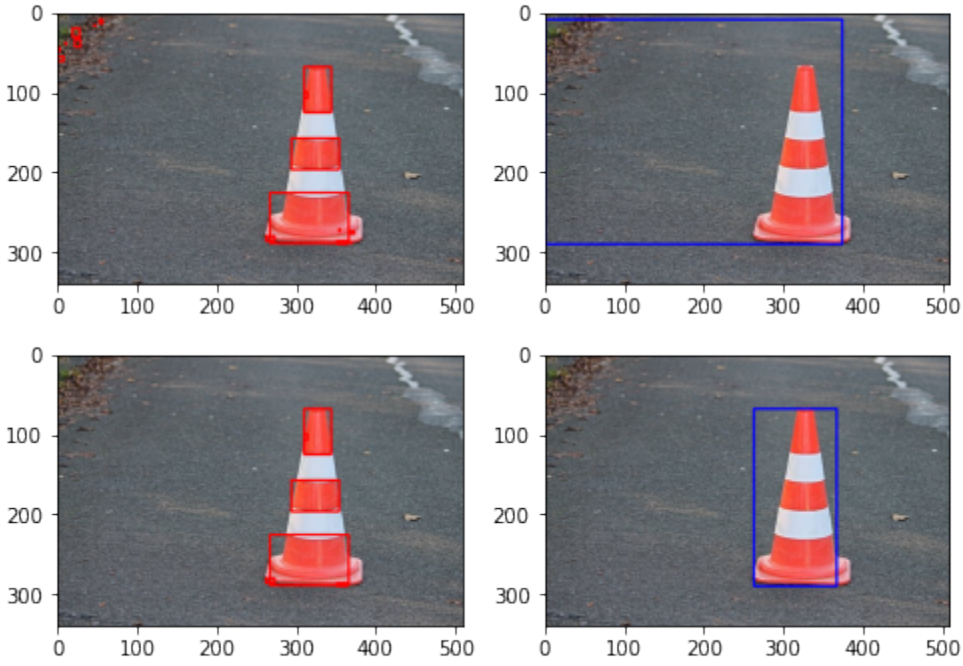

Figure 9: With noise (above) versus without noise (below)

To be able to follow this blob, a simple algorithm was developed to adjust the angular speed of the Juno. First the center of the blob was calculated, and then based on in which predefined region of the x-axis it was, a function was called adjusting the angular speed more or less. There are 5 regions, of which 1 is the center region. If the center of the blob is already in the center of the screen, the angular speed was set to zero. Depending on in which region (far left, left, right or far right), the angular speed was slightly changed or more heavily changed.

2.5.2 Juno vicinity lights

The same function detecting green blobs was modified to detect blue blobs as well. Instead of calculating the center of this blob, a toggle on function was called whenever the blue blob was detected 5 frames in row. If the light was on and the blob was not in detected 5 frames in a row, the toggle off function was called. Using the manufacturers own product API and the paho-mqtt API, messages were published from the developers laptop to a mqtt broker on a specific topic. A Shelly Plug-S is subscribed to this topic and therefore listens to the messages, relayed by the broker. The contents of these messages are HTTP commands for changing values in the plugs power settings.

3 Conclusion

Unfortunately, not everything which was developed in the preparation phase also worked on the Juno robots in Ahoy. A model based and behaviour based approach was developed for each robot, including communication between the two robots. On top of this, a homing feature was developed using computer vision. The same computer vision techniques were used to detect a different blob color as the other Juno to switch on the lights when the Juno was seen.
However, the LiDAR sensor could not get a sufficient enough range to be able to localize itself in SLAM. Therefore, some adjustments had to be made to the setup of the two Junos.

3.1 Juno 1

Since it was not possible to navigate in SLAM with the new LiDAR configuration, a behaviour based approach was required on Juno 1.
This Juno, a stripped down version of the original model (Figure 10), had the LiDAR sensor mounted close to the ground in front, getting as much range as possible while being low enough to the ground to also detect the hay bales.

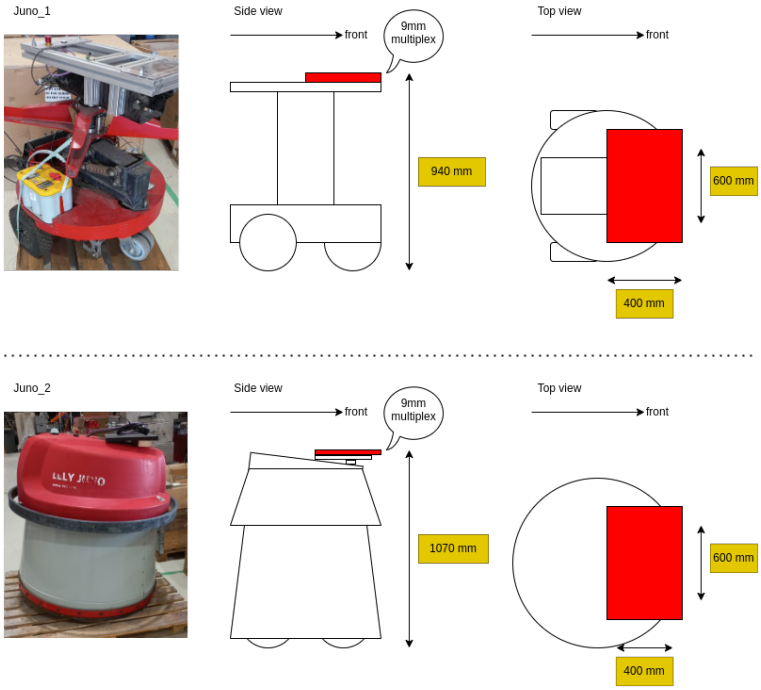

Figure 10: Stripped down Juno (Juno 1) versus regular Juno (Juno 2)

3.1.1 Problems

A problem with this LiDAR setup is that the center of the LiDAR is not the center of the Juno. This caused the Juno to turn too early in right turns, thinking the right side is already clear for the whole Juno. In reality only the front of the Juno passed the wall, while the rest was still alongside it.

3.1.2 Future improvements

Improvements can be made to this algorithm by hard coding a delay in a right turn and let the Juno drive forward for a longer time. However, to finetune this behaviour extensive testing would be required.

3.2 Juno 2

Needing to attach sonar sensors to the second Juno to be able to navigate, wedges were developed such that the sonar sensors were pointed downwards. This was needed because placing the sonars all the way on the bottom was not possible due to the power cables for the sensors not being long enough.

Unfortunately, there could not be made a sufficient distinction in distance between the side of the hay, and the ground or top of the hay bales. This required

us to give up on navigating with the second Juno, and instead only demonstrate our blob following algorithm on this Juno. The feature to lit the lights next to the barn environment was implemented on this Juno as well.

3.2.1 Problems

The problem with the blob following algorithm was that it was still very sensitive to noise. When there was green detected somewhere else in the image, the square would be drawn around this blob as well. This caused the square of the center to be in the wrong place and not correct the angular speed of the robot the right way.

3.2.2 Future improvements

A solution for this problem could be to only calculate the center of the biggest detected blob, or weight the center of the smaller blobs relative to the size of the blob.

4 References

Batalin, M. A., Sukhatme, G. S., & Hattig, M. (2004). Mobile robot navigation using a sensor network. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, *1*, 636–641.

Huang, T. (n.d.). Rplidar-a3 laser range scanner$_r$*obotlaserrangescanner*. https://www.slamtec.com/en/Lidar/A3

Jones, J., & Roth, D. (2004). *Robot programming: A practical guide to behavior-based robotics*. McGraw Hill Professional.

Master and client communication documentation. (n.d.). http://wiki.ros.org/ROS/Tutorials/MultipleMachines

Matarić, M. J., & Michaud, F. (2008). Behavior-based systems. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (pp. 891–909). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-30301-5_39

The Construct. [Online; accessed June 8, 2022]. (2022).

Yi-bo, L., & Jun-Jun, L. (2011). Harris corner detection algorithm based on improved contourlet transform. *Procedia Engineering*, *15*, 2239–2243.

Appendices

A ROS

Robot Operating System is a set of frameworks that allows us to communicate between different components of a robot environment using Python and C++. ROS is generally used in Ubuntu but can also be used in Windows. There are several distributions for ROS, however for this project, only ROS1 Noetic

(all hardware used in this project) and ROS1 Melodic (Juno robot) are used. As mentioned in Communication, ROS works by using topics, listeners and talkers to communicate information and data between nodes, which in turn are seperately run programs. This method of communication between different processes allows for easy programming of robot control, but can also be used to work with different aspects of robots, such as sensor input processing. There are a plethora of open source ROS packages and libraries on the world wide web which can be used for a variety of end goals, most important being the connection between sensors, robots and our control over such robots.

B Hardware

The hardware used for the preparation, testing and demonstration of our code is based on the Mirte navigation robot provided by Robohouse, the Turtlebot3 navigation robot provided by the UvA and the Juno navigation provided by Lely during the 3 day hackathon. Each of these robots use a differential drive motor for wheels. On top of this, each of these has been supplied with the required software to convert the hardware signals into easily usable software code. For each of these robots, the main processing units are an Orange Pi Zero, a Raspberry Pi 3B+ and our own laptops respectively. Each of these run a version of Ubuntu. The Mirte robot is supplied with an RPLidar 360 degrees laser range scanner, 4 HC-SR04 ultrasonic sensors, a Logitech C930e stereo camera and a MPU6050 gyroscope + accelerator. The turtlebot3 is supplied with a similar LiDAR and gyroscope + accelerator. The Juno robot contains NDA-classified hardware and as such only the USB CAN connection between the hardware and our laptops is available as information.

C Software

Both Mirte and Turtlebot3 are supplied with their own ROS packages with varying levels of usability and flexibility. Turtlebots are widely used in real world robot tests and as such contains far more open source packages that can be used for testing purposes. The Juno robot supplied to us contains a docker image with ROS1 Melodic with limited usability. It's sole goal is to communicate with our laptops for controlling the wheels. All other computations must be done on our laptops.

D Sensor hardware and software

The connection between sensor hardware and sensor software is called sensor streams. Sensor streams refer to the connection between the sensor inputs that our robot receives from the attached sensors and the stack. Software was set up to receive these software inputs. For our implementation, it is not required

to have any sensor inputs other than the LiDAR scanner for model-based navigation, since the laser scans will provide sufficient accuracy for path planning within the barn environment.
By default, most ROS packages and code that make use of LiDAR inputs are already linked to the standard scanner topic, while our LiDAR laser scanner has packages that send the inputs to the same topic, thus setting up the sensor stream is of relative ease. For behaviour based navigation, we use the HC-SR04 ultrasonic sensors, as they can provide us with enough sensor data to mimic a limited LiDAR laser scanner. This is required for the challenge as we are only allowed to use one LiDAR in total. The sonars use open source ROS packages that easily set up the software stream required for use with the sonar.

E Other open source ROS packages

The following links contain all the open source packages used during the project:
https://github.com/microsoft/ros_azure_iothub
https://github.com/Slamtec/rplidar_ros
https://github.com/engcang/HC-SR04-UltraSonicSensor-ROS-RaspberryPi
https://github.com/ros-planning/navigation https://github.com/tu-darmstadt-ros-pkg/hector_slam https://github.com/yoraish/lidar_bot/tree/wall-follow-example/src/ros/wall_follower_sim

F diagrams

ROS Smach FSM used for model-based route planning