Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

An Entertaining Demonstration of a Multi-Robot System

Bohumil Brož

Supervisor: prof. Ing. Tomáš Svoboda, Ph.D.
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Brož Bohumil** Personal ID number: **492348**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

An Entertaining Demonstration of a Multi-Robot System

Bachelor’s thesis title in Czech:

Zábavná demonstrace multirobotického systému

Guidelines:

The aim of the project is to prepare an entertaining demonstration of a multi-robot system. The result can be used in the promotional materials of the faculty, like open days or for various visitors. The work may exploit the existing multirobotic system developed for the DARPA SubTerranean Challenge [1]. The work should design an imaginative choreography with the aim of promoting the school or autonomous robotics at CTU FEE as interesting as possible. Preferably, it should be an interplay of at least two robots.
The result will often be used in an environment where it will not be possible to separate the robotic working space from the present spectators. An important part of the choreography must be the ability to deal with the people present in a way that is deemed socially accepted and safe [2,3]. The approach can be based on the concept of robot leader and robot monitor, see [4]. The work assumes working mutual communication between robots and possible real time coordination, however, it is possible to be inspired by work addressing possible collisions between robots and humans [5]. The estimation of movement vectors of the present spectators can draw from the latest results in the field of modeling the movements of a group of people [6,7].

Bibliography / sources:

[1] T. Rouček et al. System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA Subterranean Challenge. Preprint arXiv:2110.05911
[2] M. Shiomi, F. Zanlungo, K. Hayashi, T. Kanda. Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. In International Journal of Social Robotics, August 2014
[3] M. Emirler, H. Wang, B. Guvenc. Socially Acceptable Collision Avoidance System for Vulnerable Road Users Socially Acceptable Collision Avoidance System for Vulnerable Road Users. IFAC2016
[4] M. Saska et al. Formation control of unmaned micro aearial vehicles for straitened environments. In Autonomous Robots, March 2020
[5] S. Wang et al. Repulsion-Oriented Reciprocal Collision Avoidance for Multiple Mobile Robots. In Journal of Intelligent and Robotic Systems. 2022, 104:12
[6] K. Katyal et al. Learning a Group-Aware Policy for Robot Navigation. arXiv:2012.12291
[7] Y. Liu, Q. Yan, A. Alahi. Social NCE: Contrastive Learning of Socially-aware Motion Representations. ICCV 2021

Name and workplace of bachelor’s thesis supervisor:

prof. Ing. Tomáš Svoboda, Ph.D. Vision for Robotics and Autonomous Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **03.02.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

prof. Ing. Tomáš Svoboda, Ph.D. Supervisor’s signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to thank my supervisor prof. Ing. Tomáš Svoboda, Ph.D. for valuable advice and guidance when writing this thesis. I would also like to thank Mgr. Martin Pecka, Ph.D. for consultations on ROS and the robots.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20.5.2022

.....
Signature

Abstract

The aim of this work is to design a system for skid steer robots, that allows them to follow a given trajectory in formation. The system can be used in environments with obstacles that are blocking the path of the robots. The purpose of the movement is purely aesthetical, the intended use of the system is for various presentations, e.g. on an open day. Virtual leader approach is used to control the movement of the formation. Robots use LIDAR to detect obstacles and try to avoid them with the usage of virtual force field, if the obstacle is too close, the robot stops. A simple language for the choreography description was created. Purely geometrical 2D simulator was created in order to design and verify new scenarios. The proposed solution was verified by simulation and on real robots. The results show, that virtual leader is a viable way of controlling formation of mobile robots, even when there is a transmission delay in the communication between the robots.

Keywords: mobile robots, formation of robots, virtual leader, virtual structure, virtual force field, ROS

Supervisor: prof. Ing. Tomáš Svoboda, Ph.D.

Abstrakt

Cílem této práce je vytvořit systém, který umožní jízdu smykem řízených robotů ve formaci po zvolené trajektorii v prostředí, které může obsahovat překážky. Cíl pohybu je čistě estetický, systém by měl sloužit k různým demonstracím, jako například při dni otevřených dveří. K řízení pohybu formace je využit přístup virtuálního lídra, roboti pomocí LIDARu detekují překážky, objedou je díky virtuálnímu silovému poli anebo zastaví, pokud je překážka příliš blízko. Čistě geometrický 2D simulátor byl vytvořen pro účely návrhu a testování nových choreografií. Je navržen jednoduchý jazyk pro popis žádané choreografie. Dosažené výsledky jsou ověřeny simulací i nasazením na skutečných robotech. Výsledky ukazují, že přístup virtuálního lídra je vhodný pro tento účel a že odstraňuje i problémy způsobené zpožděnou komunikací mezi roboty.

Klíčová slova: mobilní roboti, formace robotů, virtuální lídr, virtuální struktura, virtuální silové pole, ROS

Překlad názvu: Zábavná demonstrace multirobotického systému

Contents

1 Introduction **1**

2 Description of the used robots **3**

3 Description of the system **7**

4 Motion control **11**

4.1 Limitations of the formation movement 11

4.2 Formation control 12

4.3 Control of robot's motion 14

5 Safety **15**

5.1 Failure detection 15

5.2 Obstacle detection 15

6 The language for choreography description **17**

7 Testing and results **23**

7.1 Simulation 23

7.2 Results 26

7.2.1 Simulation 26

7.2.2 Real robots 37

8 Conclusions **41**

Bibliography **43**

A List of attachments **45**

Figures

2.1 The position of the `base_link` frame. The x axis is red and the y axis green, the z axis is pointing upwards from the figure. The angular and linear component of the robots movement are depicted. 4
2.2 HUSKY robot 5
2.3 TRADR robot 5

3.1 All nodes and the topics they use to communicate 8
3.2 The functioning of network_test node. Duplicate received messages are omitted. The box start means, that the program was launched. Published messages are grey, received are blue 8
3.3 The process of the synchronization. This figure is continuation of the figure 3.2. Therefore the robot R1 starts publishing first, followed by R2 and R3. Duplicate received messages are omitted. Times T_1, T_2 and T_3 are the times proposed by the corresponding robots, T is the last proposed time. Sync messages are published at higher frequency, only the important sync messages are displayed. Note that the robot R3 receives the sync messages from the other robots before the `all_ready` signal. The subscriber is active from the time when the program is launched. The network test phase makes sure, that all sent sync messages are delivered to other robots. Robots R1 and R2 had a verified connection to R3 before the network test on R3 ended, therefore they could start publishing sync messages, these messages were properly delivered to R3. R3 starts publishing the sync messages after its `all_ready` signal is true, at that time, it is the last remaining robot to sync, thus it immediately starts waiting. R1 and R2 receive the sync message from R3 with some delay, but as long as the time delay is reasonably small (smaller than the waiting time of four seconds), the synchronization works correctly. . . . 9

6.1 The trajectory described in scenario and the trjectory of the virtual leader. Red axis is the x axis and the green is y. The depicted coordinate system is the `base_link` of the virtual leader. The frame with full lines is the current position and the dashed line is the new desired position of the virtual leader. The picture on the left side shows the trajectory described in the scenario. There are two waypoints - virtual leader is currently in the first one, the second one is the new desired position. The picture on the right side shows the real trajectory of the virtual leader. It does not turn on spot and thus deviates from the prescribed trajectory between the waypoints. 18
6.2 The two movement instructions. Red axis is the x axis and the green is y. The depicted coordinate system is the `base_link` of the virtual leader. The frame with full lines is the current position and the dashed line is the new desired position of the virtual leader. The picture a) shows the instruction *move*. Rotation is applied first, translation is done afterwards. The picture b) shows *circle* instruction. The centre of the arc is placed on the y axis of `base_link` at the position r. The virtual leader should rotate around the centre by the specified angle Θ. 19

7.1 The modified `turtlesim`. Trajectory of each robot is shown with different color, obstacles are drawn with black line. 24
7.2 The simulated LIDAR from 2D simulator for the same world as in figure 7.1. 25
7.3 The graphical interface of the 3D simulator. 25
7.4 The simulated LIDAR from 3D simulator for the same world as in figure 7.3. 25
7.5 Speeds required by the controller and distances for the first test scenario. The inner robot almost stops while turning, because the virtual leader is too close. The distance between the robots decreases while turning, because the robots are not following the leaders trajectory perfectly (they are allowed to do it, this leads to better rigidity of the formation). Even though the distance to virtual leader deviates from the desired one while turning, the robots are able to return to the desired distance shortly after the corner. . . 28
7.6 The trajectory of the robots for the first test scenario is not perfect, the virtual leader is turning with finite velocity and thus is not on the desired trajectory. The robots follow the leaders trajectory and that's why the part after the turn is not straight but slightly curved - the virtual leader is slowly returning to the desired trajectory. The inner robot is closer to the other robot at the corner exit than it should, therefore, it is turning away from the formation at the corner exit. 29

7.7 Speeds required by the controller and distances for the second test scenario. The inner robot slows down and the outer speeds up in the arc, but the speed difference is not big enough. The outer robot lags behind the inner robot (from time of 52 seconds to 75 seconds), but the difference is then lowered in the straight section of the trajectory. Formation splits at the time of 78 seconds (sudden change of distance to the virtual leader). Both robots follow their virtual leader well after the formation splits. The distance decreases while turning, but returns to the expected value after the turn. 30
7.8 Trajectory of the robots for the second scenario. The result is as expected with the same shortcomings as in the first test. 31
7.9 Speeds required by the controller and distances for the first test scenario in 3D simulation. The inner robot is capable of better following the virtual leader than in the 2D simulation. Because of that the distance between the robots is increasing this time. Robots again return to the desired position shortly after the turn. 33
7.10 The trajectory is more deformed than in the 2D simulation but still acceptable. The inner robot again has to turn away from the formation at the corner exit, because it got too far while turning. 34
7.11 Speeds required by the controller and distances for the second test scenario. The distance between the robots in the circular arc is closer to the desired one than in the 2D simulation. As a result, the outer robot lags behind the inner robot less. 35
7.12 The resulting trajectory looks more or less the same as the one from the 2D simulation. Trajectories of both robots are smooth and match the scenario quite well. 36

Chapter 1

Introduction

Formations of robots have wide range of use. They can be used for scientific research, each robot carries different measuring device and certain distance is required between the robots. It is also possible to use formations of robots in rescue missions where they are searching for survivors in a dangerous area, arrangement of the robots increases coverage of the area by sensors with limited range of each robot. Robots could be used to autonomously transport large payloads. In this case, robots have to move synchronously to keep the load evenly distributed across all robots. Lately, large formations of drones have also been used to create various shapes on the sky with light sources they are carrying.

Various approaches can be used to control a formation of robots. One of them is leader-follower, used in [1],[2],[3]. One of the robots is designated as a leader and the other robots are following it. The main disadvantage of this solution is dependency on one central node. If the leader fails, the whole formation stops. Successful control also relies on fast and reliable communication. If some messages are lost, leader position perceived by the follower might be different from the real one. This could lead to oscillations of the followers - they are on the correct position but some messages with leader position were lost and the robots therefore continue to drive forward even though they should stop. If the messages are delayed robots react to something that already happened. This causes unsynchronized movement of the formation, because the delay of the message could be different to each robot. As a result, the leader-follower approach is feasible only if communication between robots is flawless or if the formation shape does not have to be perfectly maintained.

The second approach is virtual structure/virtual leader [4], [5], [6], [7]. The formation is considered as one rigid body that moves according to the assignment. The structure has a reference point (the virtual leader) and each robot has a desired distance to the reference point. One of the advantages is, that frequent communication between the robots is not required, each robot can compute the current position of the reference point on its own. The computation can be done by one central node, but in the case of failure, this node can be replaced by another one immediately. Even if one of the robots fails, all other robots can continue to follow the prescribed trajectory.

The environment might contain several obstacles. When one of the robots is blocked by the obstacle, it has to deviate from the desired trajectory to avoid the obstacle [8], [4], [5]. Virtual repulsive field can be used to steer the robot around the obstacle [9], [10]. Each point detected by the robot's sensors (LIDAR or ultrasonic) creates a small repulsive force. These forces are summed and the resulting repulsive force is added to the pulling force of the target. Their sum then determines the desired heading of the robot's movement. Another way of avoiding obstacles is to use planning to create a collision free path [11], [12].

The aim of this work is to design algorithms for an existing multi-robot system [13], [14], that would allow movement of multiple robots in formation along a given trajectory. The main target is aesthetically pleasing and safe movement of the robots. The system can be used in environments with lot of obstacles, that could get in the way of one of the robots. Robot has to react to these obstacles in a safe but entertaining way. When obstacle blocks path of the robot, it speaks and starts beeping to alarm the spectator. The movement of the formation is described by a simple language that was created for this work. Virtual leader is used to control the movement of the formation. Each robot has its own virtual leader and it moves according to a scenario to maintain the desired distance to the virtual leader. It is possible to change the formation shape while the robots are moving and even split or merge the formations.

Chapter 2

Description of the used robots

The algorithms were designed for an existing multirobotic system of Department of Cybernetics. The system consists of multiple different robots. The first one is built on chassis HUSKY from company Clearpath Robotics (figure 2.2). It is 99 cm long and 67 cm wide, the maximal linear speed is 1 m/s.

The second robot is tracked, its name is TRADR (figure 2.3). It is 73 cm long and 60 cm wide. TRADR has additional tiltable tracks at the front and rear (called flippers). These tracks are in their default position folded on the side of the robot body. This robot can reach linear speed of only 0.5 m/s. Both robots can rotate rather quickly, they are capable of reaching angular speeds in excess of 180 degrees/s. Both robots can only move forward/backward and rotate around their centre. Both robots are equipped with 3D LIDAR and 5 RGB cameras, that cover the whole surroundings of the robot. HUSKY uses ROS Noetic and Python 3, whereas TRADR uses ROS Melodic and Python 2. Therefore, the code of this project has to be compatible with Python 2 and 3.

Wifi is used for communication between robots. Nimbro network is used to share topics and services across the robots.

The root of robot's TF tree is frame `map`. This frame is created as a combination of LIDAR data and odometry. Child of `map` frame is frame `map_fast`. This frame is published at higher frequency, but it is less accurate. Robot body is represented by frame `base_link`. This frame is based in the center of the robot body (see figure 2.1). The x axis is pointing to the front of the robot, the y axis to the left. Its children are frames that describe different parts of the robot, for example LIDAR, flippers, all of the cameras.

The LIDAR is capable of registering objects, that are at least 30 cm away from it. Point clouds are published at frequency of 10 Hz as sensor_msgs/PointCloud2 on topic `/points`. This point cloud contains points, belonging to parts of robot that are visible for the LIDAR. These points could cause problems for obstacle detection. This issue is solved by a filtered point cloud, that is published on topic `/points_filtered`. This point cloud has not only removed robot body, but the overall number of points is reduced too. LIDAR data on topic `/points` lags approximately 300 ms behind reality. It is necessary to factor this delay into obstacle detection. Objects that appear far from the robot on LIDAR image can be much closer in reality (HUSKY

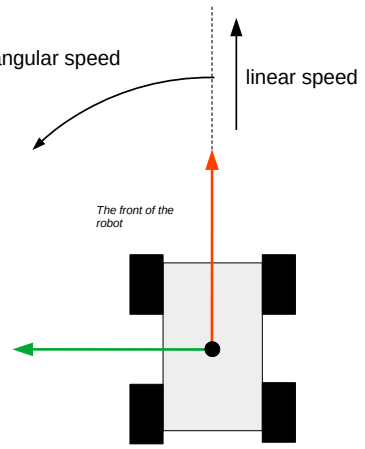

Figure 2.1: The position of the `base_link` frame. The x axis is red and the y axis green, the z axis is pointing upwards from the figure. The angular and linear component of the robots movement are depicted.

is capable of travelling at speed of 1 m/s, therefore an object could be 30 centimetres closer to the robot, than the latest LIDAR image shows).

Robots accept movement commands on two topics. Messages on these topics have to be published with frequency of at least 10 Hz, otherwise they would be ignored. Topic `/nav/cmd_vel` has lower priority than `/cmd_vel`. The code of this project uses the first topic. The user can take over the control with the gamepad that is used to manually control the robot (its input is published on `/cmd_vel`). The messages are of type geometry_msgs/Twist and have only two valid elements – linear.x and angular.z (other elements are ignored). The linear component is the speed of the movement in the direction of the x axis of `base_link` (movement forward or backward) and the angular component is the speed of rotation around the z axis of `base_link`. The two components of the robot's movement are explained in the figure 2.1.

Figure 2.2: HUSKY robot

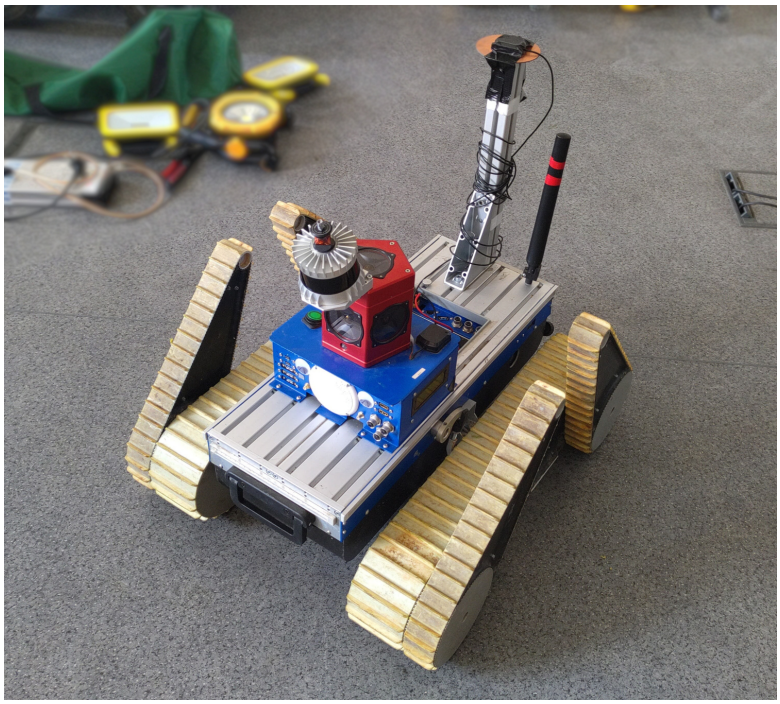

Figure 2.3: TRADR robot

Chapter 3

Description of the system

The program was created in Python with the usage of rospy. The program is started by a launch file `formation.launch`. This results in three running nodes, their connection is shown on figure 3.1. The first one is called `network_test`. Its goal is to check, whether the program is running on all robots and if the connection to all robots is functional in both directions. The node creates a list of all robots in the formation according to the provided scenario. The node publishes messages of type std_msgs/String to topic `/sync` with frequency of 10 Hz. These messages contain the role of the sender. The node is also subscribed to this topic. If message from another robot is received, the role of the robot is added to its end and the message is published again. If node receives back an altered message, the role at the end is removed from the list of robots. Once all robots are removed from the list a std_msgs/Bool message with value true is published to topic `/all_ready`.

The second node is `formation_member`. This is the main node that controls the virtual leader and the robot's movement. Topic sync is used to synchronize the movement of the robots. Further description of the synchronization is in chapter 4.2. The synchronization starts, only after true was published to `/all_ready`.

Node `formation_member` launches (through the Python roslaunch API) the third node – `obstacle_detection`. This node processes the data from LIDAR. It has three outputs – topics `/stop_front` and `/stop_rear` of type std_msgs/Bool and `/obstacles` of type sensor_msgs/PointCloud2. One of the arguments (optional) of the node is robot name. It is necessary to use this argument for simulations, because all robots are running on a single ROS master. All three topics then have the robot's name as a prefix. The robot's name does not have to be the same as the role.

Launch file also sets two parameters - `/length` and `/width`. These parameters describe the size of the robot. The value is set according to the environment variable `HOSTNAME`. If it matches one of the robots (`HOSTNAME` ctu-robot or husky-robot), the parameters are set to the dimensions of the particular robot, otherwise the values are set to the size of the robots in 2D simulator (length is 1.2 meter and width 0.6 meter). These parameters are required for correct functioning of obstacle_detection node.

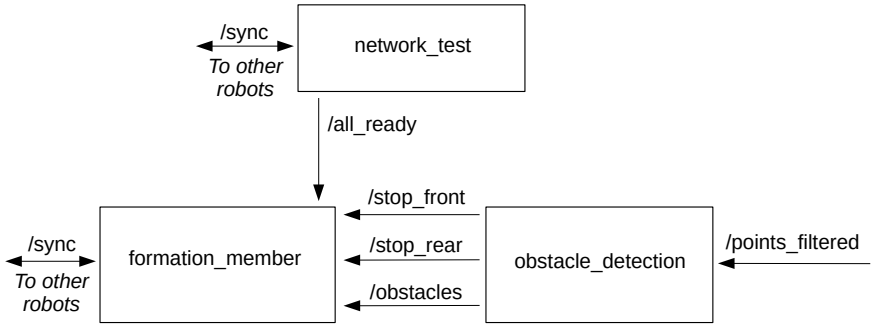

Figure 3.1: All nodes and the topics they use to communicate

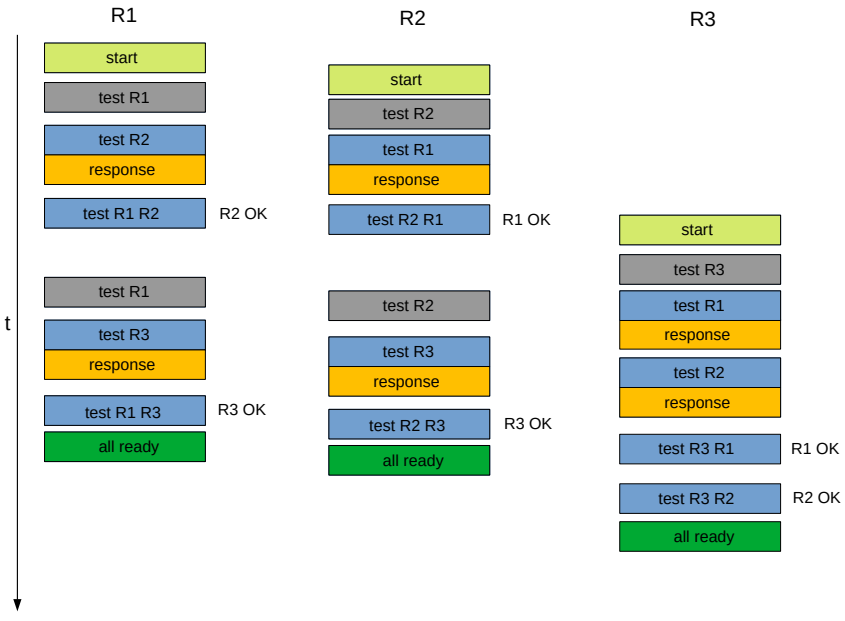

Figure 3.2: The functioning of network_test node. Duplicate received messages are omitted. The box start means, that the program was launched. Published messages are grey, received are blue

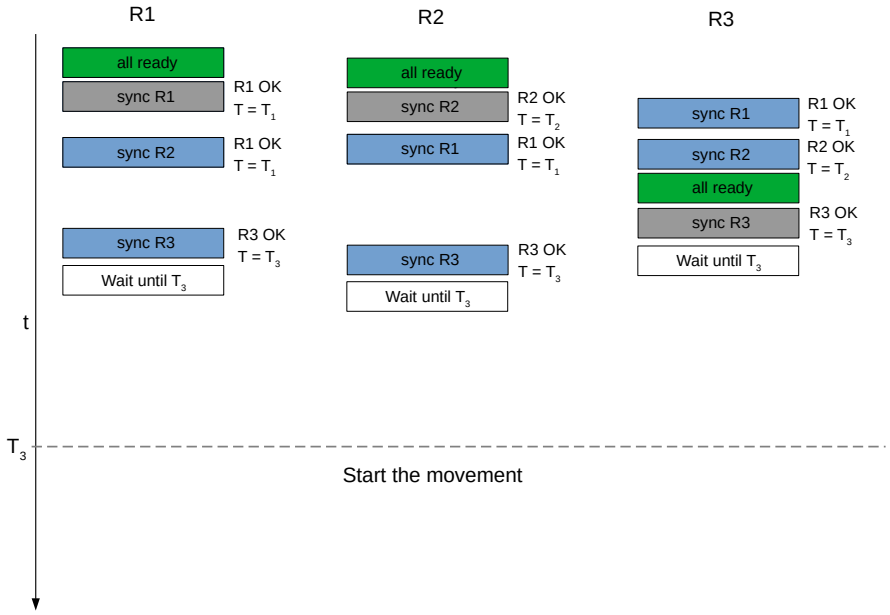

Figure 3.3: The process of the synchronization. This figure is continuation of the figure 3.2. Therefore the robot R1 starts publishing first, followed by R2 and R3. Duplicate received messages are omitted. Times T_1, T_2 and T_3 are the times proposed by the corresponding robots, T is the last proposed time. Sync messages are published at higher frequency, only the important sync messages are displayed. Note that the robot R3 receives the sync messages from the other robots before the `all_ready` signal. The subscriber is active from the time when the program is launched. The network test phase makes sure, that all sent sync messages are delivered to other robots. Robots R1 and R2 had a verified connection to R3 before the network test on R3 ended, therefore they could start publishing sync messages, these messages were properly delivered to R3. R3 starts publishing the sync messages after its `all_ready` signal is true, at that time, it is the last remaining robot to sync, thus it immediately starts waiting. R1 and R2 receive the sync message from R3 with some delay, but as long as the time delay is reasonably small (smaller than the waiting time of four seconds), the synchronization works correctly.

Chapter 4

Motion control

4.1 Limitations of the formation movement

Since the used robots can't move sideways, the possibilities of formation movement are limited. It is impossible to maintain completely rigid formation. In a sharp corner, at least some of the robots in the formation would have to move sideways to keep the desired position.

One of the possible solutions to this issue is to allow the robots to move further away from the desired trajectory. This allows robots to cut the corners and better maintain the formation shape (then the corner is not that sharp and the difference between actual position and the desired position is smaller). This change brings its own problem. The robots can follow trajectory that barely resembles the desired one. If the desired trajectory has two 90 degree turns right after each other, robots will instead move in one long arc. If the scenario has many direction changes in short time, the deviation from the prescribed trajectory is going to increase with each turn. In the end, the real trajectory could end up completely different than was the intention. Therefore, this solution is only acceptable, when the target is to move the robots in the desired formation to target, without respect to their trajectory. However, such use contradicts the aims of this work, because the aesthetic intention of the user is completely ignored.

The approach used in this work is to allow the formation to alter its shape while moving. The formation changes its length (along the x axis of frame `base_link`), but keeps the distance in the perpendicular direction. The robots in the second row of the formation get a little bit closer to the first row of robots, while the first row is turning, and the distance slightly increases, while the second row is turning.

The radius of the corner has significant impact on the way how the robot on the inside of the corner moves. If the radius is bigger than the distance in the direction of y axis between the robot and the virtual leader, the robot can drive forward while turning. On the contrary, if the radius is smaller, the robot has to reverse to maintain the desired distance to the virtual leader and the other robots in the same row. To allow the robots to reverse, the lengthwise distance between the robots has to be greater than the smallest possible. Unless the distance is big enough, the robot's movement will be

blocked by the robot behind. This could also affect the robot on the outside of the corner. Provided that the transverse distance between the robots is small, the inside robot could be partially standing in the way of the outer robot (the robot on the inside drives at first further into the corner than it should be at the corner exit and then moves backwards).

Another limitation is the linear speed of the robots. The formation has to move slower than the maximal speed of the slowest robot in the formation. The user has to provide the maximal speed of the slowest robot as one of the parameters (the entered speed should ideally be slightly lower). The virtual leader then moves at 70 % of the given speed to allow even the slowest robots to sufficiently speed up in order to close the gap.

4.2 Formation control

Requirement for correct functioning of the designed algorithms is, that all robots are on a plain surface, with minimal differences in z coordinate. This enables the control to be done only in two dimensions.

Communication over Wi-Fi is not adequately reliable, the delay between one robot sending a message and other robot receiving it is not constant and not negligible. If one of the robots was designated as leader of the formation and the other were following it, they would receive the information about leader's movement with a delay. Which means that they would react to something that already happened. This means that the formation movement would appear chaotic to the observer (transmission delay could be different to each robot).

Therefore, all robots in the formation follow a virtual reference point (virtual leader) whose position is calculated by each robot independently. The scenario defines a desired distance (along the x and y axis) from the virtual leader for every robot. The virtual leader has two frames, the `/role/vl_map` and its child `/role/virtual_leader`. The frame `virtual_leader` corresponds to the frame `base_link` of the robot.

In order to unify the position of virtual leader on all robots throughout their movement, the robots have to be synchronized at the beginning. Each robot periodically publishes message of type std_msgs/String on topic `/sync`. These messages contain the role of the robot (its name in the scenario) and proposal of time when the formation will start to move (current time acquired by `rospy.Time.now()` incremented by four seconds to accommodate any transmission delay). If robot receives message with one role for the first time, this role is removed from the list of robots, that are waiting for synchronization (the list is assembled from the scenario) and the proposed time is saved as the last valid proposal. When the last robot is removed from the list, robot waits until the last proposed time and then the virtual leader starts to move. All robots in the formation have internet connection and all have the same clock thanks to NTP. Each robot uses the same controller to control the virtual leader movement and thus all the virtual leaders will be at the same place at all times.

The usage of virtual leader instead of a physical one allows the formation to easily split and join together. Only thing that changes is the required distance to the virtual leader for each of the robots. It is also possible to create new virtual leader at a different position than the old one, this is beneficial when the formation is splitting. The best placement for the virtual leader is in front of the formation and in the middle in the direction of y axis. This ensures that all robots use roughly the same trajectory. Provided that the virtual leader was placed on the left side of the formation and the formation was to sharply turn left, the robots at the right side of the formation would have to follow a much longer trajectory. Since the linear speed of the robots is limited, the formation would have to noticeably slow down to allow the robots on the right side to stay on the right position throughout the turn. Still, the outer robots might not be able to speed up enough and the formation would fall apart.

The controller for virtual leader's motion is:

$$linear = \min(\max(1.5 \cdot G_x, 0.1), 0.7 \cdot v_{max}) \tag{4.1a}$$

$$angular = 2 \cdot \mathrm{sgn}(G_y) \cdot \alpha \tag{4.1b}$$

Saturation for angular speed is used, because the rotation of the formation can't be too fast, otherwise robots will be unable to follow such a sharp corner. The limits are $-\pi/2$ and $\pi/2$. G is the next waypoint for the virtual leader and G_y its y coordinate, α is the angle between the x axis of `base_link` and the waypoint, it is always a positive number, the correct direction has to be chosen separately (as the sign of y coordinate of the target). v_{max} is the maximal speed entered by user.

The controller for linear speed is designed in a way, that the virtual leader is always moving. This prevents the formation from turning on spot (that is a movement that skid steer robots are not capable of following). The virtual leader is not allowed to drive backwards.

Virtual leader uses separate controller for circular arcs, because in this situation, the trajectory of the virtual leader is not the shortest possible one as is the case with the controllers 4.1a and 4.1b. The equation 4.2 describing velocities for motion on circular arc is used to get the ideal linear speed in the linear speed controller (equation 4.3a). There is a tolerance for reaching the waypoint, therefore the leader could be slightly off the starting point of the arc. Angular speed of the movement is controlled (equation 4.3b) to ensure that the leader reaches the end of the arc.

$$v_{id} = \omega_{id} \cdot r \tag{4.2}$$

$$linear = \omega_{id} \cdot |r| \tag{4.3a}$$

$$angular = \mathrm{sgn}(r) \cdot \omega_{id} \cdot (1 - 5 \cdot (r - r_{real}) \tag{4.3b}$$

r is the radius of the arc from the scenario (can be negative, because it defines the placement of the arc centre - positive means centre on the left

side of the robot, negative right side), ω_{id} is the desired angular speed from the scenario and r_{real} is the current radius (current distance to the centre of the arc).

4.3 Control of robot's motion

Each robot in the formation creates its own queue (`deque` from the library collections) of virtual leader positions. Instead of following directly the virtual leader, robots follow the path from the queue.

Linear speed of the robot is given by the deviation between the current distance to the virtual leader and the desired one. The direction of the movement is given by the position of the queue head. Angular speed is calculated from the angle between the x axis of `base_link` frame and the desired position of the robot (the head of the queue). Heading of the virtual leader in the target position is not taken into consideration. The queue is filled at a frequency of 10 Hz, as a result, the individual points in the queue are close to each other. Provided that the robot reaches the point from the queue, it has to select the same orientation as the virtual leader, because that is the heading, that leads to the next point on the trajectory. The angle to the trajectory point is always the smallest possible one, i.e. if the point is behind the robot, the angle will be the one that allows the robot to reach the point by reversing.

If the virtual leader gets too far away (the current distance is double of the desired), the queue is cleared and robot starts following directly the virtual leader. This means that robot takes the shortest path to the virtual leader and has the best chance to close the gap.

The resulting controller has the form of two equations 4.4a and 4.4b

$$linear = \text{sgn}(G_x) \cdot 0.7 \cdot v_{max} \cdot |1 + 3 \cdot (d_{act} - d_{req})| \tag{4.4a}$$

$$angular = 3 \cdot \text{sgn}(G_x \cdot G_y) \cdot \alpha - 2 \cdot \text{sgn}(F_x) \cdot \alpha_2 \cdot |F| \tag{4.4b}$$

d_{req} is the required distance to virtual leader and d_{act} is the current one. Robots are allowed to go backward, to determine the correct steering angle direction, $sgn(G_x \cdot G_y)$ has to be used. Target is either the virtual leader or the head of the queue. If the robot is closer to the queue head than 30 cm, new point is popped from the queue (if the queue is not empty). If there is no point in the queue, robot follows directly the virtual leader. α_2 is the angle of the virtual repulsive force of obstacles, $|F|$ is the strength of the virtual force. α is the smallest angle between x axis of `base_link` and the target (the angle itself or $\pi - \alpha$).

$0.7 \cdot v_{max}$ is the maximal speed of the virtual leader. This value is multiplied by $|1 + 3 \cdot (d_{act} - d_{req})|$, it is a non-negative number that is in ideal conditions (robot is following the virtual leader with the given gap) equal to one. If the gap between the robot and the virtual leader is smaller than the desired one, the value is lower than one and robot moves slower than the virtual leader (the gap is increasing) and vice versa.

Chapter 5

Safety

5.1 Failure detection

The program checks the availability of LIDAR data and all the necessary transformations. If one of the transformations was unavailable for some amount of time, control of the robot's movement would be impossible. If the data from LIDAR is lost, robot is unable to detect obstacles in its surroundings and a collision is possible.

Provided that three consequent exceptions in tf_buffer lookup occur or the LIDAR data is unavailable for more than 0.3 seconds, robot gets into error state. It is impossible to recover from this state, because any such error would by caused by a hardware failure or a termination of one of the nodes, and these errors can't be fixed without the response of the user. Fixing the error takes some time and once the issue is resolved, the gap to the rest of the formation would be too big.

5.2 Obstacle detection

As already mentioned in the chapter 2, LIDAR lags behind the reality. Therefore, robot has to react to any obstacles in advance. Signals `/stop_front` and `/stop_rear` are set to true, when the obstacle is closer to the robot than 55 cm (at the front or at the rear). If the distance was smaller, robot would be unable to stop on time. These signals prevent the robot from moving in the given direction.

The output point cloud `/obstacles` of node obstacle_detection is created from the LIDAR data. First of all, an area of 4 by 4 meters around the robot (robot is in the centre, it is 2 meters to each side of `base_link` origin) is cut out from the LIDAR image, only points lower than 1.5 meter are accepted, other points are not blocking the robot's movement (for example the ceiling of the room). The image is not perfect. The first points of floor around the robot have wrong height, the ground is slightly raised around the robot. Therefore, all points that are higher than 10 cm in `base_link` are considered as an obstacle. The output point cloud has only two dimensions, the z coordinate of all points is set to zero.

The main purpose of the point cloud is obstacle avoidance by virtual force field. Obstacle avoidance is not suited for cramped conditions. Because of the robot's dimensions, the obstacle has to create noticeable force even from a distance of 1 to 1.5 meters (to overcome the pulling force of the target). If the robot is to move alongside some obstacles, this repulsive force will force greater distance between the robot and the obstacles. It is not possible to reduce the distance at which the force begins to affect the robot, it has to prevent the robot from turning into an obstacle, even when the desired position lies in it.

If the robot has to stop because of an obstacle, `sound_play` [15] is used to say sentence "Obstacle is too close, I have to stop", robot then beeps every 3 seconds, until the way is clear again. Fixed obstacles are not expected to be in the way of the robot (the scenario should be planned so that there is a clear path for the robots. Only expected obstacles are the spectators). The beeping should warn the spectator that he is blocking the robot.

Chapter 6

The language for choreography description

The choreography and the formation shape are described in text files. The whole scenario begins with a list of the used robots. Each robot has its own unique role – an identifier that is used throughout the scenario to describe all robots affected by the instruction. The role has to be without spaces and semi-colons. The next part of the file is optional, it is definition of different groups of robots. Groups are required if the formation is split and each new formation has its own instructions. The group identifier has to be unique and without spaces. It is not connected with the formation shape, only with the robot roles in the group. The same identifier can be used for two or more separate formations, if they consist of the same robots and if they don't occur at the same time (i.e., if the formation splits, joins back together and then splits again into the same groups of robots).

The rest of the scenario describes the different formation shapes and their movement. The first instruction has to be formation description. There are four supported formation shapes.

- *row* - robots are side by side, origins of their `base_link` frames are in one line
- *column* - robots are behind each other, origins of their `base_link` frames are in one line
- *triangle* - one robot is in the first row, the others are in the second row. Both rows are centered - the robot in the first row is positioned in front of the center of the second row (the middle robot or the middle of the gap between two robots in the middle of the row).
- *square* - designed for four robots in two rows. It is possible to set the distance between the rows and the distance between the robots in one row (can be different for each row).

The notation for each of them is the same. Firstly, the arrangement of the robots is described (roles are used here). Each row of robots is separated by semi-colon and the robots in one row are separated by comma. Next, the distances between the rows of robots are set. Finally, the spacing between robots in one row is defined, rows are again split by semi-colon. All values

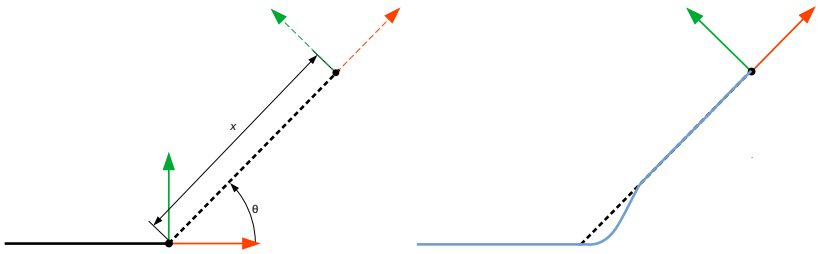

Figure 6.1: The trajectory described in scenario and the trjectory of the virtual leader. Red axis is the x axis and the green is y. The depicted coordinate system is the `base_link` of the virtual leader. The frame with full lines is the current position and the dashed line is the new desired position of the virtual leader. The picture on the left side shows the trajectory described in the scenario. There are two waypoints - virtual leader is currently in the first one, the second one is the new desired position. The picture on the right side shows the real trajectory of the virtual leader. It does not turn on spot and thus deviates from the prescribed trajectory between the waypoints.

are in metres and decimal point is used (the values in one row are separated by comma). All three parts of the description are separated by space.

The movement of the formation is described by two instructions (see figure 6.2) The instructions decsribe the formation's trajectory as a series of waypoints. The first instruction is *move* x Θ. The parameter x is distance in the direction of x axis of `base_link` and the second one rotation around z axis in degrees. The waypoint is created by rotation around the z axis and then translation in the direction of new x axis. The direction of the rotation is defined by the sign of the angle, positive angle means counter clockwise rotation (robot turns left). Robots and virtual leaders do not fully follow this trajectory, they are not turning on a spot, they turn while driving forward/backward instead. The resulting trajectory is longer and curved (see figure 6.1.

The other instruction is *circle* r Θ ω. This is movement along a circular arc with the given radius r and central angle Θ. The centre is placed on the y axis of `base_link` when the robot is at the beginning of the arc. The radius can be a negative number, the direction of rotation is decided by the

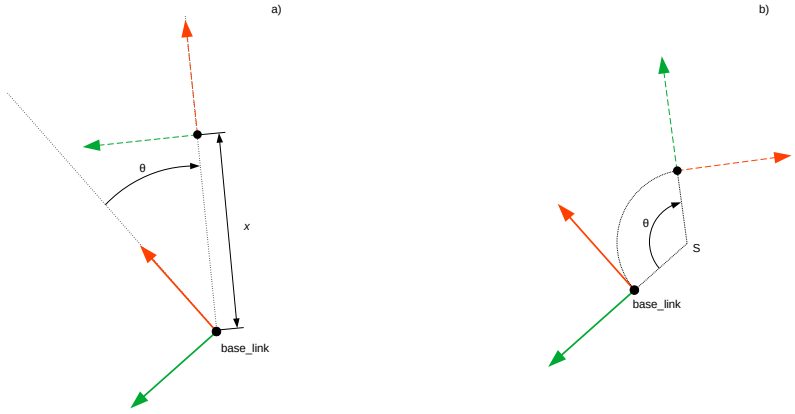

Figure 6.2: The two movement instructions. Red axis is the x axis and the green is y. The depicted coordinate system is the `base_link` of the virtual leader. The frame with full lines is the current position and the dashed line is the new desired position of the virtual leader. The picture a) shows the instruction *move*. Rotation is applied first, translation is done afterwards. The picture b) shows *circle* instruction. The centre of the arc is placed on the y axis of `base_link` at the position r. The virtual leader should rotate around the centre by the specified angle Θ.

sign (positive means arc on the left side of the robot). The central angle is in degrees and has to be positive. The third parameter is the desired angular speed. It might be impossible to reach the desired angular speed, because the angular and linear speed of circular motion are bound by equation 6.1

$$v = \omega \cdot r \tag{6.1}$$

If the necessary linear speed is too high, the linear and angular speed will be halved until the linear speed is lower than the maximal speed of virtual leader.

The radius of the arc has to be bigger than the maximal distance between the robots and the virtual leader in the direction of y axis. If the radius was smaller, the robot with greater distance would have to revers to maintain the proper position. The resulting movement of the formation would not resemble circular arc, it would be similar to the turns created by the *move* instruction. The reversing robot would also block any robots in the rows behind it, this could lead to the formation falling apart.

If the formation shape has to be changed, new formation description is inserted into the scenario. When robots reach this instruction, their desired

distance to the virtual leader is changed to the new value. It is sometimes beneficial to move the virtual leader to a different place. For example, when the formation splits into two, better results will be reached, if the virtual leader is again placed in the middle of the subformation in the direction of the y axis. Reset of the virtual leader can be done by instruction $newVL$, it has to be placed immediately after a new formation description (no other instructions can be between them). When this instruction is reached, robots stop, new position of the virtual leader is computed (the default position – 50 cm in front of the first robot and in the middle of the formation in transversal direction) and then robots start moving again.

Robots do not share information about their formation with others. Unless the scenarios are identical on all robots, the trajectories of the different robots could cross each other, robots could stop or even crash if the gaps between them are too small (because of the delay of LIDAR data).

Comments can be inserted into the scenario. They start with "# " and can be placed on a new line not at the end of a line with instruction. It is possible to leave blank lines in the scenario. When two subformations are present, the orders for each of them have to be ordered. The instructions can be separated into two independent blocks or written all together in sequential order (first instruction for first subformation, first instruction for the second one, etc.), examples 6.1 and 6.2 have the same results even though the instructions have different order.

Listing 6.1: Instructions in sequential order

```
move 1 45 group1
move 1 -45 group2
move 3 -45 group1
move 3 45 group2
move 1 -45 group1
move 1 45 group2
move 2 45 group1
move 2 -45 group2
```

Listing 6.2: Instructions in two separate blocks

```
move 1 45 group1
move 3 -45 group1
move 1 -45 group1
move 2 45 group1

move 1 -45 group2
move 3 45 group2
move 1 45 group2
move 2 -45 group2
```

An example of a correct scenario is:

Listing 6.3: An example of a correct scenario

```
robots R1 R2 R3 R4
groups
group1 R1 R3
group2 R2 R4

start
square R1,R2;R3,R4 1.2 1;1
move 2 0
move 4 -90

# split the formation in half longitudinally
column R1;R3 1.2 0
column R2;R4 1.2 0
newVL

# move the two columns apart and then back together
move 1 45 group1
move 1 -45 group2
move 3 -45 group1
move 3 45 group2
move 1 -45 group1
move 1 45 group2
move 2 45 group1
move 2 -45 group2

# join the two subformations
square R1,R2;R3,R4 1.2 1;1
newVL

circle -3 270 0.1
```

The formation consists of four robots with roles R1, R2, R3 and R4. Two robot groups will be used in the scenario – group1 and group2. At the beginning, robots are in a square formation, the distance between the rows is 1.2 meters and the distance between the robots in one row is 1 meter. The formation will move two meters forward, then turn 90 degrees to the right and go forward another 4 meters. Afterwards, the formation will split in half longitudinally, two new formations are columns of two robots with distance of 1 meter. The two formations are then going to move away from each other and then back together. They will join back together into one square formation. The last instruction is a circular arc with radius of 3 meters. The robots will try to move with angular speed of 0.1 rad/s and the central angle is 270 degrees. Robots are going to rotate to the right, because the value of centre is negative.

A Python script `check_scenario` is part of the package. The script checks, if all the instructions are correctly written and if there is a possible collision of robots' trajectories. This script also provides a demo - if the parameter provided on startup is demo, a commented scenario is printed to the command line. This scenario explains the usage of all the possible instructions and the meaning of the different values.

Chapter 7

Testing and results

7.1 Simulation

It is not practical to use the existing 3D simulator just for new scenario verification, because it is very computationally intensive and even a short scenario could take more than 30 minutes to simulate. That's why I created a simple 2D simulator on the basis of `turtlesim` from ROS tutorials [16]. The simulation is purely geometrical, no real properties of the robots are taken into account. Simulator also creates a simplified LIDAR for each of the robots. The simulated world is a square room with a side of 10.673 meters (the size is set as a parameter `/room_size` by the modified `turtlesim` node). This room is on LIDAR represented only by its walls, the floor and the ceiling are not displayed. It is possible to insert an obstacle into the world. The obstacle has to consist of finite amount of line segments. Robots and obstacles are 1 meter high, the walls of the room are 2 meters high. No parts of the robot body are visible to the LIDAR of the same robot. The number of rays is decreased from 1024 on real LIDAR to 256 in simulation. The obstacles are detected only in 2 dimensions, the third dimension is added artificially – the points are copied with a space of 10 cm up to the desired height. The 2D simulator does not use simulated time (the simulation is real-time), too high number of robots could cause decreased performance (if the number of robots is higher than 4, a warning is displayed. One of the consequences of the high number of robots is decreased frequency of LIDAR images).

2D simulator uses three types of nodes. The first one is the `turtlesim turtle_node`. It is an altered version with higher frequency of the main loop and with a MARV robot image instead of turtles (the package is renamed to `sim2d` to avoid any confusion with the original package). The second node `turtle_bridge` provides the same interface as a real robot. New topic `cmd_vel` is created, because the `turtlesim` one works in a different way. Real robot has two `cmd_vel` topics with different priority. 2D simulator is subscribed to both, but they are equal. The TF tree is simplified, only the frames `map`, `map_fast` and `base_link` are present. Each robot has its own TF tree, the trees are unconnected and its up to the user to create a world frame. The z axis of frames `map` and `map_fast` is inverted in comparison to `base_link`, this is consistent with the 3D simulator. The last node type is

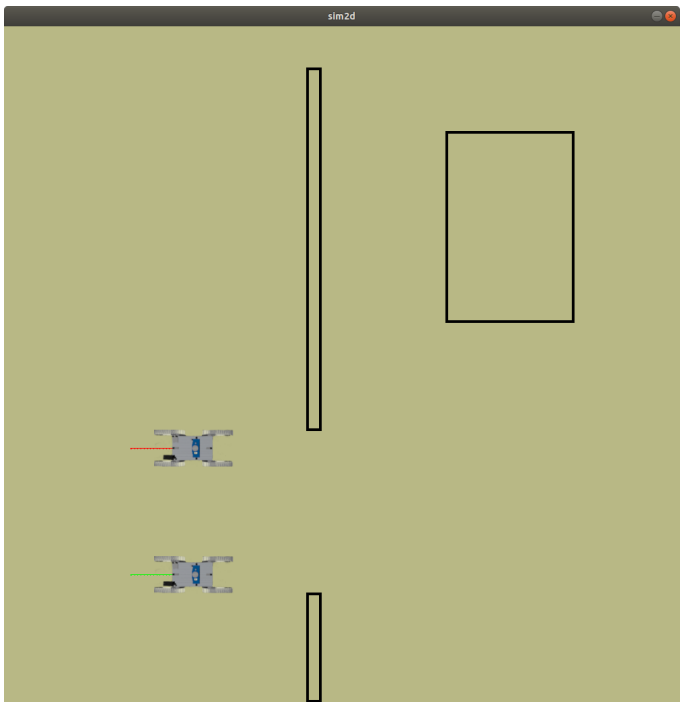

Figure 7.1: The modified `turtlesim`. Trajectory of each robot is shown with different color, obstacles are drawn with black line.

`lidar_sim`. This node is responsible for point cloud creation. Each robot has its own node that is launched by the `turtle_bridge` node. The obstacles, robots and walls can be described as finite number of line segments. The ray is calculated and then intersections with all lines (not the segments but whole lines) are found. If the intersection lies inside the line segment the point is selected as valid. Only the closest valid point to the robot's base link is then selected, the only exception are the upper parts of the wall. The lower half of the wall is added to the point cloud only if no valid intersection was found for the ray, but the upper half of the wall is always visible.

`turtle_bridge` accepts one parameter – file with world description. It is a text file with two parts. The first one is the description of the robots and their position (x, y and rotation). Each robot has a name that is used as a prefix for all topics and frames, because all simulated robots are using one ROS master. Obstacles are described by a sequence of points. The points have to be in correct order on the circumference of the obstacle, but the direction does not matter (clockwise or counter clockwise). Obstacle is drawn in the `turtlesim` window by a robot named draw with black color.

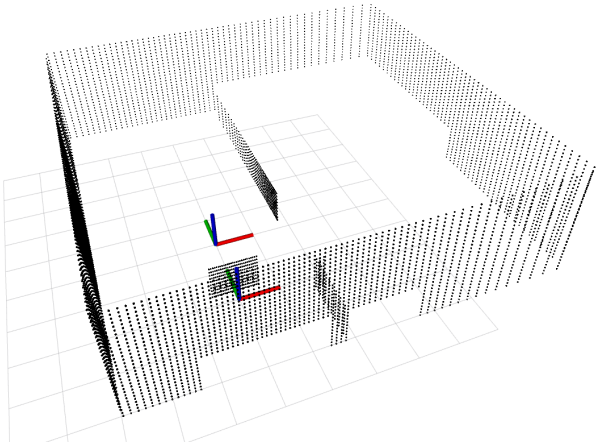

Figure 7.2: The simulated LIDAR from 2D simulator for the same world as in figure 7.1.

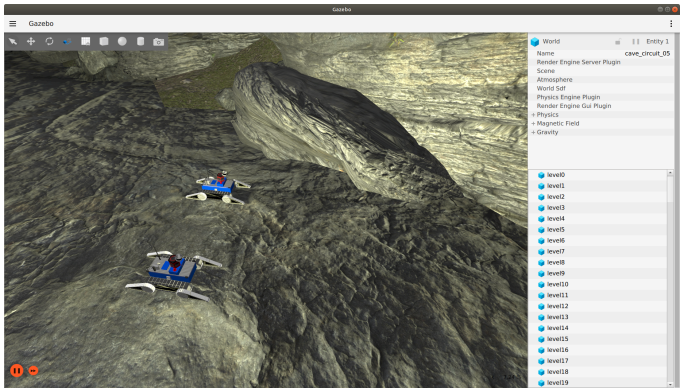

Figure 7.3: The graphical interface of the 3D simulator.

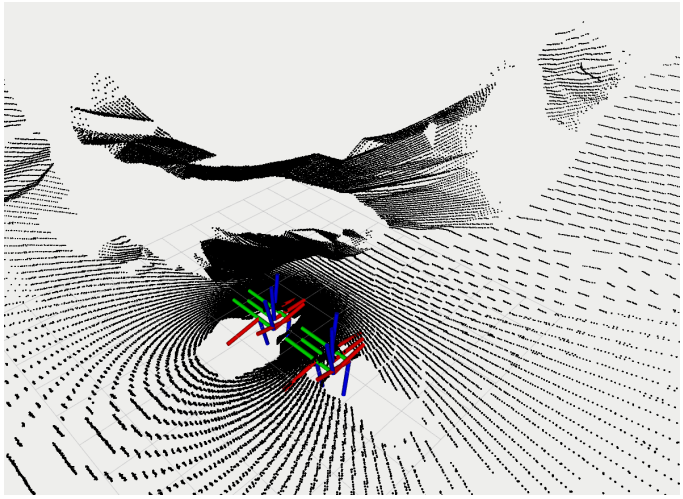

Figure 7.4: The simulated LIDAR from 3D simulator for the same world as in figure 7.3.

7.2 Results

7.2.1 Simulation

Two scenarios for two robots were used to test the designed algorithms. The first one is fairly simple, just two 90 degree turns, the first one to the right and the second one to the left.

Listing 7.1: The first testing scenario

```
robots R1 R2
row R1,R2 0 1.2
move 2 0
move 2 -90
move 4 90
```

The second scenario is much more complicated. It contains sharp and gentle corners, straight sections of different lengths and a long circular arc. The formation also splits, new virtual leaders are created and the robots continue to move separately.

Listing 7.2: The second testing scenario

```
robots R1 R2
groups
group1 R1
group2 R2

start
row R1,R2 0 1.2
move 2 0
move 2 -90
circle 2 270 0.1
move 1 0

row R1 0 0
row R2 0 0
newVL
move 1 30 group1
move 1 -30 group2
move 1 90 group1
move 1 -90 group2
move 2.5 90 group1
move 2.5 -90 group2
```

2D

The trajectory followed by the robots is not ideal, but that is caused by the virtual leader. Its angular speed is limited and therefore, robots don't go

completely straight after the turn, because the virtual leader is not directly on the desired trajectory, but slightly off. The distance between the robots slightly changes while turning, the outer robot lags behind at the beginning of the turn, but manages to close the gap. On the other hand, the inner robot has a small loss on the outer robot at the corner exit.

Robots are not at their ideal position at the corner exit and are therefore slightly turning even after the corner. These small corrections lead the robots to their correct position, but are definitely noticeable. The incorrect position at the corner exit is caused by imperfect following of the trajectory. If the robots were to follow the trajectory perfectly, the gap to the virtual leader and the distance between the robots would be bigger. It is impossible to do a perfect turn with skid steer robots. Omnidirectional robots are required for rotation of the whole formation. The robots will always slightly deviate from their desired position in the corners. This issue gets more prominent in sharp corners. However, if the straight section behind the corner is long enough, any errors in robot position will be eliminated.

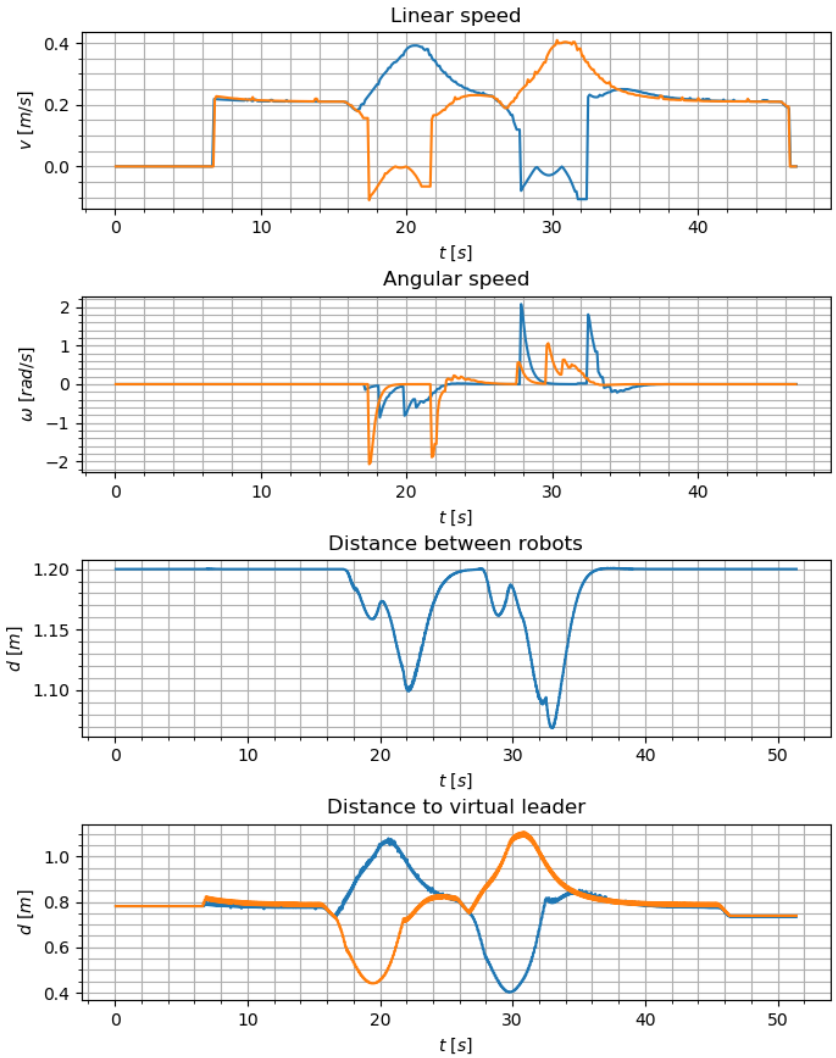

Figure 7.5: Speeds required by the controller and distances for the first test scenario. The inner robot almost stops while turning, because the virtual leader is too close. The distance between the robots decreases while turning, because the robots are not following the leaders trajectory perfectly (they are allowed to do it, this leads to better rigidity of the formation). Even though the distance to virtual leader deviates from the desired one while turning, the robots are able to return to the desired distance shortly after the corner.

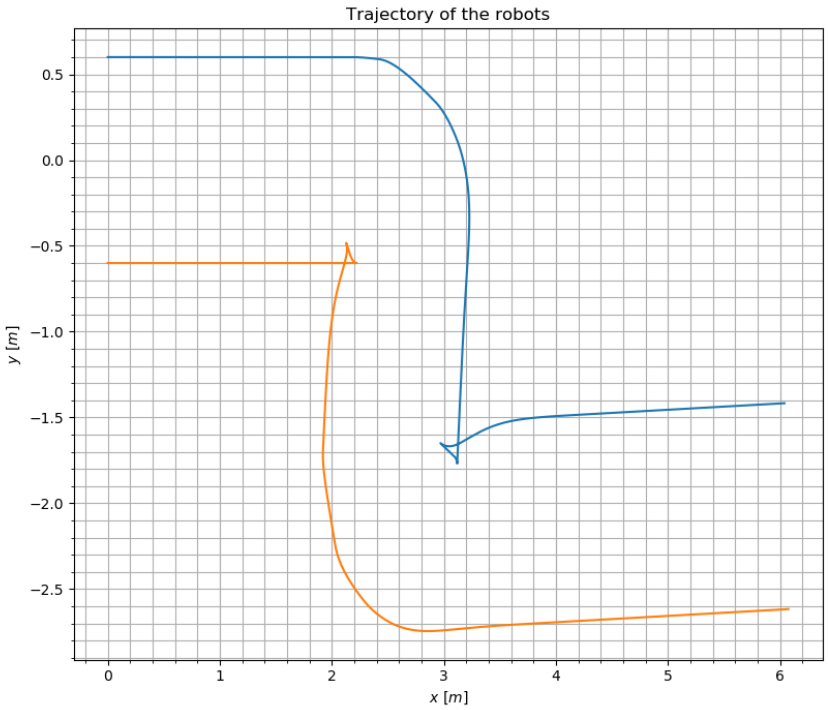

Figure 7.6: The trajectory of the robots for the first test scenario is not perfect, the virtual leader is turning with finite velocity and thus is not on the desired trajectory. The robots follow the leaders trajectory and that's why the part after the turn is not straight but slightly curved - the virtual leader is slowly returning to the desired trajectory. The inner robot is closer to the other robot at the corner exit than it should, therefore, it is turning away from the formation at the corner exit.

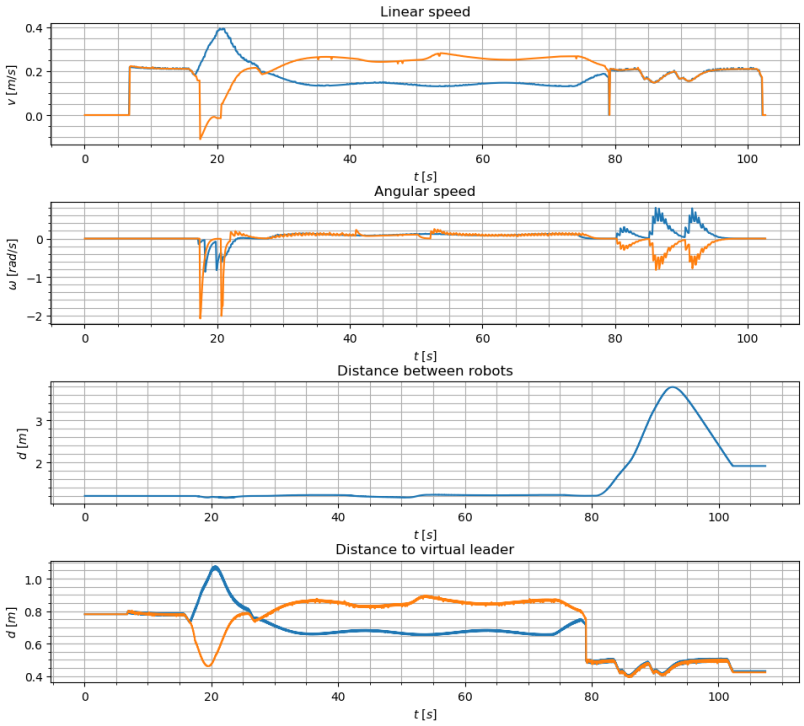

Figure 7.7: Speeds required by the controller and distances for the second test scenario. The inner robot slows down and the outer speeds up in the arc, but the speed difference is not big enough. The outer robot lags behind the inner robot (from time of 52 seconds to 75 seconds), but the difference is then lowered in the straight section of the trajectory. Formation splits at the time of 78 seconds (sudden change of distance to the virtual leader). Both robots follow their virtual leader well after the formation splits. The distance decreases while turning, but returns to the expected value after the turn.

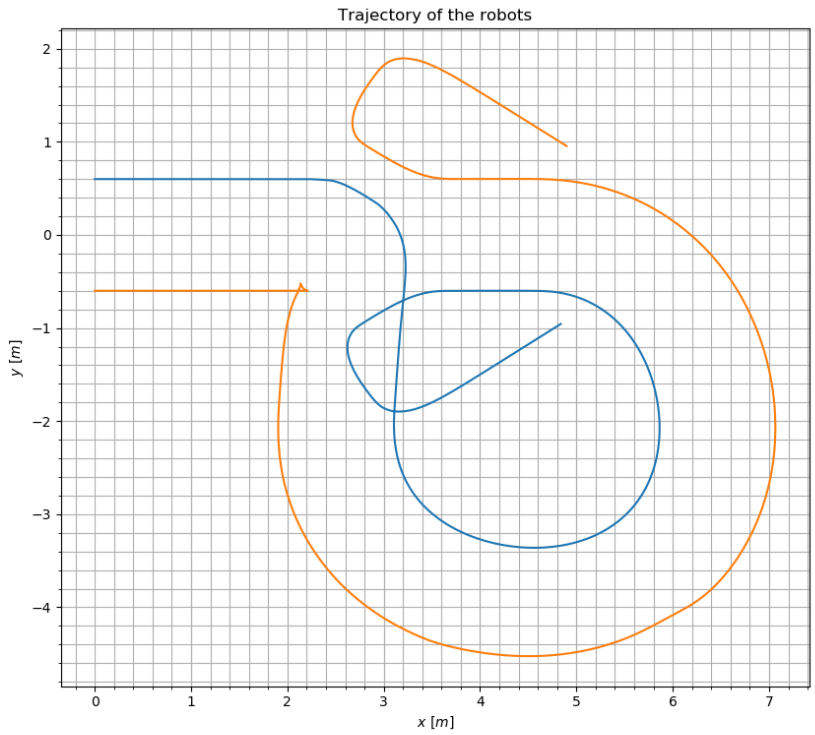

Figure 7.8: Trajectory of the robots for the second scenario. The result is as expected with the same shortcomings as in the first test.

3D

The 3D simulation reflects real properties of the robot, therefore, the results are worse than in the ideal world of 2D simulation. Robots are still capable of properly following the virtual leader, but the distance to the virtual is less stable. On the other hand, the maximal change in the distance between the robots is almost the same (around 16 cm for 2D and 18 cm for 3D), the formation looks as stable as in the 2D case. In the second test, the system performed better than in 2D simulation. The distance between the robots was more stable, which resulted in a better looking formation movement. This is due to the imperfect movement of the robots. Robots in 2D respond to control inputs immediately, but the real robots have some inertia and therefore are not capable to change the direction instantly. The speed of the robots is smoothed out which could lead to better formation movement.

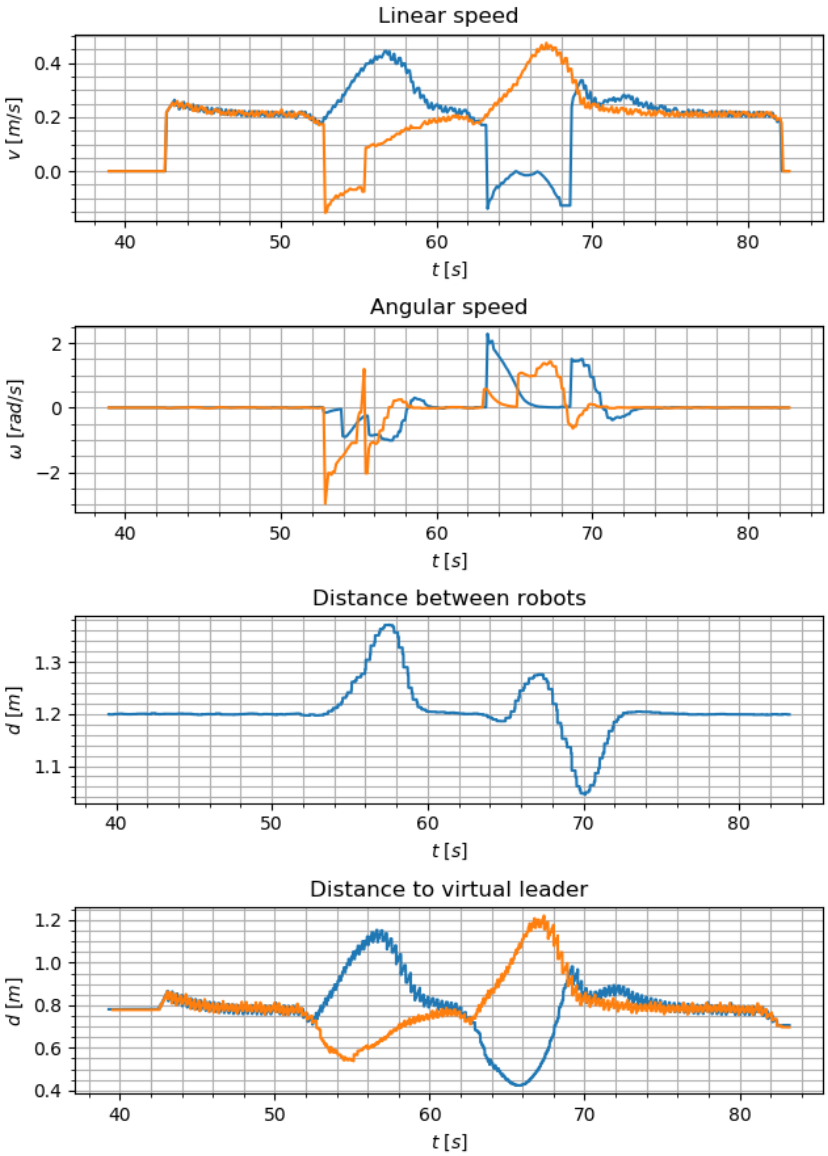

Figure 7.9: Speeds required by the controller and distances for the first test scenario in 3D simulation. The inner robot is capable of better following the virtual leader than in the 2D simulation. Because of that the distance between the robots is increasing this time. Robots again return to the desired position shortly after the turn.

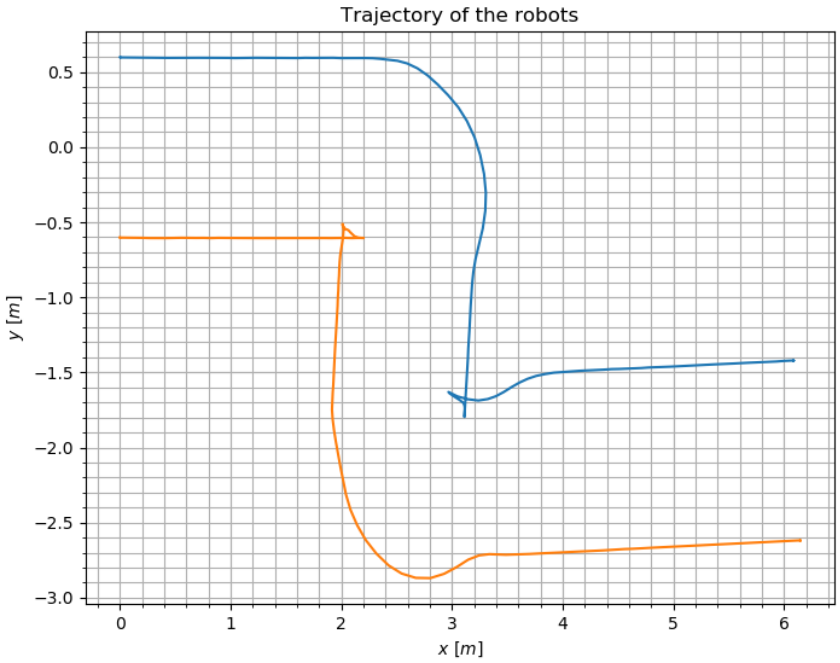

Figure 7.10: The trajectory is more deformed than in the 2D simulation but still acceptable. The inner robot again has to turn away from the formation at the corner exit, because it got too far while turning.

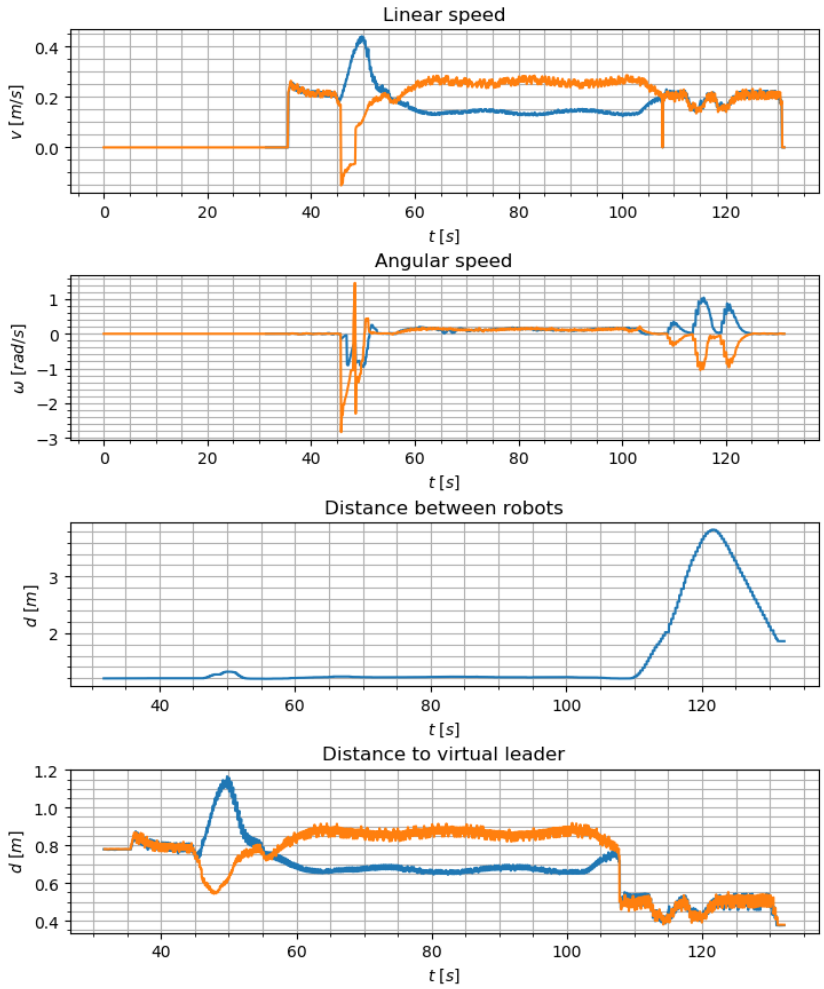

Figure 7.11: Speeds required by the controller and distances for the second test scenario. The distance between the robots in the circular arc is closer to the desired one than in the 2D simulation. As a result, the outer robot lags behind the inner robot less.

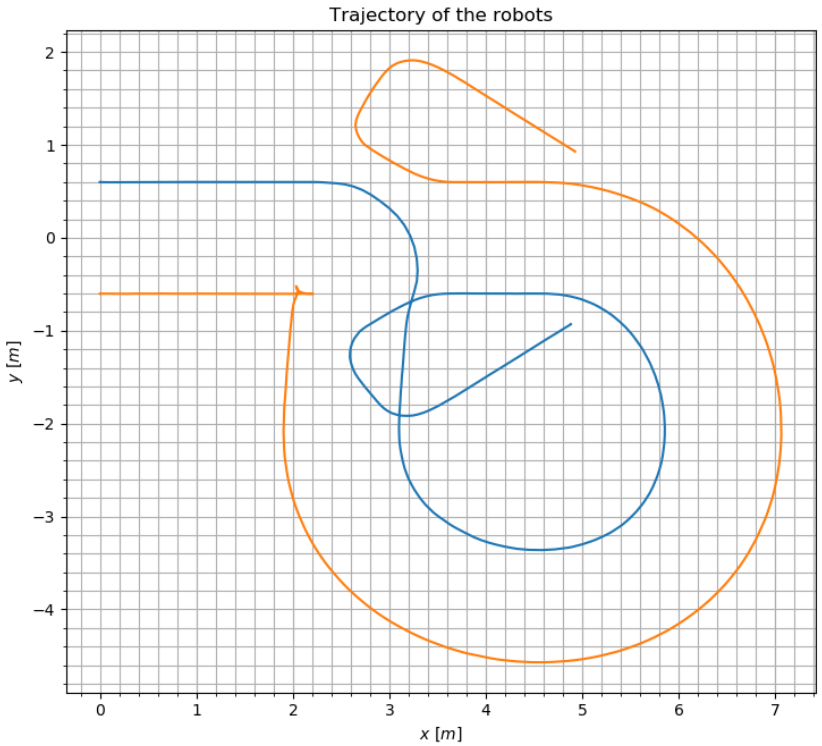

Figure 7.12: The resulting trajectory looks more or less the same as the one from the 2D simulation. Trajectories of both robots are smooth and match the scenario quite well.

7.2.2 Real robots

Three scenarios were used for the testing. The first one has only a single turn (scenario 7.3, the second one includes formation splitting and the third one was also used for simulation (7.2).

Listing 7.3: The first testing scenario

```
robots R1 R2

row R1,R2 0 1.2
move 1 0
move 2 -30
```

Listing 7.4: The second testing scenario

```
robots R1 R2

groups
group1 R1
group2 R2

start
row R1,R2 0 0.8
move 1 0

row R1 0 0
row R2 0 0
newVl

move 1.5 -30 group1
move 1 -40 group2
move 1 -90 group2
```

Testing on real robots was done with an older version of the programs. The controllers were set slightly worse, which resulted in bigger gap while turning. The inner robot was sometimes turning too fast in sharp corners, which lead to the robot turning around (the robot exited the corner in reverse and continued driving this way). The robots were again capable to close the gap after the corner. If one of the robots was blocked by an obstacle, it correctly stopped in safe distance and after the obstacle moved from the robot's way, the robot continued to follow the given trajectory.

In circular arc the outer robot was lagging behind the inner one and the inner robot's movement was unstable – it did a lot of small direction changes to correctly follow the arc. It is caused by the proximity of the target. If the target is too close to the robot, any small offset to the side causes a big change in the angle and therefore a higher angular speed of the robot (robot has to turn faster to get the correct heading in comparison to the situation, when the target is far away). These issues were mostly resolved by changes

in the controller tuning and the distance at which the target is considered as reached (the distance for the test with real robots was 20 cm and 30 cm for the results presented in section 7.2.1). The angular speed of the virtual leader was also limited after the test with real robots. This results in worse virtual leader trajectory (not so sharp corners), but increases the ability of the robots to correctly follow the leader's trajectory (the necessary direction changes are less abrupt and even though the robots are capable of high angular speeds, it is impossible to control such a fast motion – the robot is unable to stop as desired which leads to overshooting of the desired heading).

Further tests with the updated controllers are necessary to prove, that the proposed solution works not only in simulation, but also with real robots. The tests that took place with the old version of the controllers show, that all the basic parts of the system work – robots correctly synchronize and are able to keep the correct distance in a straight part of the trajectory. The only problem was the instability in the corners that is solved by the changes.

Scenario 7.3

The movement of the inner robot through the corner was not ideal, it first started to turn in opposite direction (to move slightly to the left side), but then it quickly reached the correct heading. The corner was not that sharp and thus the outer robot managed to keep up with the inner robot. Robots exited the corner side by side.

This scenario was also used to test robots ability to stop in front of obstacles. One robot (HUSKY) had the way blocked. Robot properly stopped and if it was equipped with a speaker, it would make the sounds described in section 5.2. After the obstacle moved away, the robot continued to follow the trajectory. In order to catch up to the other robot, it had to significantly speed up. As a result the robot went too far into the corner and had to stop because of the other robot. As soon as the outer robot exited the corner, HUSKY turned and again closed the gap to the other robot.

Scenario 7.4

This scenario was executed without flaws. The robots drive in one formation just forward without turning, therefore the movement was perfectly synchronized. After the formation splits, the robots continued to follow the virtual leader's trajectory without errors, because the leader had no offset to the side (virtual leader lied at the x axis of robot's `base_link` frame). Therefore, the robot had to move only as fast as the virtual leader, movement with this velocity could still be controlled.

Scenario 7.2

One of the tests was unintentionally carried out with different versions of the scenario for each robot, the trajectories of the robots were crossing each other. The robots were driving side by side.The robot on the left started a 90 degree

turn to the right, while the other robot was driving forward. The turning robot bumped into the other robot, because of the LIDAR data lag. At the time of the crash, no data suggesting that the way is blocked, was available and therefore the robot continued to move. This shows a big limitation of the whole system, if an obstacle appears right in front of the robot, it might not be able to stop in time.

The inner robot turned around in the 90 degree corner. Robots were mostly synchronized in the circular arc, but the inner robot was oscillating.

Chapter 8

Conclusions

The main target of this work was to control formation of skid steer robots. The formation should follow a trajectory planned by the user. Obstacles could appear in the environment and robots have to react in a safe way. Virtual leader was used to control the movement of the whole formation. Safe operation is ensured by LIDAR - program receives the point cloud and searches for obstacles in them. Robots try to avoid obstacles with the usage of virtual force field. If the obstacle is too close to the robot, robot stops to prevent collision.

The results from simulations and from the tests on real robots show, that the proposed solution works as intended. The beginning of the movement is correctly synchronized and the robots maintain their desired position throughout the majority of the choreography. The formation loses shape in corners, robots in one row preserve the prescribed distance, but the rows get closer or further away from each other as the formation is turning.

There are some limitations of the possible followed trajectory (based on the limited capabilities of the robots), the formation can't turn on spot and the formation is not completely rigid while turning. All tests show that the robot on the outside of a sharp turn slightly lags behind, but this loss is eliminated after the corner. There are also some deficiencies in the obstacle detection. If the robot or the obstacle is moving too fast, a collision may happen. The data from LIDAR lags behind reality and in some cases the obstacle might be undetected until the collision or detected on a place that is not obstructing the robot (even though the obstacle is right in front of the robot).

The usage of the program is not limited to the robots used for testing. The proposed solution could be modified (the structure of the TF tree and the commands for robot control) for different multi-robot system. Since the obstacle detection is done in a separate node, the robots do not have to be equipped with LIDAR for correct functioning of the main node.

Bibliography

[1] P. Paniagua-Contro, E. G. Hernandez-Martinez, O. González-Medina, J. González-Sierra, J. J. Flores-Godoy, E. D. Ferreira-Vazquez, and G. Fernandez-Anaya, "Extension of leader-follower behaviours for wheeled mobile robots in multirobot coordination," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[2] Y. Dai, V.-H. Tran, Z. Xu, and S.-G. Lee, "Leader-follower formation control of multi-robots by using a stable tracking control method," in *International Conference in Swarm Intelligence*, pp. 291–298, Springer, 2010.

[3] T. D. Barfoot and C. M. Clark, "Motion planning for formations of mobile robots," *Robotics and Autonomous Systems*, vol. 46, no. 2, pp. 65–78, 2004.

[4] M. Saska, V. Vonásek, T. Krajník, and L. Přeučil, "Coordination and navigation of heterogeneous MAV–UGV formations localized by a 'hawk-eye'-like approach under a model predictive control scheme," *The International Journal of Robotics Research*, vol. 33, no. 10, pp. 1393–1412, 2014.

[5] M. Saska, V. Spurný, and V. Vonásek, "Predictive control and stabilization of nonholonomic formations with integrated spline-path planning," *Robotics and Autonomous Systems*, vol. 75, pp. 379–397, 2016.

[6] Q. Chen, Y. Sun, M. Zhao, and M. Liu, "A Virtual Structure Formation Guidance Strategy for Multi-Parafoil Systems," *IEEE Access*, vol. 7, pp. 123592–123603, 2019.

[7] M. A. Lewis and K.-H. Tan, "High precision formation control of mobile robots using virtual structures," *Autonomous robots*, vol. 4, no. 4, pp. 387–403, 1997.

[8] J. Alonso-Mora, E. Montijano, T. Nägeli, O. Hilliges, M. Schwager, and D. Rus, "Distributed multi-robot formation control in dynamic environments," *Autonomous Robots*, vol. 43, no. 5, pp. 1079–1100, 2019.

[9] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, pp. 396–404. New York, NY: Springer New York, 1990.

[10] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[11] O. Takahashi and R. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 2, pp. 143–150, 1989.

[12] A. Sgorbissa and R. Zaccaria, "Planning and obstacle avoidance in mobile robotics," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 628–638, 2012.

[13] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, D. Heřt, M. Petrlík, T. Báča, V. Spurný, F. Pomerleau, V. Kubelka, J. Faigl, K. Zimmermann, M. Saska, T. Svoboda, and T. Krajník, "DARPA Subterranean Challenge: Multi-robotic Exploration of Underground Environments," in *Modelling and Simulation for Autonomous Systems* (J. Mazal, A. Fagiolini, and P. Vasik, eds.), (Cham), pp. 274–290, Springer International Publishing, 2020.

[14] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, T. Azayev, D. Heřt, M. Petrlík, T. Báča, *et al.*, "System for multi-robotic exploration of underground environments CTU-CRAS-NORLAB in the DARPA Subterranean Challenge," *arXiv preprint arXiv:2110.05911*, 2021.

[15] ROS Wiki, "sound_play," 2016 [Online]. `http://wiki.ros.org/sound_play` (Accessed 18 May 2022).

[16] ROS Wiki, "turtlesim," 2020 [Online]. `http://wiki.ros.org/turtlesim` (Accessed 18 May 2022).

Appendix A

List of attachments

- videos_2d.zip - videos of experiments in 2D simulator
  - final_test1_2d.mkv - scenario 7.1
  - final_test2_2d.mkv - scenario 7.2
- videos_3d.zip - videos of experiments in 3D simulator
  - final_test1_3d.mp4 - scenario 7.1
  - final_test2_3d.mp4 - scenario 7.2
- videos_real_robots.zip - videos of experiments with real robots
  - scenario1.mp4 - scenario 7.3
  - obstacle_test.mp4 - scenario 7.3 with an obstacle
  - scenario2.mp4 - scenario 7.4
  - different_scenarios.mp4
  - scenario3.mp4 - scenario 7.2 without the formation splitting
- code.zip - two ROS packages - `sim2d` and `dancing_robots`