university of groningen / faculty of science and engineering

Master Thesis

Distributed Formation Control with Dynamic Obstacle Avoidance of Omni-wheeled Mobile Robots via Model Predictive Control

Author:
P.J. (Pytrik) van Rosendal

Supervisor:
prof. dr. B. (Bayu) Jayawardhana
Second Assessor:
M. (Mauricio) Muñoz Arias, PhD

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Faculty of Science and Engineering

April 18, 2020

Declaration of Authorship

I, P.J. (Pytrik) van Rosendal, declare that this thesis titled, "Distributed Formation Control with Dynamic Obstacle Avoidance of Omni-wheeled Mobile Robots via Model Predictive Control" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*Begin at the beginning," the King said, gravely, "and go on till you come to the end; then stop.*"

Lewis Carroll

UNIVERSITY OF GRONINGEN

Abstract

Faculty of Science and Engineering

Master of Science

Distributed Formation Control with Dynamic Obstacle Avoidance of Omni-wheeled Mobile Robots via Model Predictive Control

by P.J. (Pytrik) van Rosendal

This master thesis was conducted in the Nexus group in the Discrete Technology Production Automation (DTPA) Lab at the University of Groningen. In this thesis, a distributed control law for a group of robots is presented, that solves the problem of formation control and path planning . In particular, we consider a control law that is a linear combination between distributed formation, and distributed path planning. We show through numerical simulations the effectiveness of the model predictive approach in dealing with dynamic environments as typically encountered in reality. The control system is able to move from start to goal pose in a number of challenging dynamic environments, such as worlds with high obstacle density, and worlds with narrow corridors and doorways. The implementation of the system was done in ROS, experiments were performed in the 3D physics simulator, Gazebo. This thesis concludes with a list of future work possibilities, as well as advice on which parts of the implemented system can be improved. Further research includes scaling of the system and improving robustness by merging different types of sensor data, such that the robots can sense their environment more clearly.

Acknowledgements

I thank my supervisor Bayu Jayawardhana as well as lab coordinators of the DTPA laboratory Simon Busman and Martin Stokroos for providing me with insightful discussions, help and advice. In addition to that I want to thank my family, colleagues and friends.

Contents

List of Figures

List of Tables

List of Symbols

Variable	Type(size)	Description
d	Scalar	Dimension of DS
N	Integer	Number of agents
$\dot{\xi}$	Vector (d)	Output Variable, e.g. velocity
π	Real number	Pi, defined as 3.1415926
i	Integer	Agent identification
p_i	Vector (2d)	Position of agent i
$\dot{p_i}$	Vector (2d)	Controlled velocity of agent i
m	Integer	Number of circles that encapsulate an obstacle
k	Integer	Identification of circle $\mathcal{O}_k$
p_k^{obs}	Vector (2d)	Centroid of each circle $\mathcal{O}_k$
R_k^{obs}	Scalar	Radius of each circle $\mathcal{O}_k$
$\partial\mathcal{O}_k$	Real number	Boundary of the k-th obstacle
R_i^{safe}	Scalar	Safe distance to the boundary of any obstacle
$\mathcal{P}_0$	Set	Set of initial conditions
$\mathcal{G}$	Matrix (N^2)	Graph
$\mathcal{V}$	Vector(N)	Set of vertices
$\mathcal{E}$	Vector(N)	Set of edges
$\mathcal{N}$	Vector($N-1$)	Set of neightbouring agents
L	Matrix (N^2)	Laplacian matrix
D	Matrix (N^2)	Degree matrix
A	Matrix (N^2)	Adjacency matrix
p^*	Vector ($2N$)	Desired relative position between the agents
c_f	Scalar	Formation gain
c_P	Scalar	Proportional gain
c_I	Scalar	Integral gain
c_D	Scalar	Derivative gain
w_{ij}	Scalar	Edge weight
c_α	Scalar	Obstacle avoidance gain
c_ζ	Scalar	Diffusion gain
c_κ	Scalar	Heading cost gain
c_β	Scalar	Distance to goal cost gain
c_λ	Scalar	Heading cost gain

Chapter 1

Introduction

Obstacle avoidance is a classical problem in robotics and many approaches have been proposed (**Choset2005PrinciplesLK**). These methods are usually limited to semi static environments (LaValle and Kuffner Jr, 2000). Online replanning or elastic-band methods deform the path locally and can therefore be applied for dynamic environments. However, they lose global convergence and have to be combined together with global path planning algorithms to create hybrid algorithms. Hybrid algorithms can switch between global path planning and local deformation (Vannoy and Xiao, 2008). In order to alleviate computational cost, recent work use customized circuitry on chips for faster global sampling and evaluation of all feasible paths (Murray et al., 2016).

Formation obstacle avoidance is important when for instance, a group of robots are carrying an object together and they need to cover a certain distance in an area crowded with obstacles. A formation which optimizes the use of sensors could be used to transport a larger object on top of the formation.

Formation, group motion control, and obstacle avoidance are mostly treated as separate issues in literature. When separate controllers are used jointly, conflict occurs. For example, the formation is distorted when an agent encounters an obstacle. This kind of behaviour, corresponding to animal flocking, may be useful when the formation requirement is quite loose, but the robots need to stay at a minimum safe distance.

In order to achieve group formation, motion control, and obstacle avoidance, a common coordinate frame is required e.g. a GPS or a gods-view camera above the formation. The issue of common approaches is tackled by researching methods to achieve simultaneous group formation, motion control and obstacle avoidance. By studying and implementing these methods we show that these issues are addressed appropriately. Moreover, the practical implementation of the presented algorithms in this thesis is computationally inexpensive, making the proposed algorithms attractive for autonomous robots with relatively affordable micro-controllers.

1.1 System Description

The full system is build up out of several elements. The simplified system architecture is illustrated in Figure 1.1. The control parts of the system are coloured blue for clarity. In addition to that, the state estimate feedback flow is represented as a dashed line for readability. Each element of the schematic is briefly explained below.

1.1.1 Perception and Localization

The objective of the perception and localization task is to provide the planning and formation control modules with a representation of the robot's environments and an accurate estimate of the robot's locations in the world. A predefined map, and onboard

sensors on the robots are used to construct a certain map, called an occupancy grid map, that gives a representation of the driveable and nondriveable areas. Dynamic obstacles are also detected and placed on the map. To determine an accurate position and orientation estimate of the robots, we use a standard 2D localization technique by Fox et al. (1999). The occupancy grid and the robot's positions and orientations provide the world representation in which motion planning and formation control are performed.

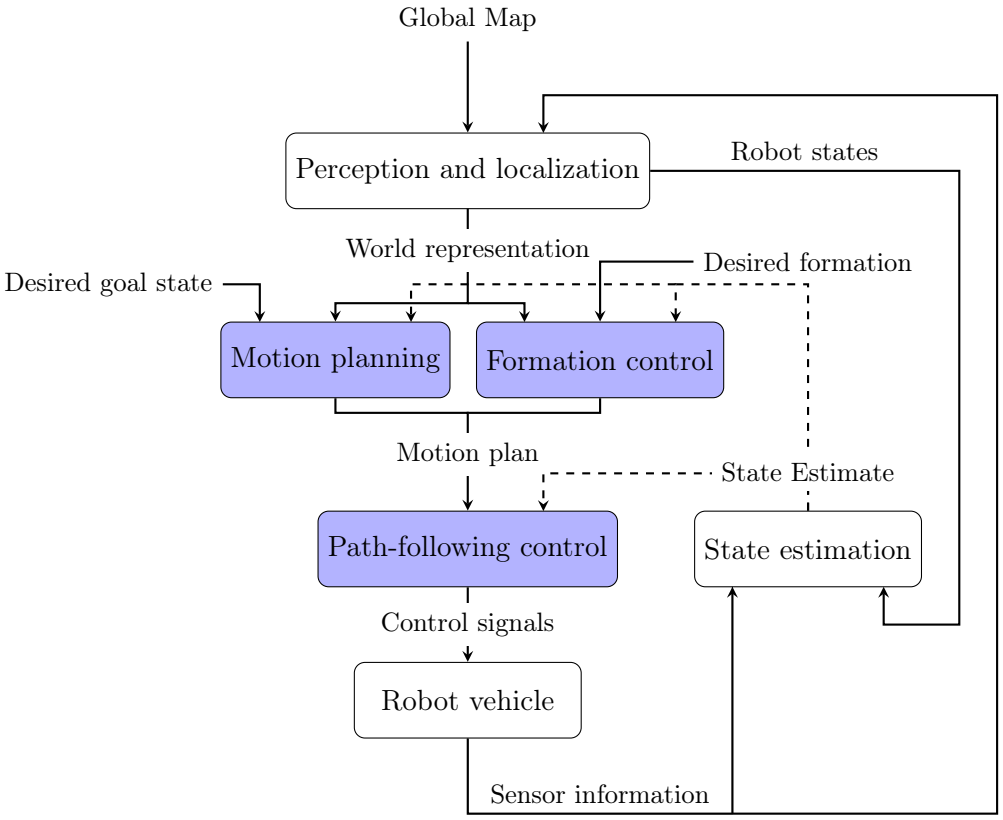


FIGURE 1.1: A schematic illustration of the proposed system architecture

1.1.2 State Estimation

To control the formation, accurate state estimations of the robots have to be obtained. We do this by cross-referencing laser scan data against IMU sensor data, and odometry information from the wheel encoders. The distinct features of the global map act as markers for the laser scanner in order to determine position and orientation of the robot. (Ljungqvist et al., 2019)

1.1.3 Motion Planning and Formation Control

Motion planning for omni-wheeled vehicles is relatively simple, since the vehicle kinematics are not complex and the state space is relatively small. The factor making path planning difficult is the non-convexity of the environment, which can cause the robots to get stuck. Therefore a global path planner is most suited. For the sake of comparing two methods, we implement, separately, both a local obstacle avoidance planner as well as a global motion planner.

1.1.4 Path-Following Control

The path-following control takes the motion plan and converts it into control signals for the mobile robots. It is crucial that the path-following control follows the states of the motion plan, otherwise collision with surrounding obstacles can occur. In order to make sure that the vehicle follows the motion plan, we implement state feedback control. The proposed path-following control has been proven to stabilize the path-following error as well as the formation error.

1.2 Problem Statement

Based on the literature overview above, where we have identified that there is currently a lack of distributed control methods that are capable to simultaneously perform formation control, avoid obstacles dynamically, and reach the target position through the use of a single low-level control, we can state the research problem that will be investigated in this thesis as follows:

How to control a group of mobile robots based on local information such that they (1) reach the desired formation, (2) avoid obstacle collision, and (3) reach the target pose.

1.3 Research Goal

Based on the research background, we can formulate the following objective in this research:

"*Design a control system using a distributed avoidance-formation control algorithm and apply it to a Nexus Robot formation equipped with a Lidar sensor*"

The focus of this report is the implementation of simultaneous obstacle avoidance and formation control. Also, being part of the Nexus group, interest lies in applying the algorithm to the physical Nexus robots.

1.4 Research Objective

The objective of this project is to deliver a solution to the problem of not having an optimal method for the nexus robots to avoid any moving obstacle. The final step of the project would be applying the algorithm to a Nexus robot formation. By applying it to a real robot we can verify our simulated results and test if the methods are robust.

1.5 Research Questions

The main research question is:

"*How can we implement a control law comprised of several parts, each responsible for a different task, such that the closed-loop system converges to the desired formation, avoids dynamic obstacles, and the whole group converges to the target position?*"

Resulting from the main research question, five sub-questions are defined in order to answer the main question.

- What is needed to model the system?

- Which algorithms are preferred for the task of simultaneous formation control and obstacle avoidance?
- How can a control system be designed to ensure stability?
- How well does the system perform under ideal conditions (e.g. numerical simulation)?
- How well does the system perform under various conditions (e.g. physics simulation or real-life testing)?

1.6 Contribution of the Thesis

This research project was motivated by the paper of Chan et al. (2018). In this paper, we made a contribution to the research in the field of systems and control. The paper by Chan et al. (2018) displays the control laws that we implemented, and it solves the problem of obstacle avoidance and rigid formation control quite beautifully. We added the visualization and thereby concretised the control law. The control law is describing a process in 2D space, and the 3D visualization is very pleasing to see. We have found something satisfying in seeing what the control law looks like and that could only have come about from the function of getting something on the screen to describe the control law in the original paper by Nelson et al. The implementation in ROS and Gazebo is available as open-source software [1] and can be used for any robot formation using laserscan range data.

1.7 Thesis Outline

This thesis is composed of several chapters. Below is a list of the chapters with an outline of what they contribute to the thesis.

Chapter 2: Background Information

This chapter reviews the current state of the art on approaches that could aid in achieving the goal of this thesis. These approaches address the problem of path planning, obstacle avoidance, and formation control.

Chapter 3: Proposed Solution

The main scope of this thesis is to build a system that fulfills the requirements set in Chapter 1. We will validate our approach in the context of a challenging task. This investigation takes the form of sending the formation of nexus robots on a specific path towards a goal location, along which it has to avoid certain dynamic obstacles. The system should instantly modify the the its motions to avoid colliding with multiple obstacles.

Chapter 4: Simulation and Experiment Design

This chapter details the simulation setup. The first section describes the Nexus robots, sensors, hardware descriptions, type of simulations, and the simulation parameters.

Chapter 5: Results

This chapter contains the results of the simulations and experiments, many of which were performed several times. A contextual analysis is done. We also comment on the robustness of the system against environment variety and initial conditions.

Chapter 6: Discussion

[1] Available at https://github.com/pytrik

This chapter analyses the results of the experiments and explains what they imply. This chapter also discusses limitations of the current approach.

Chapter 7: Conclusion

This chapter restates the aims of this thesis and contains to which conclusions we have come.

Chapter8: Future Work

This chapter contains suggestions of things we would like to see in the future with respect to the current project.

Chapter 2

Background Information

This section contains background information on motion planning, obstacle avoidance, and formation control. The goal of robot motion planning is to find collision-free trajectories that allow the robot formation to reach the goal location as fast as possible. The problem of motion planning can be stated as follows: Given the start pose, desired goal pose, geometric description of the robot, geometric description of the world, find a path that moves the robot smoothly from point a to point b while avoiding collision with any obstacle. Even though the problem of motion planning is defined in the regular world, it lives in another space: the configuration space. A robot configuration x is a specification of the positions of all robot points relative to a fixed coordinate system. A robot configuration is usually expressed as a vector of positions and orientations. An example of a robot living in the 2D plane, representation of its state could be: $x = (x, y, \theta)$. In 3D space, x would be of the form (x,y,z,α,β,γ). A robot with multiple joints such as a robotic arm would have a state consisting of multiple sub-states: $x = (x_1, x_2, .., x_n)$ The configuration space is the space of all possible configurations. The topology of this space is usually not that of a Cartesian space. The C-space is described as a topological manifold. The C-space is obtained by sliding the robot along the edge of the obstacle regions while blowing them up by the robot radius. The larger the robot, the smaller the C-space is going to be.

Below, an example of the C-space for a robot, with different radii.

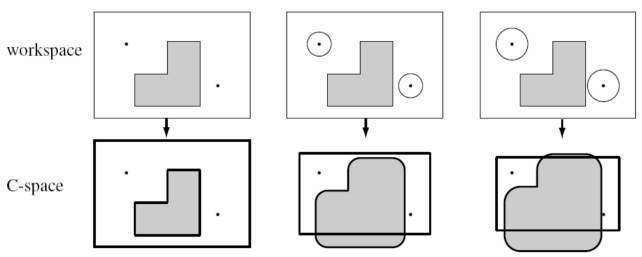


FIGURE 2.1: Example of the C-space for a circular robot. The C-space is obtained by sliding the robot along the edge of the obstacle regions.

The continuous environment needs to be discretized for path planning. There are two general approaches to discretize C-spaces. The fist approach is combinatorial planning, the second is sampling-based planning. Combinatorial planning methods produce graphs, or road maps, where each vertex is a configuration in C_{free} and each edge is a collision free path through C_{free} (Burgard et al., 2011).

2.1 Sampling-based Planning

In sampling-based planning we leave the idea of explicitly characterizing the free C-space. The way we explore the C-space is by basically shining a light around us in the form of a collision detection algorithm, that probes the C-space to find robot configurations which lie in the free C-space. We will take a look at dynamic window approach (DWA) and the rapidly exploring random trees approach (RRT).

2.1.1 Dynamic Window Approach

The dynamic window approach (DWA), also know as model predictive control (MPC) is an advanced control method used to control a process while satisfying a set of constraints and a scan of the robot's environment. What we want is to find a collision-free path, which brings the robot to the goal in a safe and fast manner. Written below is a description of how the algorithm works.

1. The algorithm takes a discrete sampling in the robot's control space.
2. For each sampled velocity, a prediction is made by forward simulation from the robot's current state to what would be the state if the sampled velocity were applied for a short amount of time.
3. Score every trajectory resulting from the forward simulations of each sampled velocity, using a cost function that contains characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard any trajectories that collide with obstacles.
4. Pick the trajectory resulting in the lowest cost, and send the associated velocity command to the mobile robot base.
5. Repeat until goal is reached.

We assume the robot takes motion command in the the form (v, ω). The question is, which sequence of (v, ω)'s bring us to the goal, are collision free and reachable under the current vehicle constraints? DWA has a certain velocity search space. Within this space is the actual velocity, and around it the obstacle free area V_a, the dynamic window, containing the speeds reachable in one time frame V_d, and all the possible speed of the robot V_s: $Space = V_s \cap V_a \cap V_d$ Problems of the DWA are overshoot when a robot drives through are corridor and has to enter a doorway, the robot does not slow down soon enough and has the make a question mark turn to enter the doorway successfully. Moreover, local minima might prevent the robot to reach the goal altogether., but this can be overcome using a local minima-free navigation function NF1 (Barraquand and Latombe, 1991). The upsides of the DWA are that it has low processing power requirements, and it is successfully used in many real-world scenarios.

2.1.2 Rapidly-exploring Random Tree

RRT is a sampling based algorithm used for path planning, where a tree of paths is built by sampling possible future states of the robots and connecting connecting the

new states to the tree. The pseudocode for the RRT algorithm is shown in Algorithm 1

Algorithm 1: GENERATE_RRT($x_{init}, K, \Delta t$)

```
T.init(x_init);
for k=1 to K do
    x_rand ← RANDOM_STATE();
    x_near ← NEAREST_NEIGHBOR(x_rand, T);
    u ← SELECT_INPUT(x_rand, x_near);
    x_new ← NEW_STATE(x_near, u, Δt);
    T.add_vertex(x_new);
    T.add_edge(x_near, x_new, u);
end
return T
```

The function names in the algorithm are fairly self-explanatory. The variable x denotes the state, u denotes the control sequence, and the tree is denoted by T. In normal language, the algorithm is basically:

- Create a random point in space.
- Find the nearest node in the current tree to the random point.
- Verify that when we take one step size in the direction of the random point, there will be no collisions.
- If no collisions and all constraints (e.g. maximum angular acceleration) are satisfied, attach the new point to the tree.
- repeat until distance between a node in the tree and the goal state is within the specified margin of error.

RRT* is also a sampling based technique first proposed by Karaman and Frazzoli (2011). Its anytime, incremental, asymptotically complete and optimal. The path can continue to be improved in real time. The longer it computes, the better it becomes.

To conclude, sampling-based planners are more efficient but offer weak guarantees. They are probabilistically complete. Moreover, they are widely used. However, complex C-spaces and problems with high-dimensionality are still computationally hard.

2.2 The Bug Algorithms

We start at one of the simplest path finding algorithms, with the simplest name. There exist many types of bug algorithms. The simplest is Bug0, which moves to the goal location until an object is detected. It then travels around the object either left or right, continuously storing the location closest to the goal. Once it has travelled all the way around, it will have a point stored which is the closest to the goal. It will then go around the obstacle again until it has reached the point, from which it the leaves in a straight line towards the goal. This algorithm is obviously extremely inefficient, which is why there exist improved bug type algorithms, such as the bug2 algorithm.

2.3 Vector Field Histogram

The vector field histogram algorithm, described first in Borenstein and Koren (1991) overcomes the issues of sensor noise and misreadings by creating a histogram of multiple previous sensor readings. The algorithm permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot toward the target. The VFH algorithm uses a two-dimensional Cartesian histogram grid as a world model. This model is continuously updated using range sensor data on-board the robotic vehicle. Figure 2.2 shows the robot observing two obstacles. The corresponding histogram is shown below. The x-axis represents the angles at which sensor readings can be performed. The y-axis represents the probability that there is an obstacle in a certain direction. The probabilities are calculated by creating a local occupancy grid map of the environment around the robot. The polar histogram is used to identify all gaps large enough for the robot to fit through. Using a minimal cost function, each passage is evaluated. The cost depends on the alignment of the robot current path with the goal.

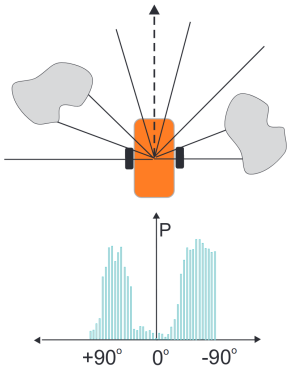


FIGURE 2.2: The polar histogram used in VHF algorithm.

The VFH method is based on the following principles:

- A two-dimensional Cartesian histogram grid is continuously updated in real-time with range data sampled by the on-board range sensors.
- The histogram is converted to a one-dimensional polar histogram, constructed around the current position of the robot. This will allow a spatial interpretation of the robot's instantaneous environment.
- Areas with probabilities below a set threshold value are designated as 'candidate valleys'. The candidate valley closest to the target direction is selected for further processing.
- The center of the selected candidate valley is determined and selected as the direction the robot should turn. The steering of the robot is then actuated so the robots heading corresponds to the desired direction.
- The speed is reduced when obstacles are encountered head-on.

The real time performance of the VFH can be gauged by looking at the sampling time T for the low-level controller, which is the rate at which steer and speed commands are issued. The following steps occur during one sampling time T:

1. Obtain lidar information from the sensor controller.

2. Obtain sonar information from the sensor controller.
3. Update the histogram grid.
4. Create the polar histogram.
5. Determine the free sector and steering direction.
6. calculate the speed command.
7. Communicate with the low-level motion controller (send speed and steer commands and receive position update).

2.4 Edge-Detection

Before the introduction of the VFH algorithm a popular obstacle avoidance method was based on edge-detection. A sensor is used to determine the position of the vertical edges of the obstacle and then steer the robot around either one of the detected edges. The line connecting two visible edges is considered to represent of the boundaries of the obstacle. This method was used in some previous work in our lab. A disadvantage of current implementations is that it has to stop moving in order to gather information and calculate the next course of action. This disadvantage is only limited by computation power, so the stopping is not an inherent limitation of the algorithm. A drawback for edge-detection approaches is the sensitivity to sensor accuracy. Shortcomings in this regard are (1) poor directionality, limiting the accuracy in determining the spatial position of an edge to 10-50cm, depending on the distance to the obstacle and the angle between the obstacle surface and the axis of the soundwave emitted by the ultrasonic sensor. (2) Frequent misreading, caused by ultrasonic noise from external sources or stray reflections from neighboring sensors. Misreadings can not always be filtered out and can cause false edges to be detected. (3) Specular reflections occur when the angle between the wavefront and the normal to a smooth surface is too large. When this happens the surface reflects the soundwaves away from the sensor and the obstacle is not detected or only partially detected and seen as smaller than it is in reality. Any of these three drawbacks can cause the algorithm to detect edges at wrong locations, resulting in wrong paths to be taken (Borenstein and Koren, 1991).

2.5 Potential Field

The potential field algorithm, as described in Khatib (1986) and Borenstein and Koren (1988), assumes that the robot is attracted by the goal and repulsed by obstacles. This means that obstacles and the goal exert forces onto the robot. The goal applies an attractive force and the obstacles apply a repulsive force. The sum of the forces subsequently determines the heading of the direction and speed to travel. It is a simple and elegant method, which is why it has a certain popularity. They can also be implemented quickly and provide good results without the need of many refinements. Research by Thorpe (source) introduces an approach that combines global and local path planning. Koren and Borenstein, of the University of Michigan have tested the potential field methods on real robots and have gained much insight in the strengths and weaknesses of this method. They have tested the algorithm before 1991, and back then computing power was limited, which gives reason to think that some problems are able to be overcome by our current controllers, however they claim the potential field method has inherent problems, and therefore not related to computing power.

The goal of their publication is to find possible remedies and to limit over-optimism with regard to the simplicity and elegance of the potential field method. Well known problems of this method are: trap situations due to local minima. If there is a U shaped obstacle between the robot and the goal, the robot is stuck in a dead-end. There are heuristic approaches to remedy this problem, however this solution is highly likely to result in sub-optimal paths. A better solution would be to abandon the heuristic recovery approach and pursue the method which uses an integrated global path planner. This way, the local path planner can monitor the path and when a trap situation is detected, the global path planner can be invoked to plan a new path based on all available information.

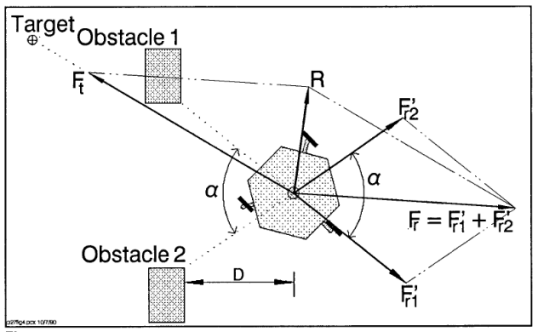


FIGURE 2.3: Under potential field based control the robot would not pass through densely spaced obstacles, the resultant force vector points away from the gap between the obstacles (Koren and Borenstein, 1991).

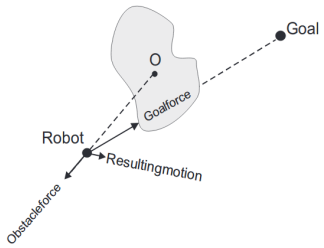


FIGURE 2.4: Illustration of the potential field algorithm (Susnea et al., 2009)

2.6 Gaps in Motion Planning Research

For many problems good solutions exist, however, the robot motion planning problem is not fully solved yet. There exist no good solutions to jointly plan the path under local constraints that overcome the decoupling of global and local planning.

2.7 Tools

Here we present an overview of the tools required to develop the control system.

2.7.1 Dynamical System

Dynamical systems theory is an area of mathematics that attempts to describe the behaviour of physical and artificial systems. A dynamical system is a system that

changes over time. Therefore it can be used to predict a systems behaviour, which allows correction of the system before a failure occurs. A continuous nonlinear dynamical system can be represented by a set of nonlinear differential equations that are expressed in terms of time $t \in \mathbb{R}^+$, a d-dimensional vector of state variables $\xi \in \mathbb{R}^d$, and an m-dimensional vector of input variables $u \in \mathbb{R}^m$:

2.7.2 PID Control

There are several types of control that deal with finding control laws for a dynamical system, including optimal control, adaptive and robust controls, inverse dynamics control and variants of Proportional-Integral-Derivative (PID) control. A PID controller has the form:

$$\tau(t) = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i \int_0^t e(\tau) d\tau \tag{2.1}$$

where $e(t)$ is the error between a reference trajectory and the actual trajectory, i.e. $e(t) = p^r(t) - p(t)$. The error is converted into a command by the controller. K_p, K_i, and K_d are gains to adjust the control behavior. The proportional gain scales the current error, such that as the error decreases, the output of the controller also decreases. The integral gain K_i accounts for removing the residual error by adding the cumulative value of the error to the control. When the error converges to zero, the integral term ceases to grow. For the derivative term it is the rate of change of the error that determines its magnitude. The faster the error changes, the larger it becomes.

PID Tuning

For small, low torque motors without any gearing, there are several approaches to PID tuning, however we will attempt a trial by error approach listed below based on Table 1 in Li et al. (2006).

1. Set all gains to zero.
2. Increase the proportional gain until the response to disturbance is a steady oscillation.
3. Increase the derivative gain until the oscillations go away in order to make the system critically damped [1].
4. repeat steps 2 and 3 until increasing the D gain does not stop the oscillations. item Set both gains to the last stable values.
5. Lastly, increase the integral gain until it provides a desired setpoint.

The disturbance we use is performance of a task, for example moving from startpoint to endpoint, or disturbing the formation by nudging a robot out of place. If the oscillation are growing larger and larger, the proportional gain needs to be reduced. Another note is that if the derivative gain is set too high, chatter can occur, where the system starts vibrating at higher frequency than the normal oscillations. In such case, reduce the derivative gain term until the chatter stops.

[1] Critically damped meaning that the damping coefficient c is equal to two times the mass m, multiplied with the natural frequency ω_n of the system, resulting in a damping ratio of 1.0. The system will not overshoot nor make oscillations when reaching the settling point.

2.7.3 Robot Navigation and Mapping

Localization is a well-known challenge in robotics. The sensors that can be used to localize our robot is the lidar and the wheel encoders, which relies on the robot not slipping. We can then use a map of our environment in combination with lidar data and data from the wheel encoders to estimate the position and orientation. The wheel encoders measure the number of rotations of each wheel and thereby change in pose [2] is determined. For this thesis we assume the map to be empty, and therefore lidar cannot aid in the determination of the pose. Another reason as to why the lidar cannot aid in mapping is because it is used for formation control, causing data to be missing or invalid in at least a quarter of the scan range of the lidar (for a square formation). For the localization we use a GPS sensor, for indoor GPS-like localization. In a real world application, we would use an ultrasound-based indoor GPS system by *MarvelMind*, which has proved to work with similar applications Rosolia et al. (2017).

2.7.4 Routh-Hurwitz Stability

The Routh Hurwitz stability criterion is a necessary and sufficient condition for the stability of a linear time invariant control system. The Routh test is an algorithm proposed by Eward John Routh in 1876 to determine whether all the roots of the characteristic polynomial of a linear system have negative real parts (Routh, 1877). The roots λ with negative real parts represent solutions of the system that are bounded $e^{\lambda t}$ and therefore stable, since as time approaches infinity, the exponent term will be zero. For second and third order polynomials it is relatively simple to determine . When roots of higher order systems are difficult to obtain a tabular method can be used to determine stability. This table is called a Routh array. We will not go into detail on the method of deriving such array, since our system will not be of higher order.

2.8 Software

Several pieces of software listed in the next subsections are needed for developing the control system.

2.8.1 Robotic Operating System

One of the requirements is a piece of software which can act as the middle-man between the operating system and the robots. Our choice is an open-source operating system, named ROS (Robotic Operating System). Its goal is to support code reuse in robotic research and development, which is also why we are going to use it. It provides easy testing and it is already implemented in Python and C++.

2.8.2 Gazebo

Robot simulation is essential for every robotics engineer. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately

[2] In computer vision and robotics, a typical task is to identify specific objects in an image and determine each object's position and orientation. The combination of position and orientation is referred to as the pose of an object Shapiro and Stockman (2001).

and efficiently simulate populations of robots in complex indoor and outdoor environments. All in all, Gazebo provides a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.

2.8.3 GMapping

The Gmapping package provides laser-based simultaneous localization and mapping (SLAM), as a ROS node called slam_gmapping (*Gmapping*). Using slam_gmapping, we can create a 2D occupancy grid map, like a building floorplan, from laser and pose data collected by a mobile robot. To make our map we use one mobile robot from the formation, with one laser range-finder. The robot provides odometry data and is the laser provides range data. Each incoming scan message is transformed into the odometry frame.

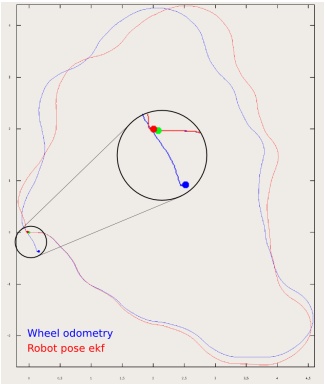


FIGURE 2.5: Map generated by the slam_gmapping topic.

2.8.4 Robot Localization

Robot localization is a state estimation tool in ROS. The tool is an implementation of a nonlinear state estimator for robots moving in three dimensional space. The localization node, called *ekf_localization_node* uses an extended Kalman filter (Wan and van der Merwe, 2006). In the extended Kalman filter, the state transition and observation models do not need to be linear functions of the state, but may instead be nonlinear The tool allows for fusion of an arbitrary number of sensors, such as odometry, an imu, gps, an even multiple sensors of the same type. Per sensor it can be decided which data should and should not be included in the state estimate. The tool allows to exclude that data on a per-sensor basis. Using multiple sensors causes a timing problem. Imagine the robot pose filter was last updated at time t_0. The tool will not update the robot pose filter until at least one piece of data is received from each sensor. Suppose a message is received from the imu at time t_1, and another message from the odometry sensors at time t_2, then the robot pose filter is updated for t_1 using the data from the imu at t_1, and a linear interpolation of the pose data from the odometry sensor of the odometry pose between t_0 and t_1.

To illustrate why it is useful to rely on multiple sensors data to determine an estimate of the robot pose, Figure 2.6 shows an experimental result where an RP2 robot started from an initial position (green dot), was driven around, and returned to the initial position. The odometry plot (blue line) shows the robot in a different position, while using the *robot_pose_ekf* tool, combining the odometry with an imu, shows a more accurate estimation of the end position.

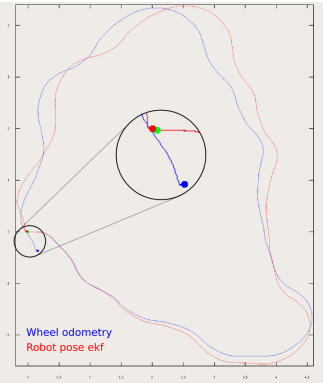


FIGURE 2.6: Trajectory of a PR2 robot plotted using both odometry, and odometry combined with imu pose data (**Robot_pose_ekf**).

In addition to the extended kalman filter, when we are estimating our pose using the dynamic window approach, we use a known map. To localize our robot on the map we utilize the AMCL package. It takes as input the map, filtered odometry from the ekf package, and lidar scan range information. It puts out a probabilistic point cloud of locations where the robot could be located.

2.8.5 DWA Local Planner

The *dwa_local_planner* package contains a controller that drives a movile base in its plane. The controller connects the path planner to the robot. The path planner creates a kinematic trajectory using a map. The robot then receives velocity commands based on the current position and the trajectory. This way it reaches its goal location. Along the trajectory, the planner creates a value function, represented as a grid map. The value function is represented as a grid map. Using the value function, the costs of traversing each grid cell in the local map is calculated. The controller's function is then to use this value function to determine dx, dy, $d\theta$ values to send to the mobile robot.
The idea of the Dynamic Window Approach (DWA) algorithm is the following:

1. The algorithm takes a discrete sampling in the robot's control space.

2. For each sampled velocity, a prediction is made by forward simulation from the robot's current state to what would be the state if the sampled velocity were applied for a short amount of time.

3. Score every trajectory resulting from the forward simulations of each sampled velocity, using a cost function that contains characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard any trajectories that collide with obstacles.

4. Pick the trajectory resulting in the lowest cost, and send the associated velocity command to the mobile robot base.

5. Repeat until goal is reached.

From the It might be interesting to name an alternative to the DWA local planner, called the Time Elastic Band (TEB) local planner. The overall idea of both planners is to plan motion of the robot along a given horizon, minimizing cost, while adhering to the kinematic constraints of the robot. The principle is well known in control theory

as model predictive control. Computing the optimal solution can be computationally expensive and therefore multiple approaches of finding it are discovered. Both approaches thus approximate the optimal solution in different ways and the optimization strategy differs. The TEB local planner primarily tries to seek a time optimal solution, but can also be configured for optimal path following (global reference path fidelity), where it strongly adheres to the reference path. The approach discretizes the trajectory along the prediction horizon in terms of time. This can result in a large number of degrees of freedom along the prediction horizon, depending on the step size of the discretization. On top of that the constraints of the optimization problem are removed in order to gain shorter computation times. This implies that constraints can not be guaranteed in any case, so obstacle collision, and breaking through velocity bounds can still occur. Since the approach relies on continuous optimization, the cost function must be smooth. It can not cope with grids and costmaps for function evaluation. Given enough computational resources, the planner achieves better controller performance than the DWA local planner, it resolves more scenarios and supports car-like robotic motions.

2.9 Hardware

When implementing the solution we will use off-the-shelf hardware components. They are listed below:

2.9.1 Computer Hardware and Software Description

For the simulations, an HP Z240 workstation equipped with an Intel Core i7-6700 CPU, containing four cores with eight threads clocked at 3.4GHz, and 16GB of memory. The GPU is an NVIDIA Quadro P2000, not required since the simulations are mostly CPU-bound. The workstation runs Ubuntu 16.04.6 LTS (Xenial) and ROS Kinetic Kame. The computer served as a master for simulated and real world experiments. For visualization of experiments we used the Gazebo simulator, version 7.0.0. This combination of Linux, ROS and Gazebo is recommended for replicating the experiments.

2.9.2 Nexus Robot

The nexus robots are equipped with mecanum wheels, which is used to make a special type of a holonomic drive. Holonomic drive means the robot can move in any direction without changing its orientation. This type of a drive is useful in close quarters. There are two different wheels which can enable holonomic drive, the omni-directional wheel and the mecanum wheel. The omni-wheel has its rollers mounted perpendicular to the wheel while mecanum wheels have their rollers mounted at a 45 degree angle to the plane of the normal of the wheel. The mecanum wheels make it necessary to have a different left and right wheel since the force vectors from the wheel to the ground have to cancel each other out such that equal input on all wheels causes the robot to go in a straight line.

2.9.3 Lidar Laser Scanner

The Nexus robots are equipped with an RPLidar A1, which is a low cost 360 degree laser range scanner. It does not need any coding to setup and range scan data can be obtained through the Serial port/USB communication interface.

In Figure 2.7 , the working mechanism of the RPlidar is represented. The system is based on laser triangulation ranging principle. The system measures distance data more than 2000 times per second. The lidar emits a modulated infrared laser signal, the object to be detected then reflects the laser and the returning signal is sampled by a vision acquisition system. The DSP embedded in the lidar processes the sample data and outputs a distance value and an angle value between the object and the lidar. The best results are obtained when detecting an object that appears white, since it reflects all wavelengths of light. Since in our application we are trying to detect other agents equipped with lidars, a white shroud has been 3D-printed and with the goal of a higher detection rate success.

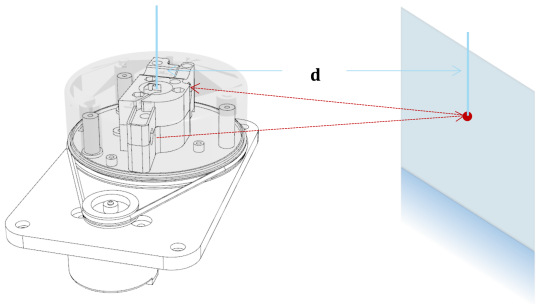


FIGURE 2.7: The RPLidar A1 working schematic

RPLidar can be used in below application areas:

- General robot navigation and localization
- Smart toy's obstacle avoidance
- Environment scanning and 3D re-modeling
- General simultaneous localization and mapping (SLAM)

In our application the lidar is going to provide data for formation control, group motion control and obstacle avoidance. The datasheet for the RPlidar A1 can be found in (Slamtec, 2016).

2.10 Requirements

Safety: In order to ensure safety, safety instructions and operating policy shall be provided as a README file. The person or persons operating the system shall be taught which measures to take in case of emergency. For the top level safety we refer to the safety rules provided by the DTPA lab support staff, dealing with the rules and safety regulations of the lab. Most importantly the safety of the batteries and how to handle with them. Operating environment:

- The system shall be deployed in an open area with at least 2 times the desired distance between agents as a safe radius around the system during the initialization phase.
- The system shall be able to detect obstacles. The system should provide warning if there is no possibility of converging to a goal location. Warnings shall be returned to the master computer.

- The system should be able to transport a large object from start to desired end location on top of the robot formation.
- The system should perform obstacle avoidance, detection, and formation control without the use of a global controller, or master.
- Each agent should broadcast feedback about their obstacle detection control. An agent's current state is constantly monitored by its neighbouring agents, therefore it is not needed to share the agent's own state data.

Chapter 3

Proposed Control Methods

In this chapter we go through the proposed control methods. The proposed control methods both consist of a linear combination of several algorithms. We propose two different solutions in order to compare them. One uses a reactive type algorithm for obstacle avoidance and the other solution uses a global planning algorithm combined with formation control. The chapter opens with the section containing the control algorithms. We want the solution to be as general as possible in order to be applicable or simple to adjust to as many problems as possible. First we go through the reactive approach, after which we explain the path planning approach using DWA.

3.1 The Control Algorithm

In this section the control algorithm based on the previously discussed control law is elaborated on. The algorithm is explained in normal English, together with requirements in order to make the algorithm work in simulation and real-world scenarios. The assumptions and also its limitations will be briefly summarized. The position and velocity of each nexus robot are described by the linear equation:

$$\dot{p_i} = u_i,\ i = 1, \ldots, N, \tag{3.1}$$

where $p_i \in \mathbb{R}^2$ and $u_i \in \mathbb{R}^2$ denote the position and controlled velocity of agent i respectively. We assume no uncontrolled forces cause changes in position. For the obstacle avoidance, we consider obstacles that can be encapsulated by $m \geq 0$ circles in the plane where the centroid and radii of each circle $\mathcal{O}_k,\ k = 1, ...m$, are given by $p_k^{obs} \in \mathbb{R}^2$ and $R_k^{obs} \in \mathbb{R}$, respectively. The boundary of each obstacle 'k' is defined by Each agent has an associate parameter R_i^{safe} which defines the safe distance to the boundary of any obstacle. We consider this parameter to be constant for all agents i.

One of the features of the control algorithm is the fact that its modular. Each part of the control law pertaining to a particular task can be removed or added without jeopardizing the successful completion of other tasks. Therefore our control law is a linear combination of control laws for different tasks.

$$u_i = u_i^f + u_i^g + u_i^o, \tag{3.2}$$

where u_i^f is the local control law for solving the formation control task of the i-th agent, u_i^g is the local control law for solving group stabilization task of the i-th agent, and u_i^o is the control law for the local collision avoidance task of the i-th agent. Each task will be detailed in the following subsections. The combination of the control laws will become our unified control framework. It has to be noticed that the framework is not restricted to these control laws, but for our application we will focus mainly on these laws to demonstrate our approach.

3.1.1 Formation Control Law

For completing the group formation task, we consider relative position-based formation control:

$$u_i^{\mathrm{f}} = c_{\mathrm{f}} \sum_{j \in \mathcal{N}_i} w_{ij} \left(p_j - p_i - (p_j^* - p_i^*) \right), \tag{3.3}$$

where $c_f > 0$ is the formation gain and $w_{ij} > 0, i, j = 1, \ldots, N$ the weight of the edge $(i, j) \in \mathcal{E}$. We can write the above control law in compact form as

$$U^{\mathrm{f}} = c_{\mathrm{f}}(L \otimes I_2)(p^* - p), \tag{3.4}$$

where U^{f} is the stacked vector of $u_i^{\mathrm{f}}, i = 1, \ldots, N$ and $\otimes$ is the Kronecker product.

3.1.2 Group Stabilization Control Law

In addition to the above distributed control task of formation control, we can add a group motion task, where we control the group's centroid to achieve certain control behaviour, such as following a certain reference trajectory. We are implementing two approaches, namely a purely reactive one, and a planning based approach. For completing the group stabilization task, we consider the existence of a central coordinator. The coordinator determines the position of the centroid of the formation based on the positions of the agents and uses it to compute the u_i^g for the group. Each agent has the same group stabilization input and therefore: $u_1^{\mathrm{g}} = u_2^{\mathrm{g}} = \ldots = u^N = u^{\mathrm{g}}$.

In order to demonstrate the framework we propose two separate control laws for obstacle avoidance and group stabilization. The first one is the following:

$$u^{\mathrm{g}} = -c_{\mathrm{P}} p_{cen} + c_{\mathrm{I}} \gamma + c_{\mathrm{D}} \rho, \;\; \dot{\gamma} = -p_{\mathrm{cen}}, \int_{\tau_1}^{\tau_2} \rho(\tau)\, d\tau = p_{\mathrm{cen}}, \tag{3.5}$$

where $c_{\mathrm{P}} > 0$, $c_{\mathrm{I}} > 0$, and $c_{\mathrm{D}} > 0$ are the proportional, integral, and derivative gains respectively. We write the compact form as

$$U^{\mathrm{g}} = (\mathbb{1}^{N \times 1} \otimes -c_{\mathrm{P}} p_{cen} + c_{\mathrm{I}} \gamma). \tag{3.6}$$

3.1.3 Obstacle Collision Avoidance

In order to safely move along a desired trajectory, the agents should avoid any obstacles during its journey. When an agent i is in close proximity to an obstacle it is assigned the task of obstacle avoidance. For such a scenario, the following control law is proposed in Tee et al. (2009):

$$u_i^o = c_\alpha \alpha_i^k(p_i) := \begin{cases} 0 & \text{if } \left\| p_i - p_{i,k}^* \right\| > R^{safe} \\ c_\alpha \frac{p_i - p_{i,k}^*}{\left\| p_i - p_{i,k}^* \right\|^2} & \text{if } \left\| p_i - p_{i,k}^* \right\| \leq R^{safe} \end{cases}, \tag{3.7}$$

where $c_\alpha > 0$ is the gain, and

$$p_{i,k}^* := \underset{x \in \partial \mathcal{O}_k}{\operatorname{argmin}} \; \operatorname{dist}(p_i, x). \tag{3.8}$$

The obstacle collision avoidance control law is a reactive algorithm, meaning the control is only calculated when required. When the relative distance between agent i and the obstacle k is less than the threshold R^{safe}, the agent i is going to activate the

collision avoidance action. The downside of this type of distributed obstacle avoidance is that it is only applied locally; when agent i activates the obstacle avoidance, the rest of the agent will still only be using the group stabilization and the formation control. Consequently, the unexpected behaviour by agent i causes the formation to deform. This is negated by introducing a dynamic obstacle controller with state variable ζ_i. This state variable is communicated to its neighbors. The local dynamic obstacle avoidance controller is described by

$$\begin{aligned} \dot{\zeta}_i &= c_\zeta \sum_{j \in \mathcal{N}} w_{ij}(\zeta_j - \zeta_i) + u_{\alpha_i} \\ u_i^o &= \zeta_i, \end{aligned} \tag{3.9}$$

where $c_\zeta > 0$ is the diffusion gain of the obstacle control law, $w_{ij} > 0, i, j = 1, \dots N$ are the weights of the edges $(i, j) \in \mathcal{E}$ and u_{α_i} is given by

$$u_{\alpha_i} = c_\alpha \alpha_i^k(p_i) \tag{3.10}$$

In compact form we write the distributed dynamic obstacle avoidance control law as

$$\begin{aligned} \dot{\zeta} &= -c_\zeta (L \otimes I_2)\zeta + U_\alpha \\ U^o &= \zeta, \end{aligned} \tag{3.11}$$

where U^o is the stacked vector of $u_i^o, i = 1 \dots N$, and U_α is the stacked vector of u_{α_i}.

3.2 Dynamic Window Approach Algorithm

The second control law is the Dynamic Window Approach (DWA), otherwise known as Model Predictive Control (MPC). The DWA has a cost function which minimizes certain characteristics and gives weightings to them. The cost functions that will be applied in order are:

- Oscillation costs: discards oscillating motions.
- Obstacle costs: discards trajectories colliding with obstacles.
- Goal front costs: prefers trajectories that make the robot move forwards to goal.
- Alignment costs: prefers trajectories that keep the nose of the robot on the path.
- Path costs: prefers that the robot stays close to the planned path.
- Goal costs: penalizes trajectories moving away from the goal.

In the dynamic window approach the commands sent to the robot are found in the velocity space. The dynamics of the robot is incorporated into the method by reducing the velocity search space to the velocities that can be attained taking into account the dynamical constraints. The admissible velocities are defined as

$$V_a = \left\{ (v, \omega) \mid v \leq \sqrt{2 \cdot \mathrm{dist}(v, \omega) \cdot \dot{v}_m} \wedge \omega \leq \sqrt{2 \cdot \mathrm{dist}(v, \omega) \cdot \dot{\omega}_m} \right\}, \tag{3.12}$$

where V_a is the set of velocities that allow the robot to come to a stop without colliding with an obstacle, $(v, \omega) = [vx, vy, \omega]^T$ and $\cdot v_m$ and $\cdot \omega_m$ are the maximum translational and angular velocities, respectively.

3.2.1 Dynamic Window

Since we take into account the limited accelerations of the vehicle motors, the search space of admissible velocities is reduced to the velocities that can be reached within the next time interval t.

We consider a three dimensional search space. A triple (v_x, v_y, ω) is considered admissible if the robot is able to stop before it reaches the closest obstacle on the corresponding trajectory. The dynamic window restricts the admissible velocities to those that can be reached within a short time interval dt given the limited accelerations of the robot.

3.2.2 Objective Function

The objective function

$$G(v,\omega) = \\ c_\sigma \left(c_\kappa \cdot \text{heading\_cost}(v,\omega) + c_\beta \cdot \text{dist}(c_v,\omega) + c_\lambda \cdot \text{vel}(v,\omega) \right), \tag{3.13}$$

where $c_\kappa > 0, c_\beta > 0, c_\lambda > 0$ are the gains of each part of the optimization function. Parameter c_σ has a smoothing function as it smoothes the weighted sum of the three components. The three components are target heading, clearance, and velocity. Target heading makes sure that the *angle* between the robot and the goal is zero. Clearance makes sure that the robots keep distance *dist* from obstacles is kept. The smaller the distance to an obstacle, the more the robot will move around it. Velocity *vel* is the forward velocity of the robot, maximizing the robot's velocity (Fox et al., 1997). Using distributed DWA and formation control the control law becomes:

$$u_i = u_i^f + u_i^{dwa}, \tag{3.14}$$

where u_i^{dwa} is the robot configuration that results in the minimal cost as determined by the DWA algorithm.

3.3 Robot Detection

The only input data we have for detection of other agents is coming from the laser scanner, therefore the returned matrix of 360 values needs to be processed. If no obstacles are detected the RPLidar returns the value of 'inf'. These are set to zero by our algorithm. The subsequent operation on the input matrix is returning the angles at which neighbouring agents are detected. We do this by returning the value of the index an object is detected. We only return the index to the output matrix z if the object is within a specified range around the desired distance. When we finished determining the angles at which neighbouring agents are located, we output the neighbours matrix z and use it as input for the third algorithm to determine the number of agents and their locations.

3.4 Obstacle Detection

As input for the obstacle detection we use two matrices, namely the laser scanner input, as well as the matrix containing the positions of the robots. In order to exclude the robots from the obstacle detection set, we subtract the robot detection matrix from the original matrix. Every nonzero value that is left should then be an obstacle. We can verify this by looking at the output of this step. The output is a vector

containing the distance and angle of reflections. The angle is represented by the index of each value in the vector.

3.5 Assumptions

We assume that initially the robots are approximately in the correct formation, in order to be able to identify which is which robot the easiest. For example, we have to assume that if the desired relative angle between robot 2 and 1 is 270 degrees, that the initial angle between those two robots is between 260 and 280 degrees when we perform the simulations. Idem with distance. For the distance we assume that during initialization, the robots are not at a distance greater than 1.5 meters.

During simulations and experiments we always assume that the desired formation is geometrically feasible.

During simulations and experiments we assume that the initial pose for each robot is known. During simulations using the dynamic window approach we localize the robots using amcl, which is a probabilistic localization system for 2D robots. We compare our sensor readings to a known map in order to estimate position then. In that case, the initial position is known with a variance of 0.5m in order to show the robustness of the approach.

For formation control we assume each agent can always obtain relative position information from its neighbours, meaning agent i has access to $p_j - p_i, \forall j$ in $\mathcal{N}_i$ as defined in Chan et al. (2018). We assume that always at least n-1 robots are detected for the control law to perform. If an obstacle moves through the formation this condition is not met. If an obstacle moves through the formation no control input is updated and re-initialization is going to have to occur.

We assume that one value is shared between the agents for the obstacle avoidance control. This means that there needs to be information exchanged between agents. In practice this means that the robots publish the obstacle avoidance state variable on a network, where, through the master node, the other agents can obtain this value by means of subscribing to the topic. We assume the communications channel will be able to handle the communication between the agents.

Chapter 4

Simulation and Experiment Design

In this chapter the simulation setup is discussed. We describe the simulation setups, sensors, hardware descriptions, type of simulations, and the simulation parameters.

4.1 Numerical Simulation Setup

The numerical simulation is done in MATLAB. In order to demonstrate the control laws we perform simulations with four agents. We consider a case where we have a world with two obstacles, described by the area of a circle. Around the circles we have another circle, which represents the safe distance between the agents and the obstacles. The radii of the obstacles and the safe distance are set to 0.75m and 0.5m respectively. The gains of the control law are determined to be $c_f = 10$ $c_P = 1$ $c_I = 0.9$ $c_\alpha = 400$ $c_\zeta = 10$. The desired relative positions for the formation are set to be $p_{21}^* = [0, 3]^T$, $p_{31}^* = [-2, 0]^T$ and can be seen in Figure 5.1.

For the simulations we assume the robots always spawn approximately in the correct formation, which makes identification easy. We use a saturation function to limit the acceleration and maximum velocity of the robots.

4.2 Gazebo Simulation Setup

In order to perform navigation in ROS, we need a map of the environment that we will operate in. The map can be empty, or filled with objects. For sake of generality we assume the environment to be very simple, a walled world , populated with a number of obstacles. The obstacles are convex and randomly distributed. The map is obtained using a single robot, moving around in our world, without obstacles, using manual control of the robot.
To perform an experiment, the formation needs to follow a certain trajectory. We let the local path planner compute a trajectory, and send the computed commands to the wheel actuators.

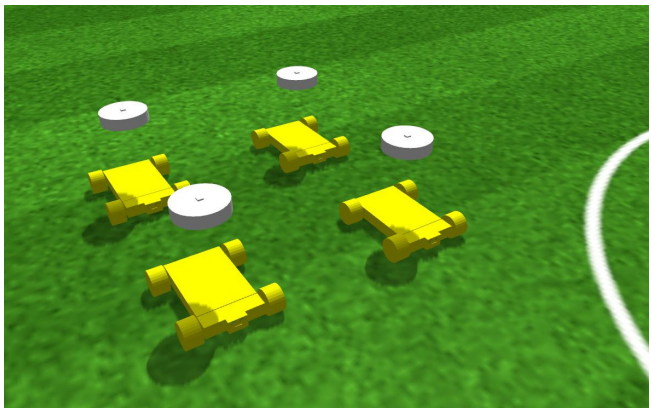


FIGURE 4.1: Group of four nexus robots.

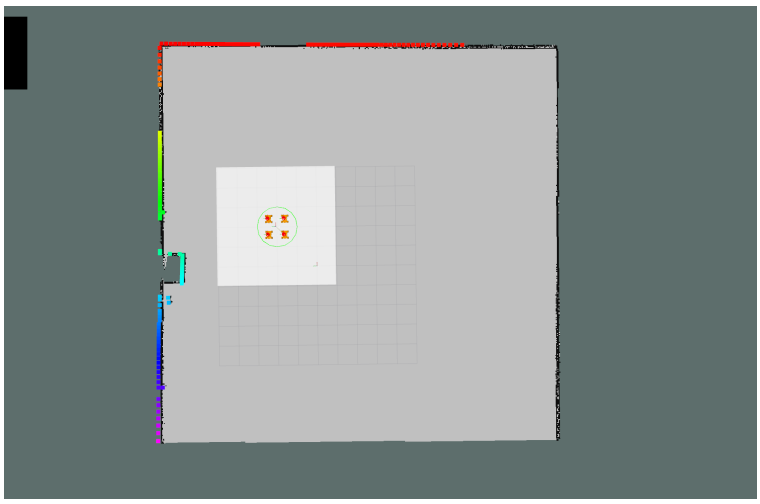


FIGURE 4.2: Lidar points nexus 1 visualized in RViz.

In Figure 4.2 we can observe multiple important elements. What we are visualizing is the map. The map is seen as a black outline. Somewhere on the map are the robots. The formation of robots is numbered as follows, from left to right, top to bottom: nexus 3, nexus 4, nexus 2, nexus 1. The lidar points are visualized as flat squares of size 0.2(m), using a rainbow color. Around the formation is a green circular outline, as well as a larger white square. The green outline is the footprint of the robot, which can be set to any shape. In this case a circle of radius 0.9(m). The white square around the outline is the local costmap of nexus 1. The middle of the costmap is using an offset from the centre of nexus 1 such that the center of the costmap coincides with the center of the formation P_{cen}. Every robot posseses such a costmap, so that the navigation is considered to be distributed. The last thing we can see is that the rainbow coloured lidar points shown on the outline of the map are interrupted. These interuptions, or blind spots, are caused by the other agents in the formation. The way that the blind spots are removed are through movement of the formation; when the formation moves, the blind spots move too and therefore all of them eventually become filled in.

4.2.1 Sensors

To simulate every sensor on the robot, a code snippet for each has to be added to the robot model xacro file. The code snippet describes the placement on the robot and

can be configured to match a real life equivalent sensor.

To simulate the RPLidar and IMU we use sensor plugins by (Hsu, 2014). The documentation on Gazebosim.org can be easily used to achieve any sensor configuration.

4.2.2 Robot Velocity Profile

Velocity profile of the Nexus model looks like the graph in Figure 4.3.

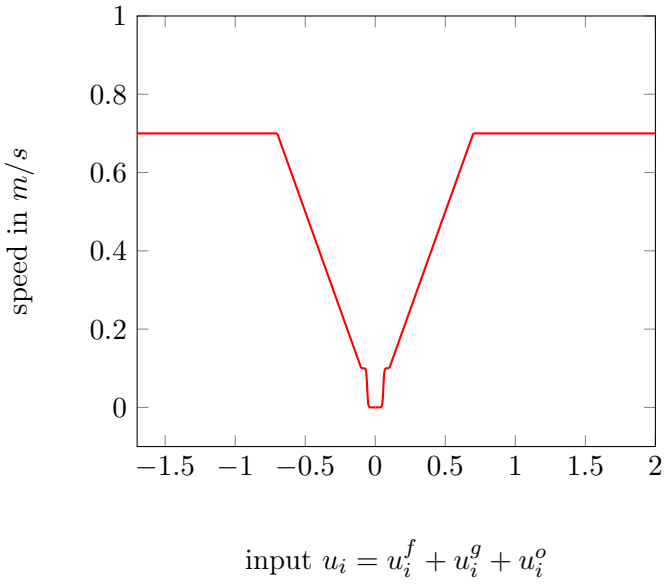


FIGURE 4.3: Velocity profile of the Nexus robots.

The velocity profile has four parts. Firstly, the range from 0 to 0.03 is set to zero since we do not want any jittery movements. Secondly, any input above the threshold of 0.03 is set to $0.1m/s$, which is the desired minimum speed. We set the speed to $0.1m/s$ using a logistic curve, which has an 'S' shape for a smooth transition. Thirdly, between the inputs of 0.1 to 0.7 we have an acceleration of $1m/s^2$. Lastly, the maximum speed is reached at $0.7m/s$. Higher speed causes the formation to display undesirable behaviour during obstacle avoidance.

4.2.3 Determining the Gain Parameters

During the implementation the gains of the control laws have to be determined. This was done using table 1 of "PID Control System Analysis and Design" (Li et al., 2006). In the table we can see the effects of every type of gain parameter. Each gain started at a value close to zero and was increased until desired behaviour was reached.

4.2.4 Configuration Space

Below, an example of the C-space for our robot formation is visualized, which can only move through translational movements.

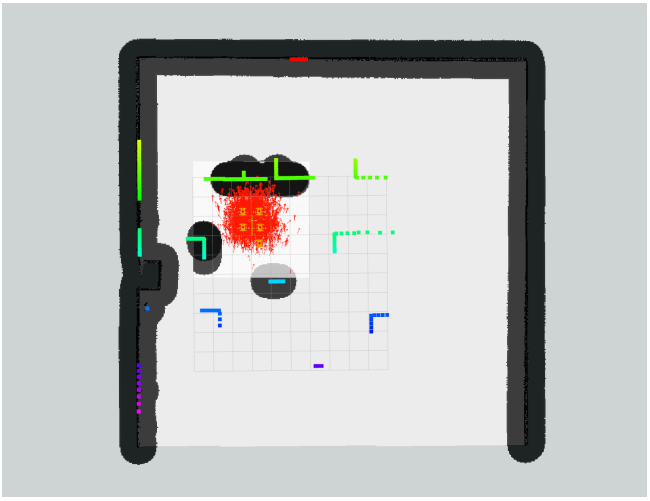


FIGURE 4.4: Example of the C-space for a robot formation, translation only. The C-space is obtained by sliding the robot along the edge of the obstacle regions.

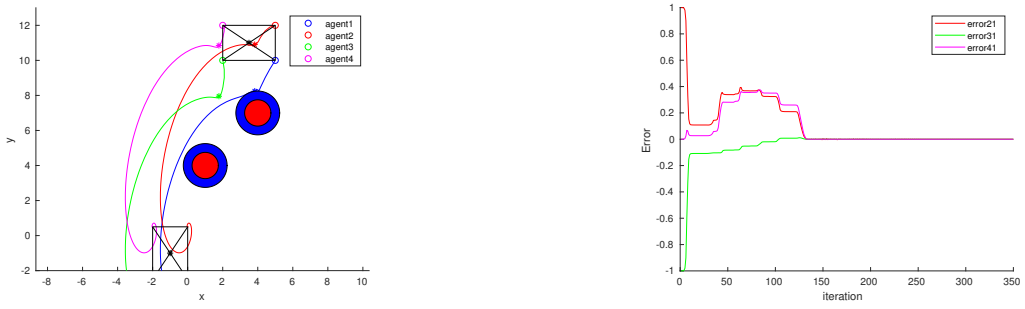
Chapter 5

Results

This chapter contains the results of the simulations and experiments that were performed in order to validate the approach. Several simulations were performed multiple times. We also comment on the robustness and the sensitivity of the system in section. 5.3.

5.1 Numerical Simulation Results

For the numerical simulation we initially consider only two cases for simulating different interesting scenario's. We start with the simplest case where only one of the agents encounters both obstacles. For the second case, two different agents 'hit' each obstacle, and the formation centroid passes over an obstacles. The result of the closed loop system using the control laws are found in Figure 5.1. From the figures, we can see that once an agent reaches the safe distance area, the obstacle avoidance is activated. In the neighbouring agent's their behaviour we see the diffusion of the obstacle avoidance behaviour. The neighbouring agents undergo a similar trajectory as the agent performing an obstacle avoidance manoeuvre. The deformation of the agents is seen in Figure 5.1(B). The deformation is minimized by the diffusive control law. A final observation can be made that the control law is able to steer the whole formation towards the goal state.



(A) Trajectory.

(B) Formation error.

FIGURE 5.1: Obstacle avoidance trajectory of four omni-wheeled mobile robots.

Figure 5.1 shows the error in the task of simultaneous obstacle avoidance and formation control. The first part of the graph shows the agents converging to a formation. The formation then travels toward its goal. At iteration 50, agent 1 encounters an obstacle, causing the rigid formation to break. After the obstacle avoidance manoeuvre is completed, the formation has converged again to its desired relative positions. The

formation then continues to move towards a goal. We then see that the formation once again encounters an obstacle. This time, the formation error is less. This is due to the collision path being less 'head on' and more at an angle. Head on collisions cause more error in the formation, whereas the system behaves more favourable during approaches at an angle.

TABLE 5.1: The numerical experiments success rates of the approach by Nelson et al. for SD=1

	SD = 1(m)		
	$c\alpha = 300$	$c\alpha = 400$	$c\alpha = 500$
cF = 8	43	49	45
cF = 9	45	46	42
cF = 10	44	49	39
cF = 11	44	49	44
cF = 12	43	42	49
	219/500	232/500	219/500

In Table 5.1 we see the results of our numerical simulations done in order to see the effect of different gains. The initial positions were taken as $p01 = [5, 10]^T$, $p02 = [5, 12]^T$, $p03 = [2, 10]^T$, $p04 = [2, 12]^T$. We perform the simulations 100 times, using a standard deviation of 1 with regard to the initial locations of the agents.

TABLE 5.2: The numerical experiments success rates for SD=2

	SD = 2(m)			
	cI = 0.7	cI = 0.8	cI = 0.9	cI = 1.0
cF = 8	46	42	43	45
cF = 9	42	44	43	45
cF = 10	46	42	43	43
	134/300	128/300	129/300	133/300

in Table 5.2, we see the result of the sensitivity analysis where the initial positions are determined based on a binomial distribution with a standard deviation of 2 meters. Among the other input variables are the formation gain and the integral gain parameters.

5.2 3D Simulation Results

The 3D simulation are performed in Gazebo, which provides a built-in physics engine called Open Dynamics Engine (ODE). ODE can simulate rigid body dynamics and collision detection, which satisfies the requirements for our application. Gazebo is used, so the results are more reliable for real world performance comparison. We consider several initial condition for simulating different interesting scenario's. Similar to the MATLAB simulations, for the first one, we consider no obstacles. In this simulation we can see that the obstacle avoidance behaviour is not diffused in the same way as the numerical simulations. The neighbouring agents seem to respond slower even when gain parameters are increased. Lastly, we observe that the group stabilization control law is able to steer the whole group towards the goal state within a reasonable amount of time and within the margin of error that was specified. In figures 5.2 through 5.5 we can observe four sub-figures. Starting form top left we

observe the traversed path of each agent. Top right we see the absolute error of each agent with respect to agent 1 over time. In the third sub-figure, the planned path of each vehicle is shown, and seen in the final sub-figure is the error of the x, y position of the formation's centroid.

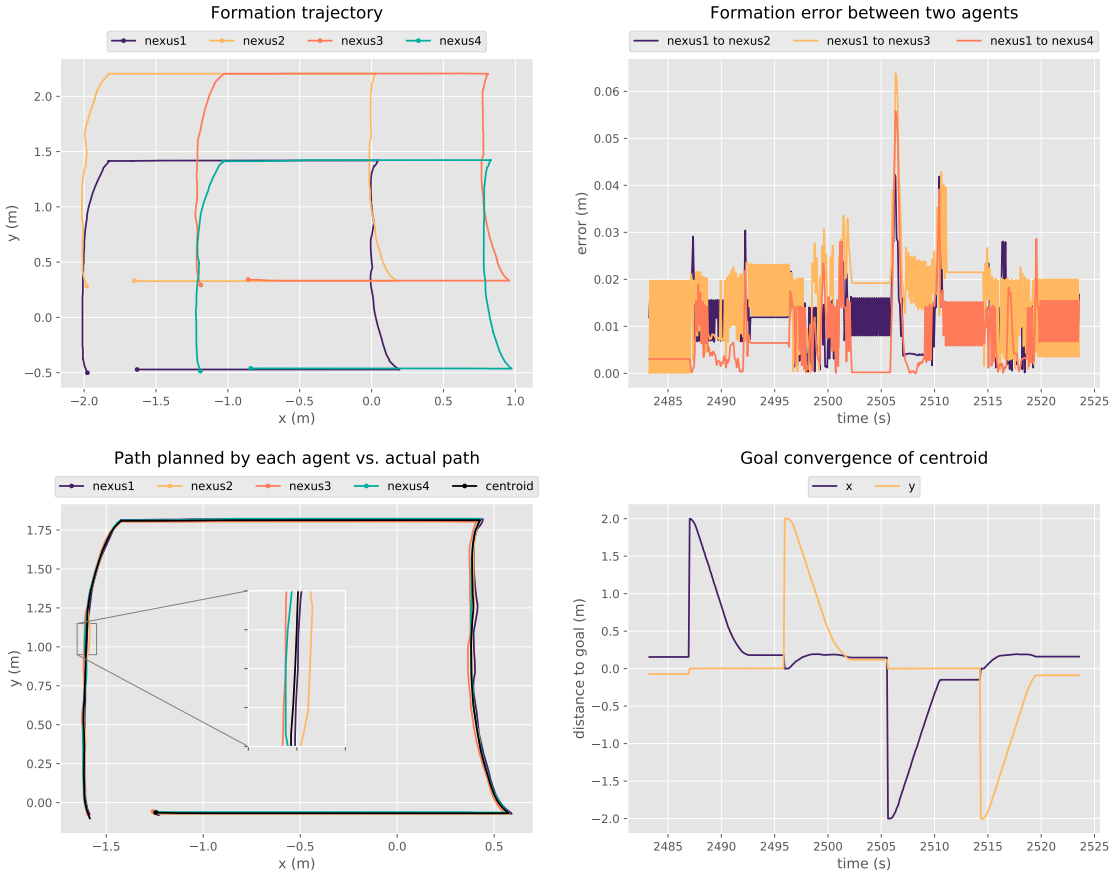


FIGURE 5.2: Experiment 1: driving along the boundary of a 2(m) by 2(m) square.

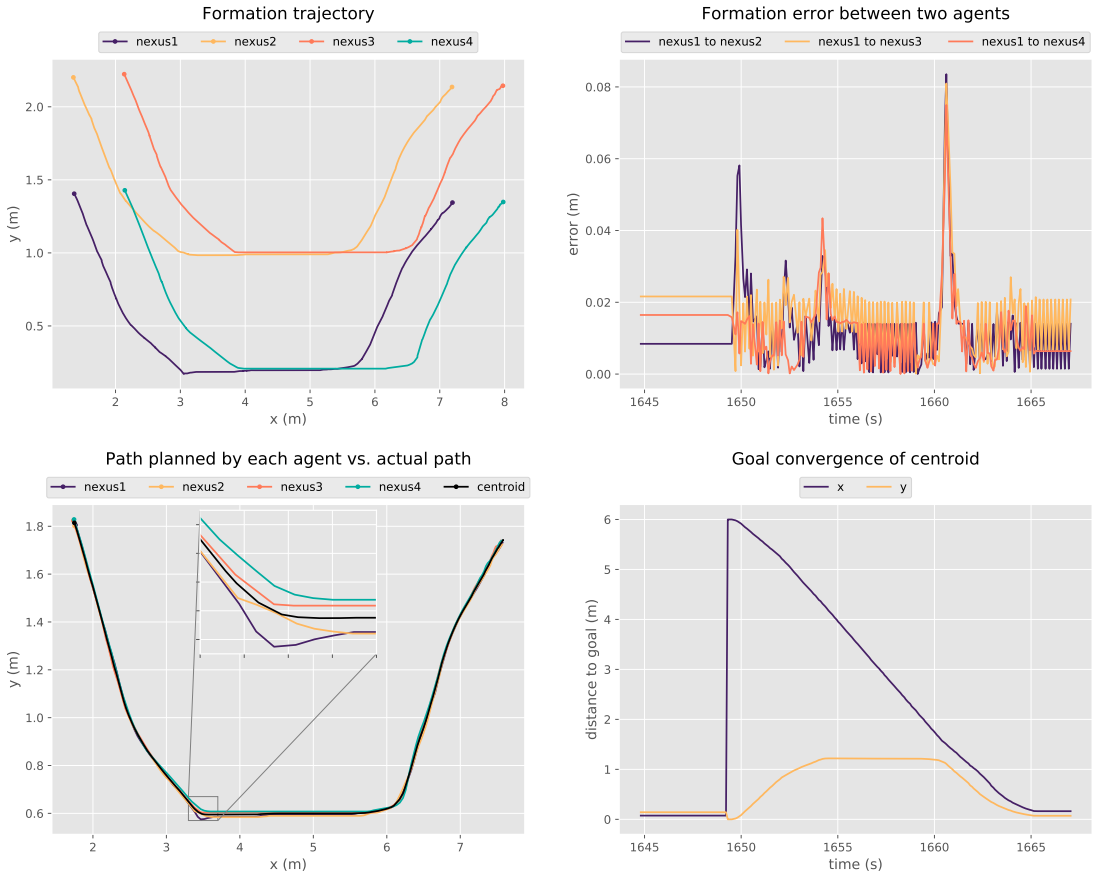


FIGURE 5.3: Experiment 2: driving around two obstacles.

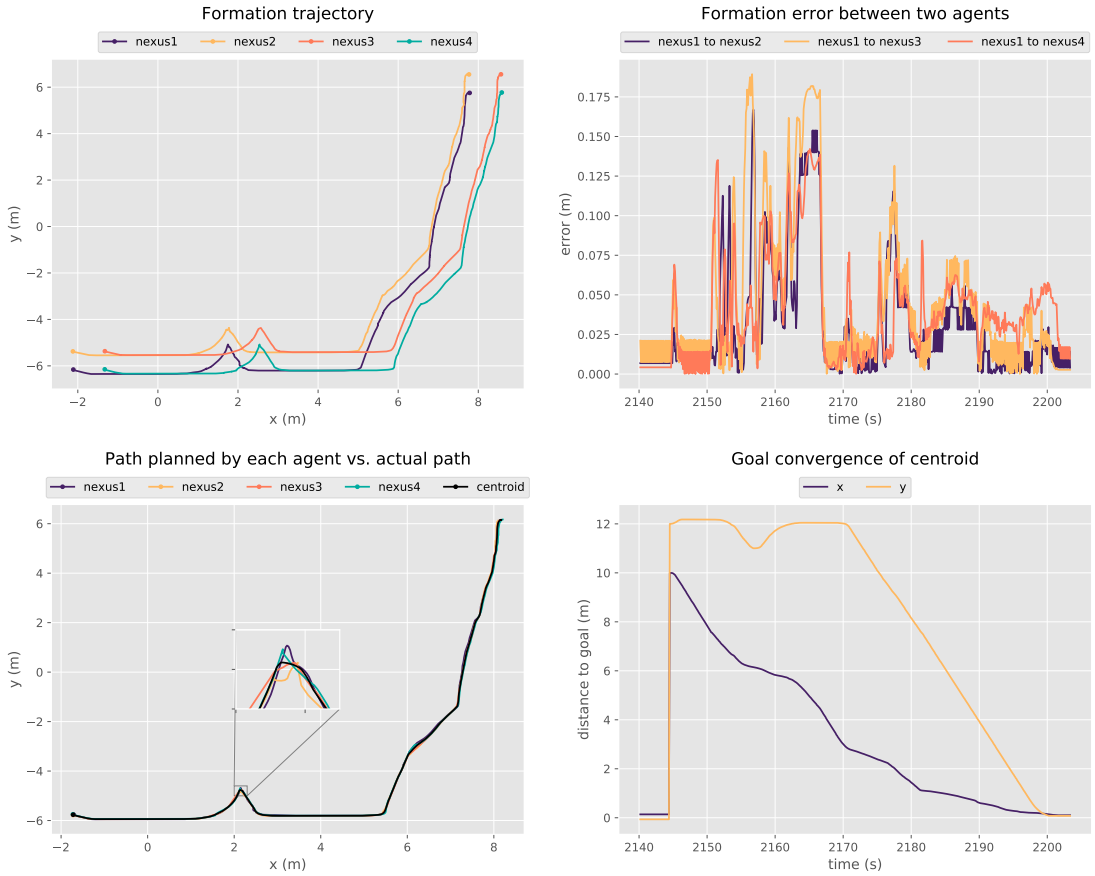


FIGURE 5.4: Experiment 3: enlarged area shows where the formation attempted to enter a corridor, but went around it instead.

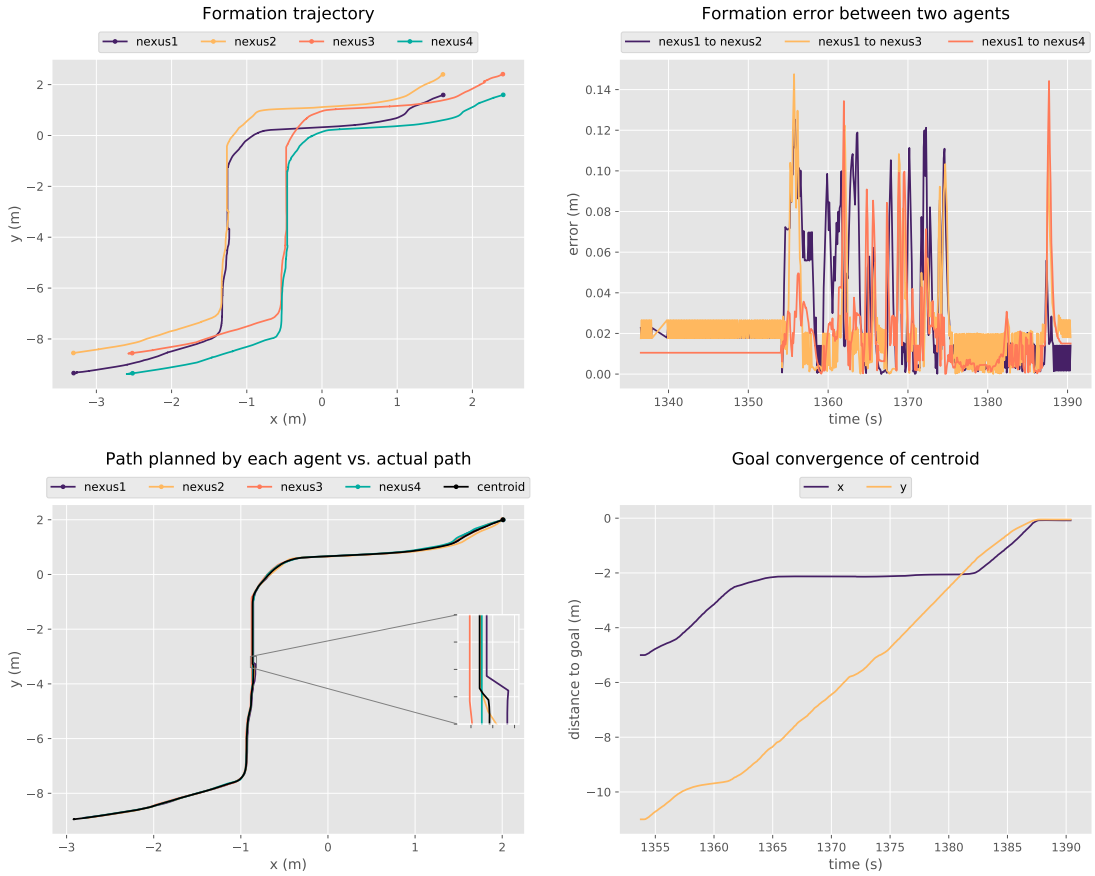


FIGURE 5.5: Experiment 4: formation travelling through narrow corridor (obstacles each side).

5.3 Robustness Analysis

To quantify the robustness and sensitivity of the system we compute the Root Mean Square Error (RMSE) and perform multiple runs. The Root Mean Square Error is defined as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}, \tag{5.1}$$

where $\hat{y}_i$ are the predicted values, y_i are the observed values, and n is the number of observations. This measure of error can be thought of as some kind of normalized distance between the vector of predicted values and the vector of measured values. RMSE is used to measure the error of prediction. So, how much the predictions differ from the predicted data. If the RMSE is sufficiently small given the problem context, then we can say the system performing well. If the RMSE is large, it generally means our model is failing to account for important underlying features of the physical system and the system is performing badly (Moody, 2019). During our numerical simulations the RMSE was always below 0.05. For our system this is acceptable, and means that the model accurately models the behaviour of the physical system. The results of the

robustness analysis for four experiment performed in Gazebo are present in Table 5.3. In all of the experiments, the desired inter-agent distance was set to 0.8 meters. The standard deviation of the error is small in all experiments. We can also see that the root mean square error does not show any particularly high values.

TABLE 5.3: Results of experiments in Figures 5.2, 5.3, 5.4, 5.5

distance agent 1-#	Experiment 1		Experiment 2		Experiment 3		Experiment 4	
	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD
2	0.0138	0.0062	0.0138	0.0064	0.0480	0.0365	0.0412	0.0292
3	0.0190	0.0097	0.0195	0.0097	0.0653	0.0476	0.0331	0.0233
4	0.0111	0.0079	0.0109	0.0081	0.0520	0.0336	0.0270	0.0200

Chapter 6

Discussion

This chapter discusses the objective of the research, what was done in the research, what was found, and what the results mean. Subsequently continuing with looking at the accuracy of the results, followed by discussing what could have been done differently in order to obtain more desirable results. Furthermore, a section is reserved for discussing limitations of the research.

The objective of this project was to deliver a solution to the problem of lacking a method for the nexus robots to avoid dynamic obstacles while maintaining formation shape. The objective was approached by formulating five sub questions. The initial phase of the project consisted of a literature study, finding the state-of-the-art approaches to tackling the task of obstacle avoidance.

The second phase consisted of implementing the distributed obstacle avoidance-formation control with coordinated group stabilization by Chan et al. (2018), which is based on a 2 degree of freedom (DOF) structure, namely the position in a 2D plane. It is also assumed that obstacle avoidance is done by a maximum of one agent at a time. The extra DOF, rotation of the formation is not considered. Once the implementation had been validated, the next phase was the Python implementation in ROS. The contribution of this thesis is the improvement of the control method and the development of a ROS package which provides the possibility of obstacle avoidance and formation control for a group of robots.

Localization of the robot in simulation was done using the wheel odometry as well as an imu-sensor, which improves the odometry estimates considerably.

6.1 Limitations

Even though the obstacle avoidance state variable ζ is shared between the neighbouring agents, the approach by Chan et al. (2018) as well as the dynamic window approach still causes the formation to deform during obstacle avoidance manoeuvres.

Failure is possible even if there is a valid path to the goal. The formation can get 'stuck' when an obstacle is concave. The formation can also get stuck when there is a clear path to the goal, but the path passes through a narrow doorway. The formation, which has a width of 1.6 meters, can pass through a two meter gap using the DWA.

The obstacle avoidance algorithm is based on a purely reactive algorithm, and needs a higher level path planner to perform well in complex environments such as a maze. A higher level path planner would be able to identify intermediary goals.

The current approach requires communication between agents for the obstacle avoidance manoeuvres in order to 'diffuse' the action. The system would be more robust if it did not rely on such communications.

The system is currently designed for three and four robots. We want to make this system more generic. A future version would be able to have the same functionality for 'n' robots. The problem then is how to determine the centroid of the formation

for group stabilization. This would be most efficient using a master node, running on an external computer.

The current approach depends on accurately chosen gain parameters and configuration of the DWA parameters. Manual tuning of the parameters can be difficult for large systems of robots.

Due to time limitations we were not able to perform experiments in the real world, as well as thorough comparison of the two approaches in Gazebo. One approach was tested in numerical analysis and one approach was tested in Gazebo. Therefore we can not make definite predictions of the performance differences, however we hypothesize the dynamic window approach would perform better in both numerical and 3D simulations.

Chapter 7

Conclusion

In this thesis we implemented a distributed control law for achieving formation while avoiding obstacles for a group of agents. In this implementation, the agents are so-called Nexus robots, which are omni-wheeled vehicles. They are modelled without taking angular velocity into account. Combining the formation control and obstacle avoidance with a coordinated group controller it has confirmed the possibility of steering the formation towards a goal destination while maintaining formation shape before, during, and after multiple obstacle avoidance manoeuvres. We show that local path planning can lead to convergence for an autonomous system.

The main novelties introduced in this thesis are that the system works without requiring GPS or a birds-eye view, we instead use inter-agent distance. We also combined formation control and the dynamic window approach into one control system.

Experiments have shown that we are able to steer the formation to a desired goal location in interesting scenarios, while maintaining formation shape during and after obstacle avoidance manoeuvres.

Experiments have shown that the system is robust against initial conditions, as well as different environments.

The system is easy to maintain and modify. The system is robust against failures, based on the sensitivity analysis. The system can operate on low-level controllers with a small amount of RAM and CPU resources since it does not rely on for example computer vision. Therefore, more power inside the batteries will be saved and can instead be used by the wheel motors or other moving parts.

Using this framework of separating the controller tasks and adding them together to form the final control law provides a good basis for further research, where different control laws can be easily interchanged.

Chapter 8

Future Work

In this thesis a dynamic window approach is proposed for path planning for a formation of omni-wheeled vehicles. The results and their limitations give rise to new research opportunities, some of which are proposed in the following sections.

8.1 Future Work

Hereby a list of recommendations in terms of future projects that can be derived from this thesis. We hope to see an implementation of different sampling based path planning approaches in order to find the best one for different use cases. We can also extend these cases from 2-dimensional navigation, to aerial navigation. Another future work that should be done is the merging of more sensor data, allowing for more accurate sensing. We would like to see an application which combines lidar with camera's for example. With camera's it is much easier to see the difference between another vehicle and an obstacle. With camera's it is possible to only see a 2D representation of the 3D world, however, combining the camera with lidar messages, we would be able to improve the robustness of the system immensely. There are also alternatives, such as the intel RealSense camera, which is a more expensive alternative and 5 sensors still would have to be merged to cover 360 degrees. Also, the processing power of the Arduino microcontrollers could possibly not process the large data stream produced by four or even five of these cameras.

8.1.1 Real World Application

The control system was supposed to be tested on the real life Nexus robots. A map of the lab was produced using SLAM, however the nexus robots only have a lidar and wheel encoders. The wheels sometimes have a certain amount of slip, causing the odometry information to have an error. An extra sensor such as an IMU should be added for measuring orientation. The SLAM system worked as intended, however the RPLidar does not identify all important features in the environment, partly because it only scans at one specific height and there were a lot of table legs and chair legs.

Appendix A

The Code

The source code can be found at https://github.com/Pytrik/FormationSim.

Bibliography

Barraquand, J. & Latombe, J.-C. (1991, December). Robot Motion Planning: A Distributed Representation Approach. *The International Journal of Robotics Research*, *10*(6), 628–649. doi:10.1177/027836499101000604

Borenstein, J. & Koren, Y. (1988). High-speed obstacle avoidance for mobile robots. *Proceedings of the 3rd International Symposium on Intelligent Control*, 382–384.

Borenstein, J. & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, *7*(3), 278–288.

Burgard, W., Stachniss, C., Bennewitz, M., & Arras, K. (2011). Introduction to Mobile Robotics: Robot Motion Planning. Freiburg: University of Freiburg. Retrieved from http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf

Chan, N. P. K., Jayawardhana, B., & Scherpen, J. M. A. (2018). Distributed Obstacle Avoidance-Formation Control of Mobile Robotic Network with Coordinated Group Stabilization. *23rd International Symposium on Mathematical Theory of Networks and Systems (MTNS 2018)*, 722–725. Retrieved from http://mtns2018.ust.hk/

Fox, D, Burgard, W, & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33. doi:10.1109/100.580977

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

Gerkey, B. Gmapping. Retrieved from http://wiki.ros.org/gmapping

Hsu, J. (2014). Gazebo plugins in ROS. Retrieved from http://gazebosim.org/tutorials?tut=ros_gzplugins

Karaman, S. & Frazzoli, E. (2011, June). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894. doi:10.1177/0278364911406761

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles* (pp. 396–404). Springer.

Koren, Y. & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings, 1991 IEEE International Conference on Robotics and Automation*, 1398–1404.

LaValle, S. M. & Kuffner Jr, J. J. (2000). Rapidly-exploring random trees: Progress and prospects. *Workshop on the algorithmic foundations of robotics; Algorithmic and computational robotics, new directions*, (4), 293–308.

Li, Y., Ang, K. H., & Chong, G. C. Y. (2006). PID control system analysis and design. *IEEE Control Systems Magazine*, *26*(1), 32–41.

Ljungqvist, O., Evestedt, N., Axehill, D., Cirillo, M., & Pettersson, H. (2019, December). A path planning and path-following control framework for a general 2-trailer with a car-like tractor. *Journal of Field Robotics*, *36*(8), 1345–1377. doi:10.1002/rob.21908

MarvelMind. Retrieved from https://marvelmind.com/

Moody, J. (2019). What does RMSE really mean? Retrieved from https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

Murray, S., Floyd-Jones, W., Qi, Y., Sorin, D. J., & Konidaris, G. (2016). Robot Motion Planning on a Chip. In *Robotics: science and systems*. Retrieved from https://www.researchgate.net/profile/George_Konidaris2/publication/304532889_Robot_Motion_Planning_on_a_Chip/links/57f6767708ae280dd0bb29e5.pdf

Rosolia, U., Carvalho, A., & Borrelli, F. (2017). Autonomous Racing using Learning Model Predictive Control. *2017 American Control Conference (ACC)*, 5115–5120. doi:10.23919/ACC.2017.7963748

Routh, E. J. (1877). *A Treatise on the Stability of a Given State of Motion, Particularly Steady Motion*. London: Macmillan and Co.

Shapiro, L. G. & Stockman, G. C. (2001). *Computer Vision*. Prentice Hall.

Slamtec. (2016). RPLidar A1 Introduction and Datasheet. Retrieved from https://www.robotshop.com/media/files/pdf/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf

Susnea, I., Minzu, V., & Vasiliu, G. (2009). Simple, real-time obstacle avoidance algorithm for mobile robots. *8th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics (CIMMACS'09)*, 24–29.

Tee, K. P., Ge, S. S., & Tay, E. H. (2009). Barrier Lyapunov Functions for the control of output-constrained nonlinear systems. *Automatica*, *45*(4), 918–927. doi:10.1016/j.automatica.2008.11.017

Vannoy, J & Xiao, J. (2008). Real-Time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments With Unforeseen Changes. *IEEE Transactions on Robotics*, *24*(5), 1199–1212. doi:10.1109/TRO.2008.2003277LK-https://rug.on.worldcat.org/oclc/4798731616

Wan, E. & van der Merwe, R. (2006). Sigma-Point Filters: An Overview with Applications to Integrated Navigation and Vision Assisted Control. *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, 201–202. doi:10.1109/NSSPW.2006.4378854