Institute for Computer Science VII
Robotics and Telematics

Bachelor's thesis

Extending Giovanni Controller with Formation Control functionality

Thomas Schmitt

March 2018

First supervisor: Prof. Dr. Andreas Nüchter
Second supervisor:

Abstract

This work is a practical report about building a Formation Controller upon a Path following robot. Therefore, similar robots, called "Volksbot", are used. Each one has its own path controller, which is based upon Giovanni Indiveri's work and thus referred to as "Giovanni Controller". As the performed tasks all belong to the category of Leader-Following, it is self-evident to implement a Leader-Follower formation. Whereas the Leader is equipped with full knowledge, the Follower has only access to local information. In order to *maintain* the formation, a step-by-step evolution and extension of the controller is described. The main principle for that is to keep the controller's extensions *as small and simple as possible while* being able to perform the task.

Contents

Chapter 1

Introduction

Nowadays, single-agent based robots are driving on the mars with a huge delay in communication to their control center. Unmanned underwater vehicles (UUV) using Simultaneous Localization and Mapping (SLAM) algorithms can scan the ground of seas not depending on human controllers. In the area of autonomous robots, Google is developing its self-driving cars. All in all, single mobile systems are advancing rapidly in order to fulfil a huge variety of tasks.

Most of them are highly specified for their assignment. This is the main reason for the trend towards multi-agent systems. In these environments, cheaper robots fulfilling simple tasks can be used. The constellation of different or similar robots replaces the specialization. Agents acting on their own can not fulfill every task possible e.g. refueling an airplane during the flight. In order to be able to solve such tasks, communication between several agents is required. In the example given above, the speed as well as the trajectory of both airplanes has to be matched. In order to guarantee the success of multi-agent systems, more complex controllers are required. These are mainly referred to as "Formation Controllers". They do not just use their own information but also those gathered by others in the formation. This work presents one controller using simple techniques in order to accomplish the tasks of "Following another robot on the same track in a constant distance" and "Driving besides a robot with a constant offset vector".

Chapter 2

Background

2.1 Giovanni Controller

This section shows the functionality of the used path controller. It was originally published by Giovanni Indiveri in [8]; therefore, it is referred to as "Giovanni Controller". The controller is designed for nonholonomic robots, bearing the idea in mind of a robot constantly moving forward. This idea's basis results from the mechanical limitations of the robots for which it is designed. As these robots have been drafted for air and underwater environments, their architecture allows them only to move in one direction and prohibits them from turning on spot [8].

Model

The model depicted in 2.1 describes the robot's kinematic by using Cartesian Coordinates:

$$\begin{aligned} \dot{x} &= u\cos\phi \\ \dot{y} &= u\sin\phi \\ \dot{\phi} &= \omega = u\frac{\tan\psi}{l} \end{aligned} \tag{2.1}$$

In the equation above, u is the velocity of the vehicle, ϕ describes the angle between the x-axis and the robot's moving direction. In order to prevent the robot from turning on spot, the steer angle ψ is multiplied with the velocity. Using the following transformations

$$\begin{aligned} e &= \sqrt{x^2 + y^2} \\ \theta &= ATAN2(-y, x) \\ \alpha &= \theta - \phi \end{aligned} \tag{2.2}$$

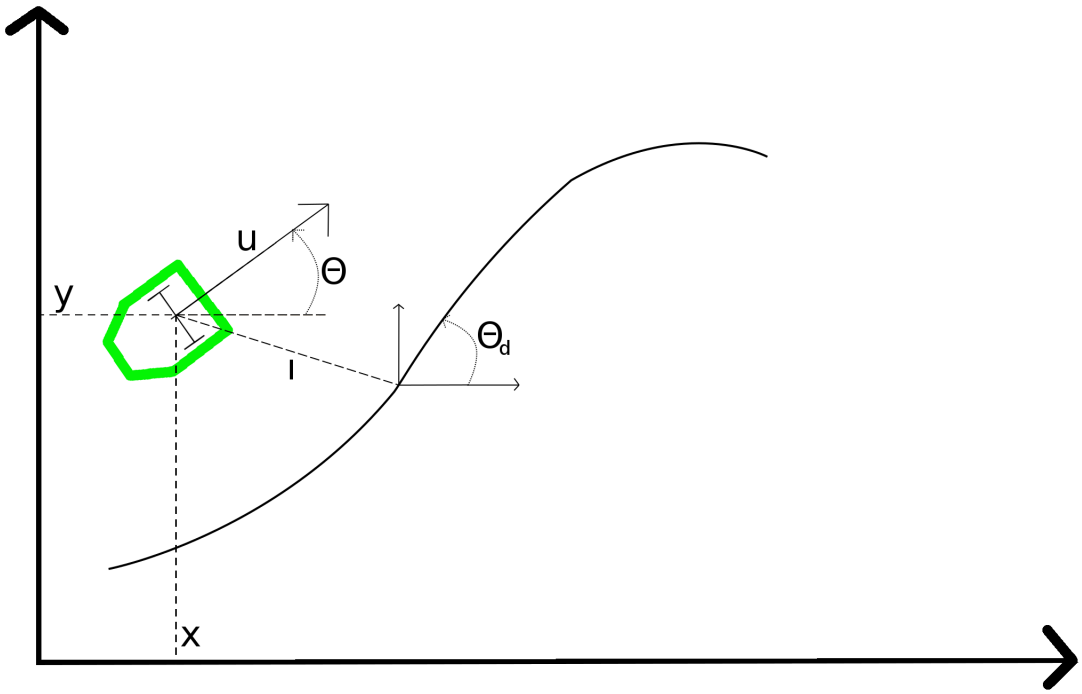


Figure 2.1: The mathematical model. Source: [7]

the model can be transformed to the polar kinematics (2.3) with e being the distance to the target point.

$$\begin{aligned} \dot{e} &= -u\cos\alpha \\ \dot{\alpha} &= -u(c - \frac{\sin\alpha}{e}) \\ \dot{\theta} &= u\frac{\sin\alpha}{e} \end{aligned} \tag{2.3}$$

The next step is to find a function so that e, α and θ converge to 0. Therefore, de Wit's approach (2.4) is used.

$$u = \gamma e \tag{2.4}$$

In order to keep the robot moving forward, γ is chosen as $\gamma > 0$ and c as:

$$c = \sin\frac{\alpha}{e} + h\frac{\theta}{e}\frac{\sin\alpha}{\alpha} + \beta\frac{\alpha}{e} \tag{2.5}$$

The constants h and β of 2.5 are selected following Indiveri 2.6 to ensure convergence and make the robot drive towards a point.

$$h > 1; 2 < \beta < h + 1 \tag{2.6}$$

Thereby, the speed is kept at a constant level as the convergence is independent from the speed.

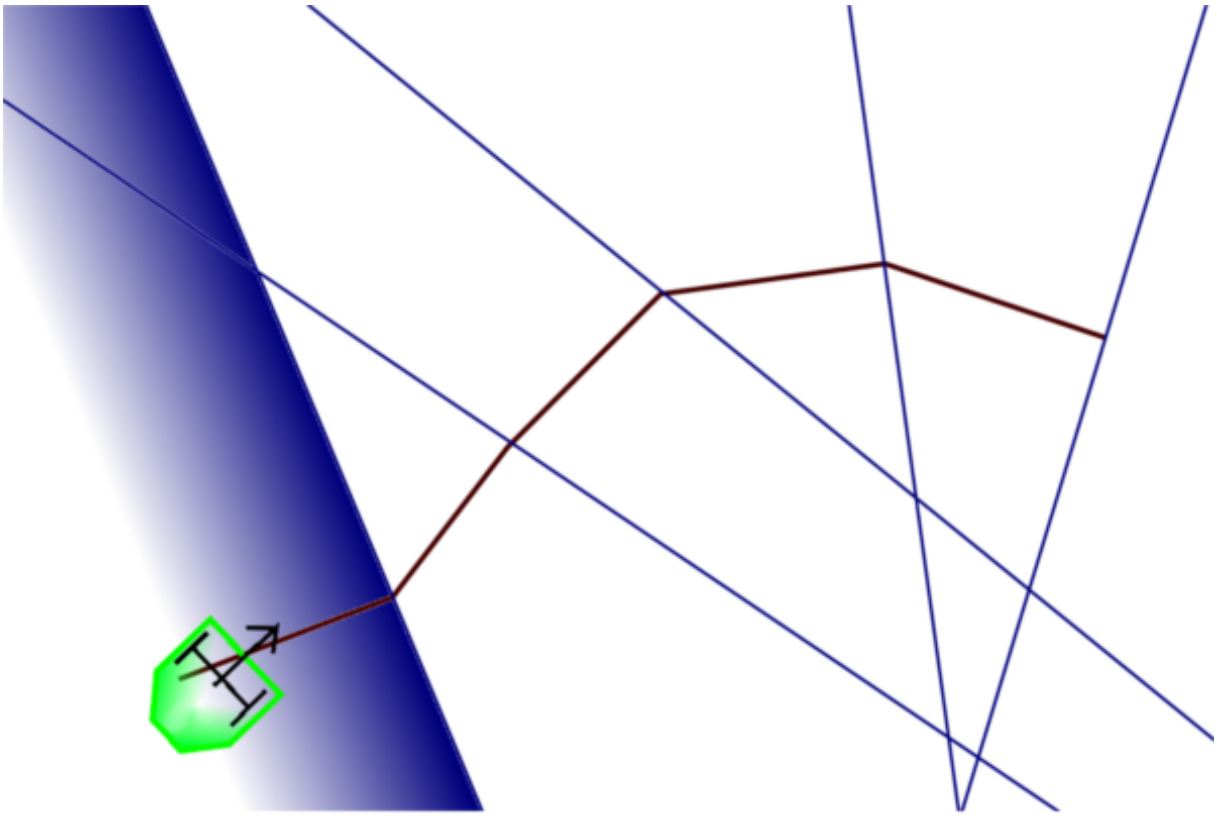


Figure 2.2: This graphic shows how the path is constructed. The wine-red lines represent the path. Each intersection with a blue line, which is vertical to the wine-red ones, represents one path point. When the robot crosses a blue line, the following point becomes the new target point for the controller. Source: [7]

A path, a list of points, can be described by linear approximations between each pair of adjacent points. By changing the distance e to the remove of the current position to that line, enables the controller to follow the path instead of driving towards a single point. The controller then prevents the robot from turning and driving backwards by adding a vertical line through each point to the previous calculated path segment. When the robot crosses this line of the current target point, the next point becomes the new one 2.2.

Positioning

Two different mechanics are used to obtain the current position. The first one is dead reckoning (2.7). It is the sum of all movements from a given starting point.

$$P(t) = P(0) + \int_0^t v \tag{2.7}$$

$$P_{k+1} = P_k + v_k * \Delta(t) \tag{2.8}$$

As the robot can only evaluate discrete functions, equation (2.7) has to be changed to (2.8). Wheel encoders measure the number of revolutions for each wheel - the movements - and therefore enable the robot to calculate its position.

$$\begin{aligned} \Delta d'_{k,r} &= (R + \frac{b}{2}) * \Delta\theta \\ \Delta d'_{k,l} &= (R - \frac{b}{2}) * \Delta\theta \\ \Delta d_k &= \frac{\Delta d'_{k,r} + \Delta d'_{k,l}}{2} \\ \Delta\theta_k &= \frac{\Delta d'_{k,r} - \Delta d'_{k,l}}{2} \end{aligned} \tag{2.9}$$

In (2.9), d' is the displacement for each wheel with R being the distance from the instantaneous center of curvature (ICC) to the wheels axis' center, and b representing the distance of the wheels to each other. θ is the orientation relative to the global frame [6].

$$\begin{aligned} x_{k+1} &= x_k + \Delta d_k \; cos(\theta_k + \frac{\Delta\theta_k}{2}) \\ y_{k+1} &= y_k + \Delta d_k \; sin(\theta_k + \frac{\Delta\theta_k}{2}) \\ \theta_{k+1} &= \theta_k + \Delta\theta_k \end{aligned} \tag{2.10}$$

Dead reckoning has an unbound drift error which renders formations based on it impossible. As a robot is not able to determine its position and thus can not decide whether it is in the correct location. Dead reckoning is often extended with other position systems like GPS in order to resolve this problem. In this work, AMCL, a Monte Carlo algorithm, is used.

AMCL is a combination of dead reckoning and a particle filter for a laser sensor. It assumes the current dead reckoning position comparing the data collected by the scanner and a local (cost) map, which contains the distance to obstacles. The most probable point in the surroundings is used and set as the new position. This method was already used in a previous work [7]. It shows that the error of AMCL is bound as long as the vehicle is in the bounds of the map. In the following, the max. error is assumed to be: $e_{max} \leq 10cm$.

2.2 Formation Theory

2.2.1 Definition of Formation

A formation is defined as the following: When at the beginning between each pair of adjacent robots a finite distance exists and it is bound during the period of time [10]. In other words, vehicles in a formation do not diverge from their neighbouring objects. The first task of this work is defined by keeping a constant distance between the vehicles, and for the second one a specific vector has to be maintained.

2.2.2 Leader-to-Follower

Formations can be implemented using different strategies like local sensing, global grouping, behavior based and Leader-To-Follower [11]. Each one has its own capabilities and thus a special application field.

The tasks described in this work do best correspond to the Leader-To-Follower strategy. One robot has the knowledge of the path, whereas the other one follows the Leader while being bound to specific restrictions. This is a typical setup for a Leader-to-Follower based formation. One controller is equipped with global information, and the followers have to position themselves relative to their direct leader(s).

According to that, formations are often described using graphs, in which each node represents a robot and edges indicate adjacent robots. A depiction of the communication flow and the function of a node is made possible by using directed edges (see Figure 2.3). Furthermore, each

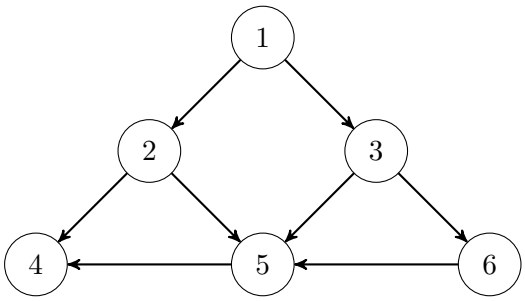


Figure 2.3: Each number represents a robot and each directed edge a Leader-Follower relationship with the base of arrow being the leader and head being the follower. Example: 1 being the formation leader with 2 and 3 following it.

acyclic formation can be divided into several local formations. This can be achieved by iterating each follower and isolating it and its leaders from the graph. Figure 2.4 shows a small example. The more connection there are to a node, the more restricted it is as each connection stands for

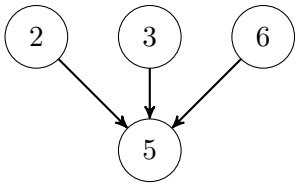


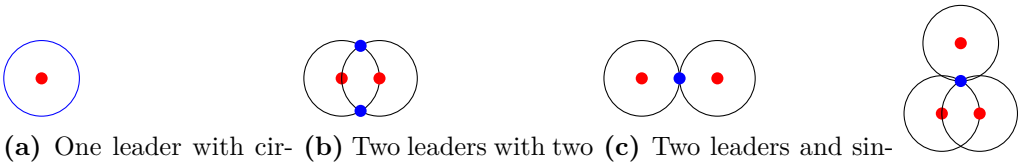
Figure 2.4: 2.3 reduced to local dependencies for robot 5.

a restriction. Too many of them leads to the fact that they either become redundant or that it is impossible to fulfil the formation. As the following equation applies: (2.11).

With P_F, $P_{L,k}$ being the position of the follower or the k-th leader respectively and $o_{F,k}$ the offset of k-th leader to the follower F, then the following applies:

$$P_F = \bigcap_{k=0}^{n} (P_{l,k} + o_{F,k}) \tag{2.11}$$

A special kind of formations are string formations. The name is derived from its basic idea: Given the condition that the distance between robots maintains constant, so that strings, which are attached to each Leader-Follower pair, remain tense, these formations are called string-stable-Formation, e.g. the march of an army battalion [10]. Thus, the restrictions are specified as distances in the euclidean metric. The string-dependency is represented by a circle with the leader being the center and the circumference on which every possible position is located. This leads to the options shown in 2.5.



(a) One leader with circle

(b) Two leaders with two points

(c) Two leaders and single point

(d) Three leaders with single point

Figure 2.5: Here the different options for intersecting two or more circles are shown. Each of these figures describes a formation with the red point as leader and the blue edges or points as allowed positions. In (a) the followers position is not more specifically detailed and in (b) two different points are permitted. The formations in (c) and (d) only allow a single point for the follower.

Only formations, in which at least one point exists that does not violate any restriction, are taken into account in the further work. In 2.5 the formation-position points are represented as blue points. Each follower-leader relationship can be reduced to three leaders and one follower (see 2.5(d)), as each further leader-circle has to intersect the same point as well. So every reduced local leader-follower formation consists of a single follower accompanied by up to three leaders. Following the structure of these local graphs, small communication networks with reduced complexity and data rate can be built as the information has only to be shared in the local-neighbourhood, while maintaining a global formation.

2.2.3 Error in formation

In order to evaluate the performance of a string formation controller, the equation (2.12) is used to calculate the error e.

$$e = \Sigma_{k=0}^{t_{end}} d(P_{L,k}, P_{F,k}) \tag{2.12}$$

with $d(P_{L,k}, P_{F,k})$ being the euclidean distance. x_o and y_o are the elements of the offset vector between leader and the optimal follower position given by the restrictions.

$$d(P_{L,k}, P_{F,k}) = \sqrt{(x_{L,k} + x_o - x_{F,k})^2 + (y_{L,k} + y_o - y_{F,k})^2} \tag{2.13}$$

Although this equation cannot be used to compare different formations to each other, but it can be used for controllers of the same formation. The controller having the smallest error, is performing in the best way. The best way to measure the positions $P_{L,k}$ and $P_{F,k}$ is by using global positioning systems[1].

[1]Global system is here seen in the sense of applicative environment and can measure the absolute position of the vehicle

Chapter 3

Implementation and Testing of the Formation Controller

3.1 Design Leader Controller

In this work, a simple Controller accomplishes the tasks. It builds on the Giovanni Controller described above in 2.1. The communication is one-directional to the other vehicles, so that the leader does not know whether the formation is hold or a follower is falling back due to an issue etc.. The controller sends the following information through the network:

- "Path": the next Path point P_i it drives to: $P_i\,(x_i/y_i)$
- "Position": its own Position measured either by AMCL or Odometry: $P_L\,(x_L/y_L)$
- "FinishNotification": a notification when the task is complete, in order to enable the follower to stop

The logic, required for keeping up the assigned formation, is implemented in the Follower Controller; this allows the system to be scalable as each follower knows how to complete its own task. The only configuration that needs to be updated is the launch of another "nimbro_network/udp_sender"-node as each of them is connected to a single ip-address. In this work, only two robots are used, so there is no need to add a TopicListener to the leader which launches - through its callback method - the required node using the ros-api[1].

3.2 Task Following Leader in Distance

As mentioned in 3.1, the main work for keeping up the formation is done by this controller. For each task an individual extension for the Giovanni Controller is used, which fulfils the task.

The first task consists in following the leader on the same path holding a constant distance. The second assignment is to follow the leader in a time independent offset $o(x|y)$.

[1]The referred API functionality can be found at: http://wiki.ros.org/roslaunch/API%20Usage

3.2.1 Design Distance Follower Controller

Task Definition

The follower takes the same path as its leader. At all times the distance between those two robots should remain stationary at a certain d_{target}. When the leader reaches the track's end, the follower shall maintain d_{target} in order to simulate an unexpected interruption of the leader.

Metric for Distance between Leader and Follower

The aforementioned task uses the structure of the path. The euclidean metric is applied to each segment between leader and follower.

$$d_{L,F} = \sum_{i}^{j-1} ||P_{i+1} - P_i|| \tag{3.1}$$

In equation (3.1), i is the current segment index of the follower, and j is the last one passed by the leader. The distances of each robot to its target point is missing. These are described with the following equations.

$$d_{j,L} = ||P_L - P_j|| \tag{3.2}$$

$$d_{F,i} = ||P_i - P_F|| \tag{3.3}$$

The distance d between the robots is the sum of the equations (3.1), (3.2) and (3.3).

$$d = d_{F,i} + d_{L,F} + d_{j-1,L} \tag{3.4}$$

Formation

The controller just needs to connect all information it receives. From the topic "Path" it can build up the track from the starting point to the one which is tracked by the leader at this moment. This results in j, from (3.4), being the number of received points minus 1.

Algorithm 1 shows the follower's control cycle. It receives the communication message from the leader in an asynchronous manner, due to that it is not included in the algorithm. Each iteration synchronizes the follower's own position and calculates d in function "calcDistanceToLeader()" according to (3.4). As the track is divided into segments, the controller releases each new one as soon as the distance to the leading robot exceeds d_{target}. Only when the next section is released, the robot is moving. Otherwise, it will hold the position as it assumes that the leader stopped for some cause. "notFinished" changes its value to false when a "FinishedNotification"(see 3.1) is received.

Available Improvements

The basic design is far from perfect and kept simple. Therefore, several improvements are available to address downsides for this approach. Some of them are discussed in this section.

Algorithm 1 Basic Control cycle

```
Require: notFinished = true
  releaseNextSegment ← false
  while notFinished do
    P_F ←updatePosition()
    d ← calcDistanceToLeader()
    if isNewSegment() then
      releaseNextSegment ← false
    end if
    if (not releaseNextSegment) and d > d_target then
      releaseNextSegment ← true
    end if
    if releaseNextSegment then
      controller.followPath()
    else
      stop()
    end if
  end while
```

The first problem addresses offsets. The current solution uses the same constant speed as the formation leader. If the beginning distance between the mobile systems is bigger than d_{target}, it will remain bigger, thus an offset emerges. The same applies to increasing distance during the execution. An addition would be to increase the speed when the follower falls back too far. This approach will be used in 3.2.2.

A closer look on the algorithm reveals that the distance in the formation is depending on the resolution of the path. This can be solved quite easily. As the track is linear approximated, the resolution of the path can be raised arbitrarily. For example, if the distance between two adjacent points is higher than a certain margin, the controller can split this segment into several ones by adding new points on the line.

Evaluating performance of formation

No measurement tool for the evaluation of this formation type has been introduced. The aforementioned error function (2.12) does not qualify for this example in its current form, as the offset is not a vector between the two positions of the robots. In order to solve that issue, it is adapted by using the distance function (3.4) for this task. Using it in combination with the mean of error, it leads to:

$$e = \frac{1}{n} * \sum_{0}^{n} (d - d_{target}) \tag{3.5}$$

As in (2.12) a better performance will lead to a smaller value. If taking into account that the result is independent from the starting conditions, the step responds will be excluded.

"Eight" - Path

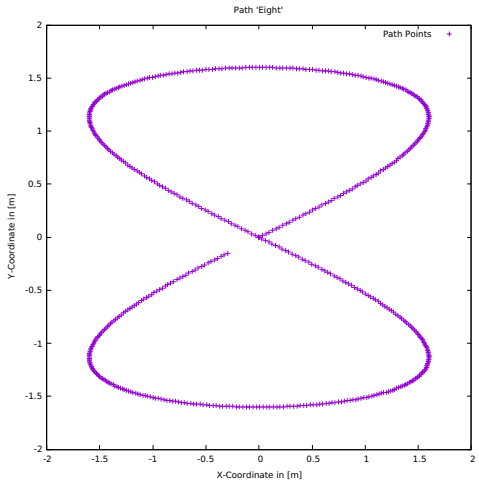


Figure 3.1: The "Eight"-Path shown with its points. The start point is located at $P(0/0)$ then the track follows to the top-right corner.

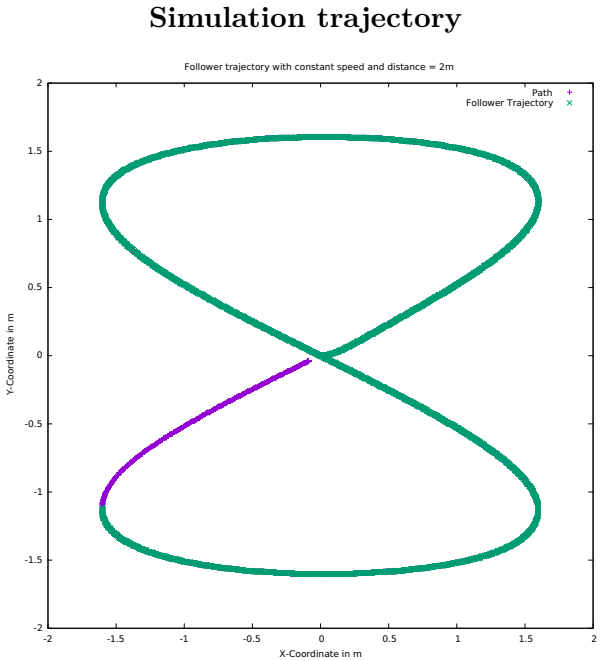
3.2.2 Experiment Distance Follower

For this experiment, the leader follows the path shown in 3.1. It is a symmetric path with two left curves and two right curves. The robot begins in the center at $P(0/0)$, and then drives to the top-right corner.

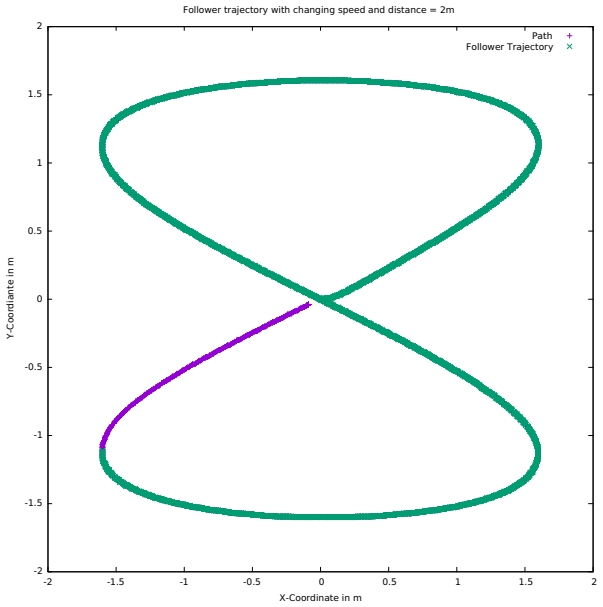
This experiment consists of the following stages.

- Simulation
  - Constant speed
  - Multistep speed controller with hysteresis
- Experiment
  - Constant speed
  - Multistep speed controller with hysteresis

At first, each controller simulates its functionality. Therefore, the odometry is used as truth data which is recorded by lifting the volksbot into air. Thus, the simulation is not only based on software, but also a simplified model of a natural environment without dead reckoning. In comparison, AMCL is used in the experiment stage in order to improve the performance. But still the position is assumed to be perfect in this scenario, as no external positioning system is used.

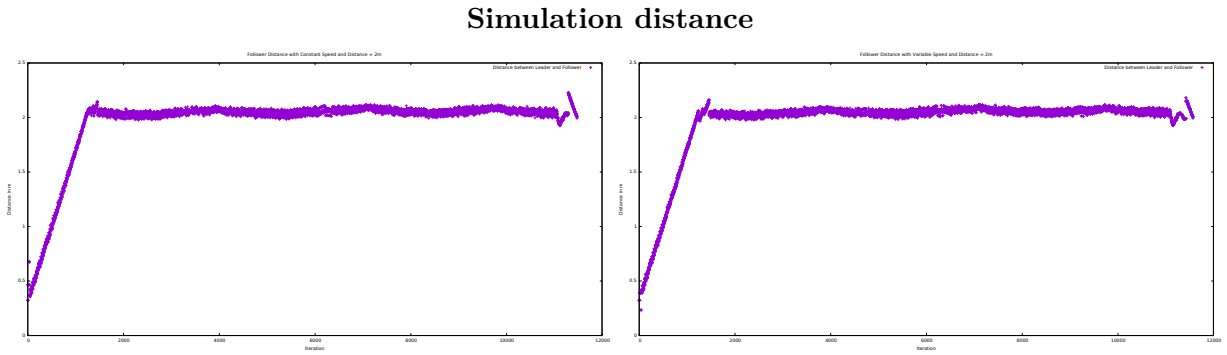


(a) Constant speed



(b) Variable speed

Figure 3.2: In this graphic, the trajectory of the robots is displayed in green, and the planned path in purple. With dead reckoning as the perfect position, both controllers follow the path perfectly. Furthermore, there is no observable difference in the end position.



(a) Constant speed **(b)** Variable speed

Figure 3.3: The distance d 3.4 over the duration of the simulation is shown here. The starting condition is that both robots start at $P(0|0)$. They fulfill the task with a precision of 0.1m.

Simulation

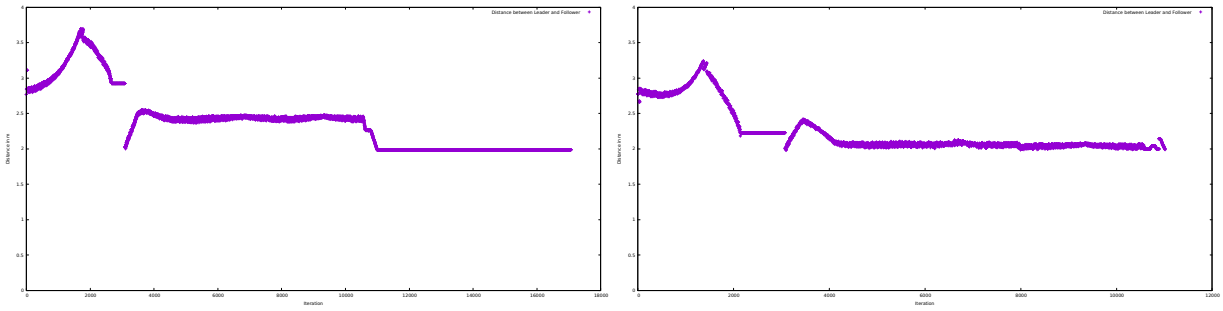
As it can be seen in 3.2 and 3.3, the controller is able to hold its distance to the leader. Both graphics look similar as the speed controller is not required when the starting distance is smaller than d_{target}. In this case, they perform on the same level. Figure 3.2 shows that the controller stops the robot at a certain distance as demanded by the task. The error calculated by using the formula (3.5). In the case of the constant controller, the value is 0.137m and for the speed controller it is 0.133m [2]. Thus, another simulation run is done with a starting distance d_s: $d_s > d_{target}$.

Starting distance bigger than target distance

This simulation run is conducted in order to show the problems with a constant speed controller. The simulation is repeated twice, one time without, the other time with the speed controller. The ros-tool "rosbag" guarantees similar starting conditions for both of them. A rosbag is recorded listening to the topic "Vel", which contains every message containing movement data for the robot. Thus, the rosbag can be played at the beginning, and then the robot repeats the commands and drives to the same location. In figure 3.4, the results of these runs are depicted. The comparison of the two graphs shows that 3.4b is able to catch up and keep the distance $d_{target} = 2$m, whereas 3.4a is only able to get at a range to the leader of 2.5m. At that range, the controller remains until the leader has finished its track. As a result, the constant controller has an offset.

[2] For this calculation, the step respond is not excluded as both controllers start under equal conditions

Followers' distance to leader with $d_{start} > d_{target}$



(a) Constant speed **(b)** Variable speed

Figure 3.4: Comparison of the two different controllers. 3.4a is here shown with about 17,000 cycles, whereas 3.4b only has 11,000. So that the change of function level in 3.4a takes place at the same interval as 3.4b ends.

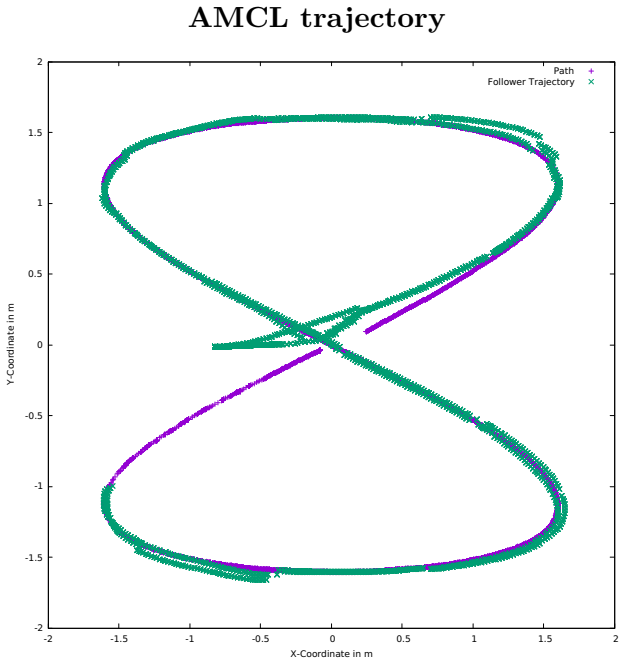
AMCL

In this experiment, the dead reckoning trajectory is corrected by the AMCL algorithm described in chapter 2, thus the trajectory is not looking as clean as in the simulation. It is important to point out that the position is still not measured by a global positioning system. Hence, the distance shown in the graphics can differ from the true distance and the quality of the controller can not be determined.

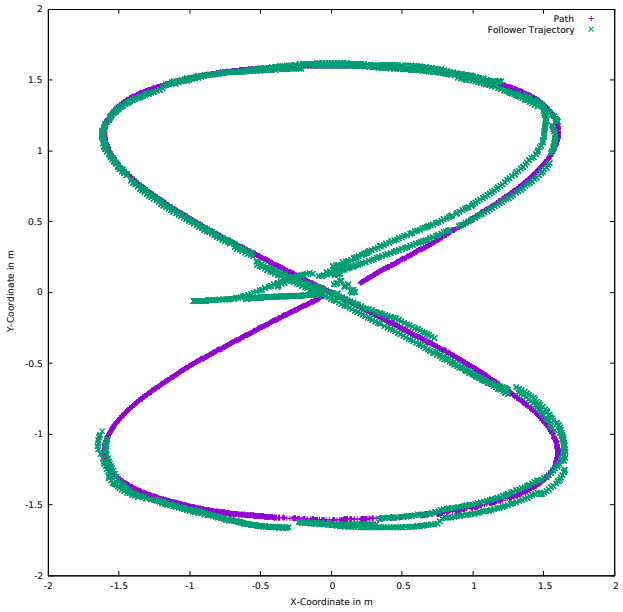
In 3.5 and 3.6, the repeated experiment using AMCL is shown. The robot's confusion about its position is observable. This could be an interference with the leading robot, which was within range of the laser scanner. Additionally, the beginning's viewing direction contains some points representing obstacles which are table legs. The combination of the assumptions described before can be a reason for the present figure 3.5. Also the trajectory of 3.5a seems to be better than 3.5b, as it oscillates less. The error reflects the same: 3.5a is 0.074m and 3.5b is 0.098m. Here the difference between both of the values is more than 10 times bigger as in the simulation.

Field of Application

In a practical sense, this formation is closely related to the safety driving distance application [12]; nowadays, it is used in cars and transporters as a security feature. The main difference between the devices such as the ones presented in [12] and the controller in this work is the source of information and the speed adjustments. Usually, the distance is gathered by sensors, as no communication between the vehicles exists. In order to improve the comfort, the speed is usually reduced smoothly in order to avoid an accident. In this work, the car stops when the distance to the front car is smaller than d_{target}, which, in the application, is the safety driving distance.

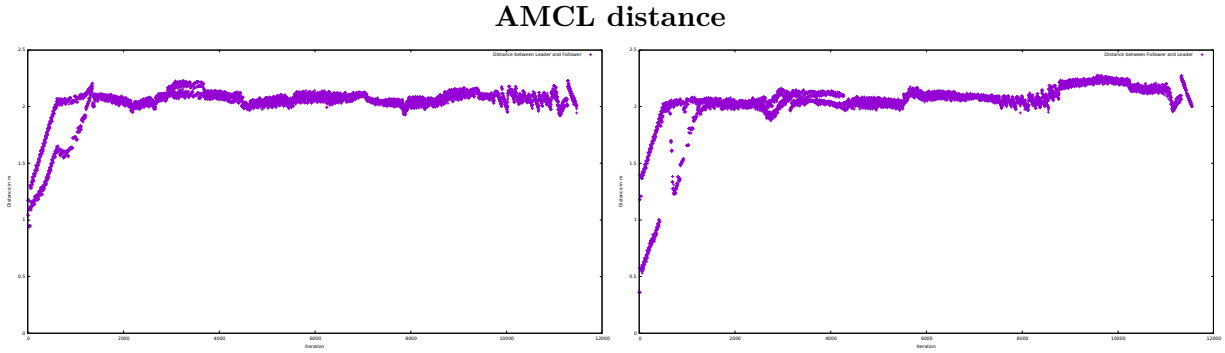


(a) Constant speed



(b) Variable speed

Figure 3.5: Here the trajectory, corrected by AMCL, is shown in green and the path is displayed in purple. In both experimental procedures, the starting points differ from $P(0/0)$. But after correcting their positions, they are able to follow the path. Confirming the previous simulations 3.2, they end at the same end position.



(a) Constant speed **(b)** Variable speed

Figure 3.6: The distance d (3.4) over the duration of the simulation is shown here. At the beginning, the same confusion about the position as in 3.5 can be seen, resulting in two different distances.

3.3 Task Offset Vector

3.3.1 Design Vector Offset Controller

Task Definition

The follower robot shall keep a specific vector $\vec{O}(x_o/y_o)$ between itself and the leader. Thereby, it is allowed to adjust the path and its speed. Due to the vector assignment, every local formation based on this principle consists of a single leader and a single follower instead of three ones, as in the standard string formation. Thus, the follower controller can be changed to only being able to communicate with a single leader.

Path Adjustment

In order to build the follower path, only local knowledge is available, which does not include future points. Thus, for each received path point message, except the first one, the following algorithm is used for calculating the new point. An example situation is depicted in 3.7.

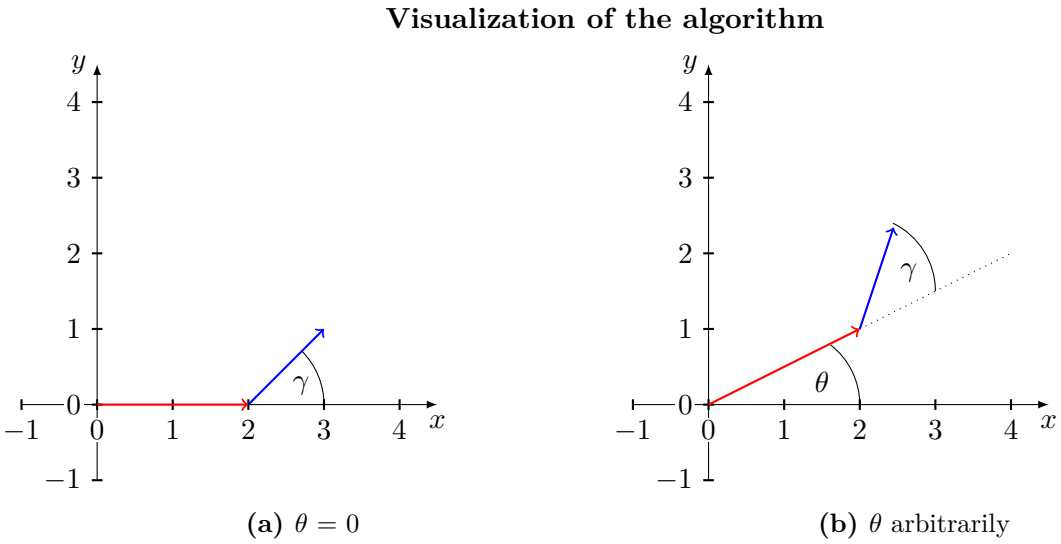
1. A path point message $P_c(x_c/y_c)$ is received and saved.
2. Difference between P_p and P_c is calculated with P_p being the previous received message.
3. Result from preceding step is normed.
4. Orientation θ of the vector is calculated using equation (3.6).
5. Offset $\vec{O}$ is rotated by θ resulting in $\vec{O}_r$.
6. Follower point is the sum of P_c and $\vec{O}_r$.

$$\theta = atan2(y_c, x_c) \tag{3.6}$$

For the path shown in 3.1, the algorithm leads to figure 3.8 with different offset vectors. When the offset is bigger than the curve radius, an anomaly from the expected path occurs. A partner point is found to each existing point in the original path, whose distance to the partner is exactly the $\vec{O}$. Hence, the emerging path represents the correct one to fulfil the task. The anomalies can be fixed by letting the follower stop and wait, instead of crossing the leader's path. If so, a starting condition is to know the path a priori; this condition fails to meet the task's requirements. Furthermore, the offset vector is not being kept as the leader does not turn on spot. So the anomalies are taken into account as a necessary feature.

Speed Controller

An early observation reveals that the follower is not able to keep the distance on the calculated path, in case it is limited to constant speed. For example, in the outer curve at the bottom of the graphic 3.8c, the follower has to cover more distance than the leader. A solution of the problem requires that the path follower is extended by a speed controller (see algorithm 2). In order to



(a) $\theta = 0$ **(b)** θ arbitrarily

Figure 3.7: The left figure shows a scenario in which no rotation is performed as it is the base coordinate frame, thus $\vec{O}$ (in blue) remains the same. On the other side a rotation of $\vec{O}$ with θ is performed. That way the angle γ between the path segment of the leader (red) and the offset $\vec{O}$ is kept constant.

avoid collisions between leader and follower, a safetyMargin can be set. If the distance between the vehicles is smaller than the margin, the follower will wait until the distance rises to the given limit. The check for distanceToPathEnd[3] enables the follower to reach its designated position by giving the follower a speed boost. The rest of the algorithm is equal to the previously used speed controller with hysteresis in 3.2.1.

Algorithm 2 The speed controller is designed as automate checking specific distances and thus adjusting the speed of the robot.

if distanceToLeader < safetyMargin **then**
  velocity = 0
else if distanceToPathEnd >headMargin **then**
  velocity = baseVelocity + hugeBonus
else if distanceToLeader >d_{target} + leaderMargin **then**
  velocity = baseVelocity - bonus
else if distanceToLeader <= d_{target} + leaderMargin **and** distanceToLeader >= d_{target} - leaderMargin **then**
  velocity = baseVelocity
else if distanceToLeader < d_{target} - leaderMargin **then**
  velocity = baseVelocity + bonus
else
  velocity = 0
end if

[3]It is calculated using (3.4)

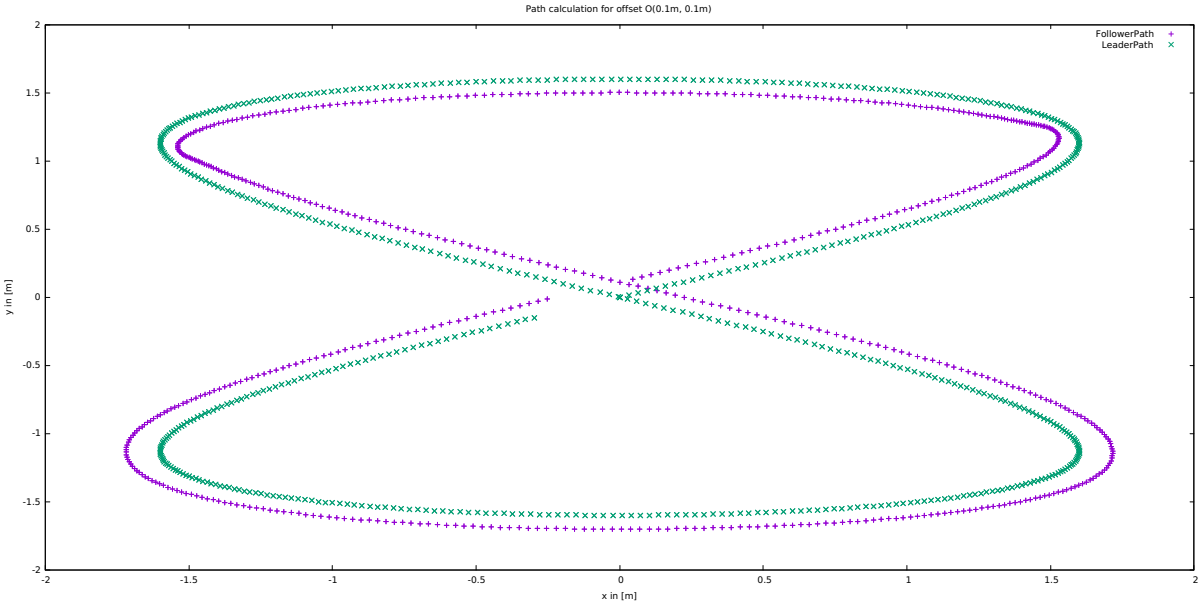
Further Improvements

The safety check is not sufficient enough to avoid collisions. The worst case scenario is that the follower is stopping on the leader's track, and waiting for it to pass. Then the leader would crash into the follower as the first one is not able to detect the follower and the latter is commanded to wait although it causes exactly the problem, which it is supposed to prevent. The first part can be solved in various ways, e.g. if the leader is equipped with obstacle avoidance, it then simply changes its track to drive around its follower. A downside of this approach is that the follower mimics the same move, which in turn can cause an equal situation, and ends in an infinite loop. Altogether, the problem is complex enough to be the subject of a continuing work. For this specific reason, it is not solved in this work. The AMCL experiment is evaluated a priori with the simulation, so that the follower does not collide with the leader. Simulations before the addition of the security result have shown that the minimum distance between leader and follower reaches 0.1m, which is less than the size of the volksbot robots.

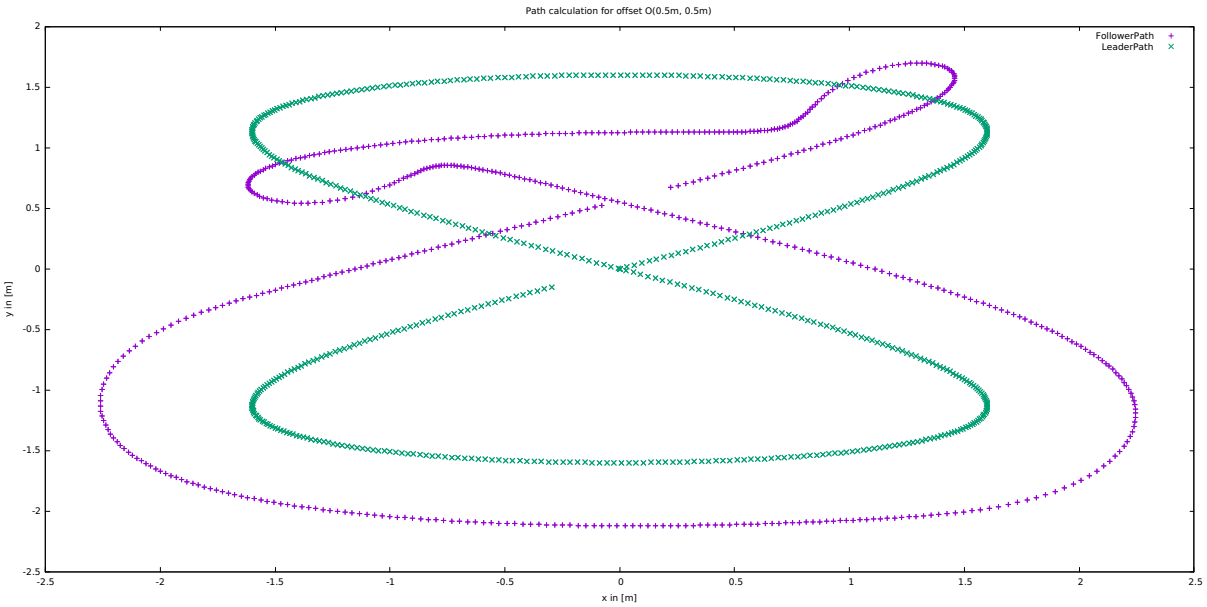
The path for the leader has to be restricted as well in terms of close curves. 3.10 shows, that the distance of about $\sqrt{2}$m $\approx$ 1.41m can not be kept in the outer curve. This is based on the fact that the outer distance increases by factor 2π. The problem can be covered with a higher speed bonus, but at some point the limitation of the mechanics prohibits it. Thus, based on the limitations of the vehicles and the performed offset distance, it is reasonable to restrict the curve radius. The paths, used in this work, are practicable enough to not require the implementation of such restrictions. But in the case of on-board path planning it is necessary.

Furthermore, in the simulation some point calculation errors where detected. A filter can be added in order to sort them out. In this work, the distance of the last points is therefore compared to the distance of the newest point to its predecessor. In case that this distance is bigger than a certain multiply, the calculation is assumed to be wrong and hence the point is ignored. The drawbacks of this approach were seen when the communication broke down for a short period of time, and afterwards any further point is ignored, as the gap created by the lost points exceeds the margin. In fact, this cancels the formation and thus has to be avoided. The "Eight" - Path 3.1 for example has a significantly smaller step size in the curves than on the straights, so that finding a good margin is heavily reliant on the path. A better approach is to compare the most recent distance to a calculated global average of the previous path. Finding an even better strategy, which also prevents the described scenario, can be the subject of another work.

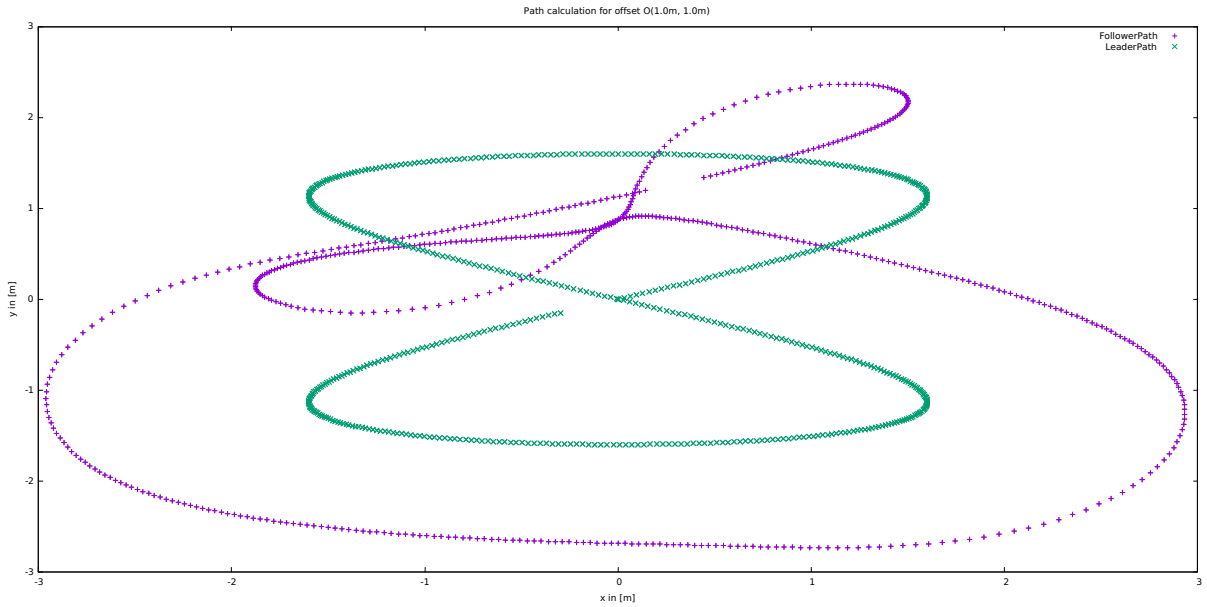
Calculated paths for different offset vectors



(a) $\vec{O}(0.1m/0.1m)$



(b) $\vec{O}(0.5m/0.5m)$



(c) $\vec{O}(1m/1m)$

Figure 3.8: In purple the resulting follower path, respectively to the leader path (green), is shown. 3.8a is as expected, whereas 3.8b has small anomalies and 3.8c has huge ones in the inner section.

3.3.2 Experiment Offset Vector Follower

All experiments use the "Eight-Path" (see figure 3.1), if not stated otherwise, as the offset vector $\vec{O}$(1m/1m) is used because it is safely applicable to simulation and AMCL.

Simulation

3.9 shows the trajectory of the robot. At the beginning, the path follower does not drive directly to the path. This problem is also dependent on the formation controller. With AMCL being the positioning system, the controller steers directly to the path 3.13. After the first loop, the rest of the track is followed successfully. 3.10 displays the distance during the experiment. The parallel lines, to the Iteration-Axis (Time-Axis), are the intervals to which the controller regulates the vehicle. Due to the nature of the path 3.1, it is impossible for the controller to keep that distance stable, especially in the outer curves.

The errors depicted in 3.11 show the precision of the vector offset controller. At this point, it is not optimized with its variables which are displayed in algorithm 2 as margins. SafetyMargin is 0.5m and distanceToPathEnd is set to 0.4m which is close to the minimums in 3.11a. As 3.11b is the error of the Giovanni Controller for the formation, an optimization for the variables is performed in 3.3.2. The parallel section in 3.11b shows the duration of the follower controller waiting for the leader in order to drive by and leave the safety margin. That period is equal to the global minimum indicated in 3.10.

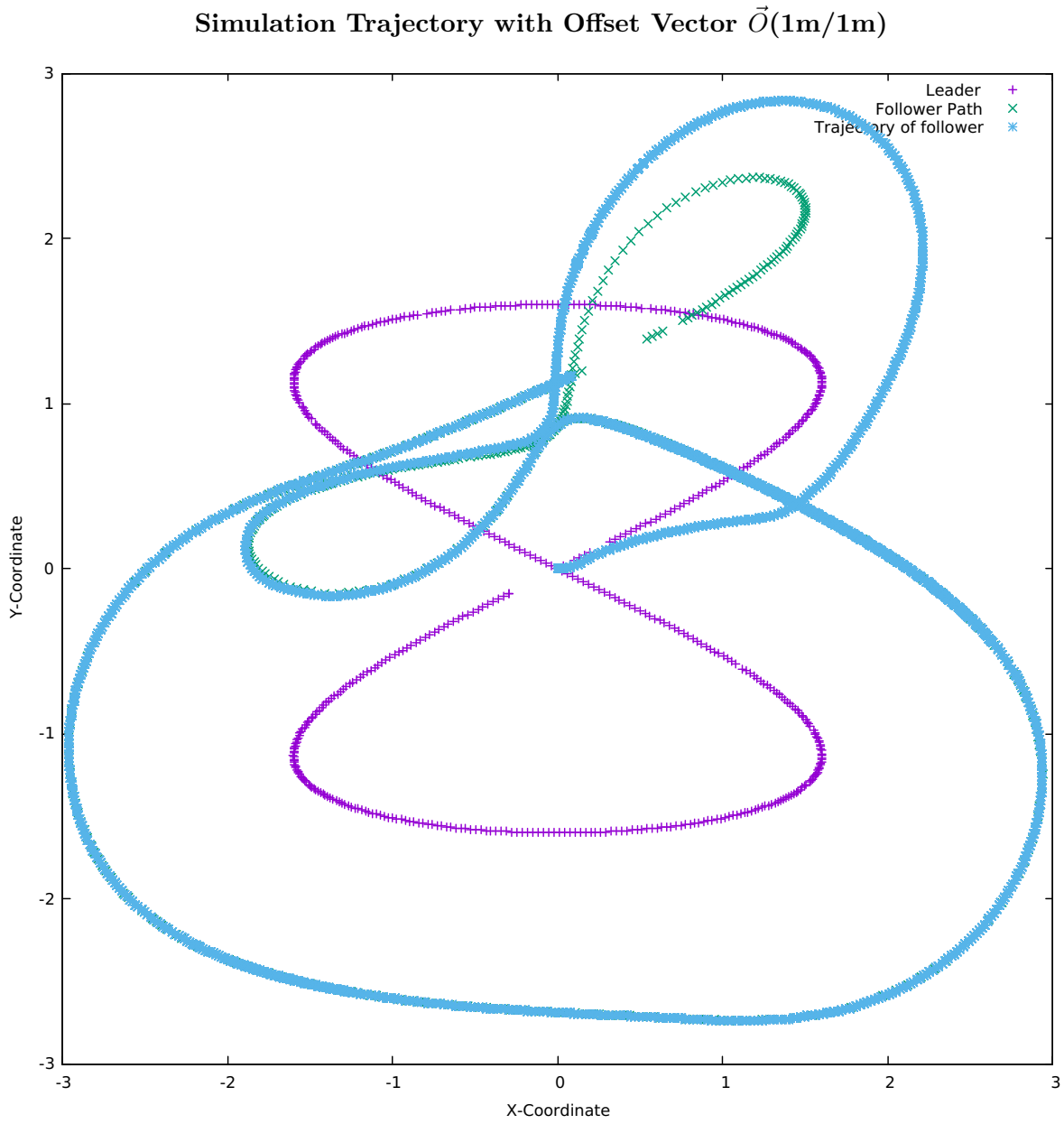


Figure 3.9: This graphic shows the trajectory of the follower (blue) resulting in the simulation by following the leader (purple) with an offset of $\vec{O}$(1m/1m).

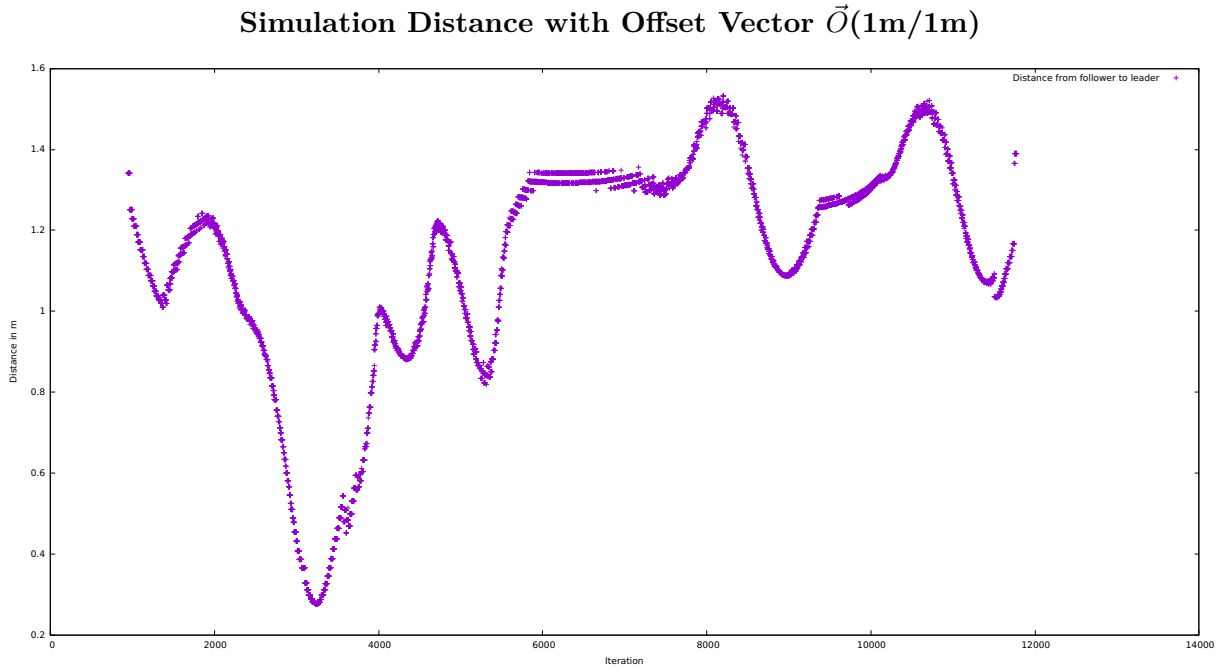
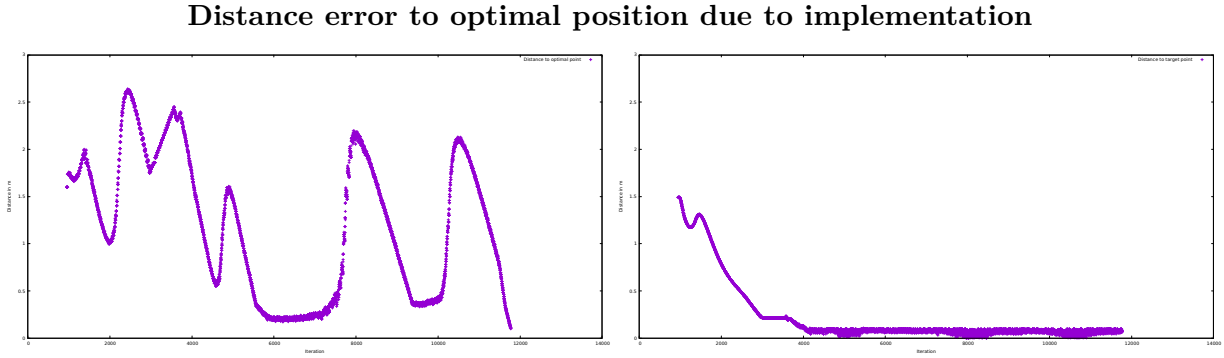


Figure 3.10: This figure shows the euclidean distance between follower and leader. The optimal distance should be around $\sqrt{2}$, due to the margin design of the controller it is not reached directly. But parallel straight sections show to which distance the robot is in a stable formation position.



(a) Distance to optimal formation point

(b) Distance to controller's target point

Figure 3.11: This graphics shows the implementation errors for the Formation-Controller in simulation with the same offset vector $\vec{O}$(1m/1m). 3.11a shows the euclidean distance to the position the robot should be and 3.11b is the error of the Giovanni Controller.

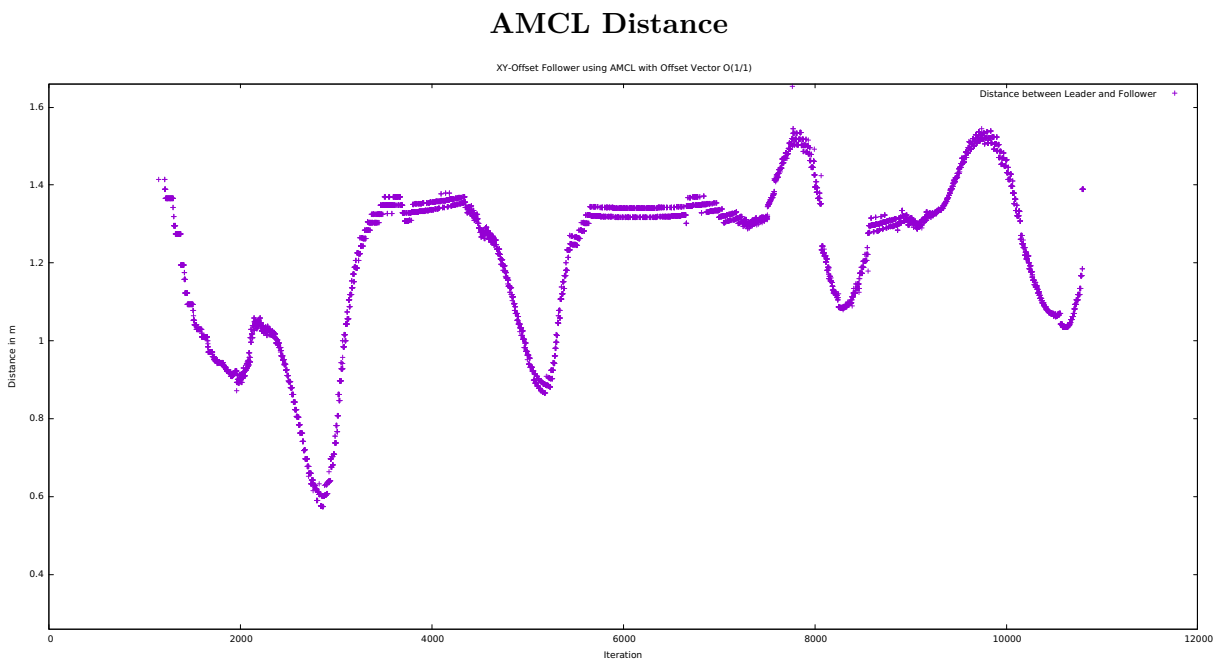


Figure 3.12: This graphic shows the distance between the leader and the follower using the AMCL positioning method. Thereby, the follower is in the right position mainly in two intervals: 4000-4500 and 6000-7000.

AMCL Experiments

In figure 3.12 the distance has stress peaks to the bottom and in the later stage, it consists of two sinus-shaped curves just as in 3.10. They originate from the 300°-curve, at which the follower first drives further away, then passes the leader and later returns to its position. At the minimum peak, the opposite happens: the follower waits for the leader to pass him, before driving back into its position. The other deviations from the follower's position occur in the curves, where it is physically impossible to maintain the formation. But intervals as 6000-7000 show that the follower is able to reduce drawbacks as the ones mentioned before over time, and during the same time the formation is active. The trajectory 3.13 shows the same properties as 3.5. In comparison to the simulation, the path controller steers earlier to the path, but does not follow it as cleanly as it does in the previous simulation.

For this experiment, no error values are calculated, as the evaluation can only be performed qualitative and not quantitative.

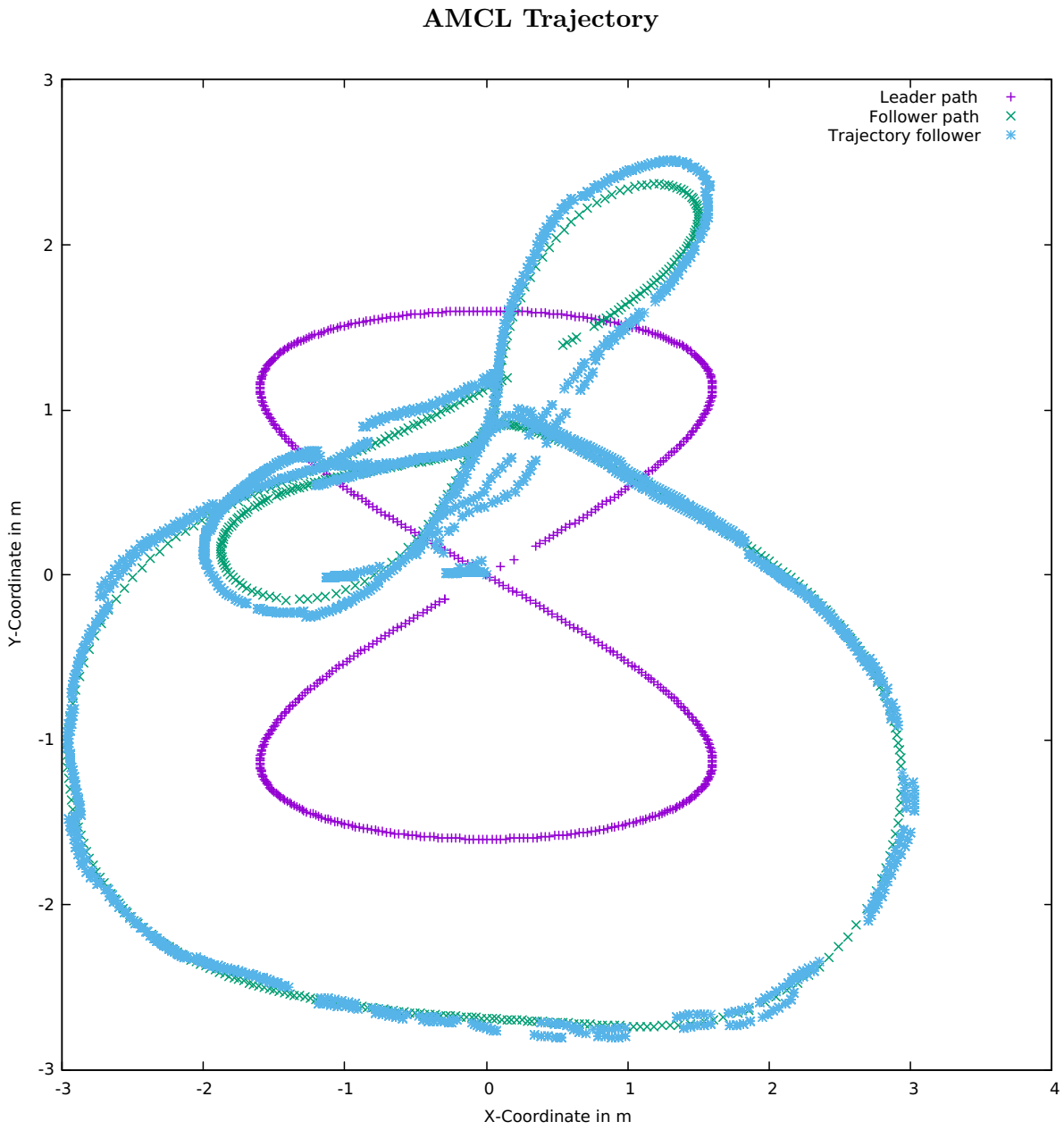


Figure 3.13: The purple curve is the leader path. The green curve depicts the optimal follower path, and the blue one is the trajectory measured by AMCL.

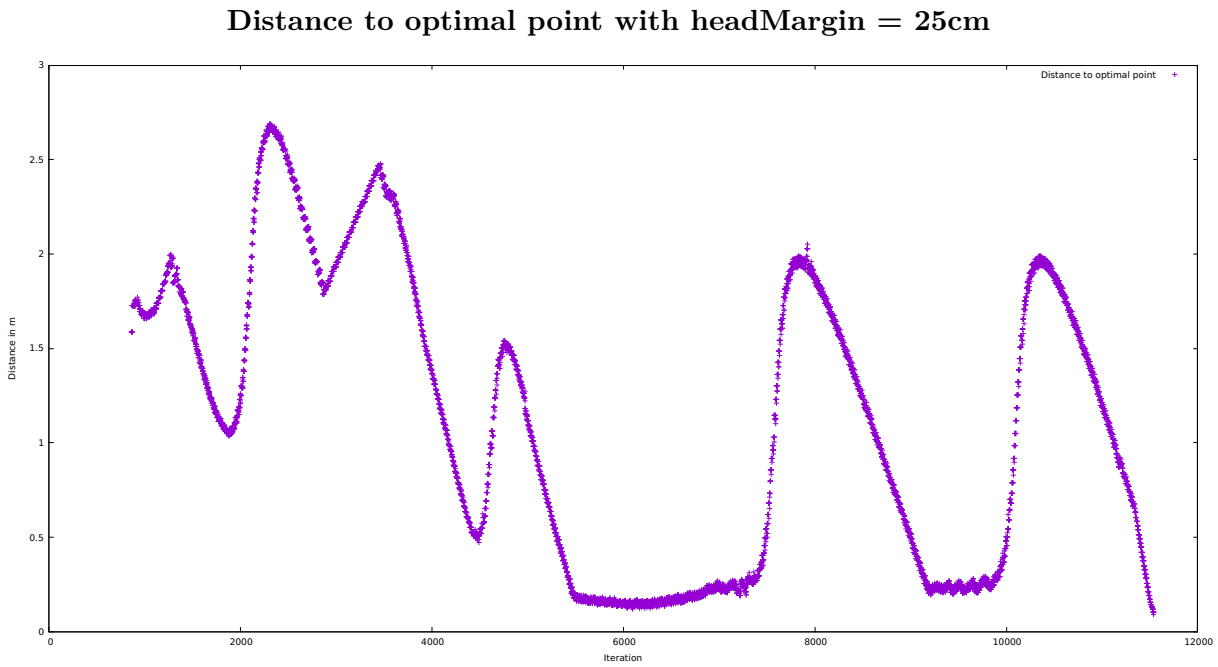


Figure 3.14

Variable Optimization

The variable headMargin is modified in order to measure the quality of algorithm 3.3.1. The value is mainly limited by the implementation of the used path library. Previous experiments are conducted with headMargin = 40cm. On the theoretical side, a smaller value for this variable will result in a faster convergence to the target vector, and thus increase the controller's performance. The experiment is repeated with the same setting as in 3.3.2. For each run, the value of headMargin is lowered (in 5 cm steps) until the experiment breaks. The smallest value for the headMargin is 25 cm. Figure 3.14 shows the distance to the optimal point. In comparison to the previous simulation results, the minimum is a bit smaller. Its effect on the performance can be seen in 3.15, in particular the interval size, in which the distance is stable at 1.4m, increases, and the distance between the curves converges to that value as well.

Field of Application

This type of formation can be used whenever several mobile frames need to be connected to a fixed frame. This can be seen in the following example: When an airplane exchanges fuel in the air with another one, or a harvester passes the harvest on to a transporter. Another application is to be found in automated industries for the transport of big items with several dolly-robots.

Distance between leader and follower with headMargin = 25cm

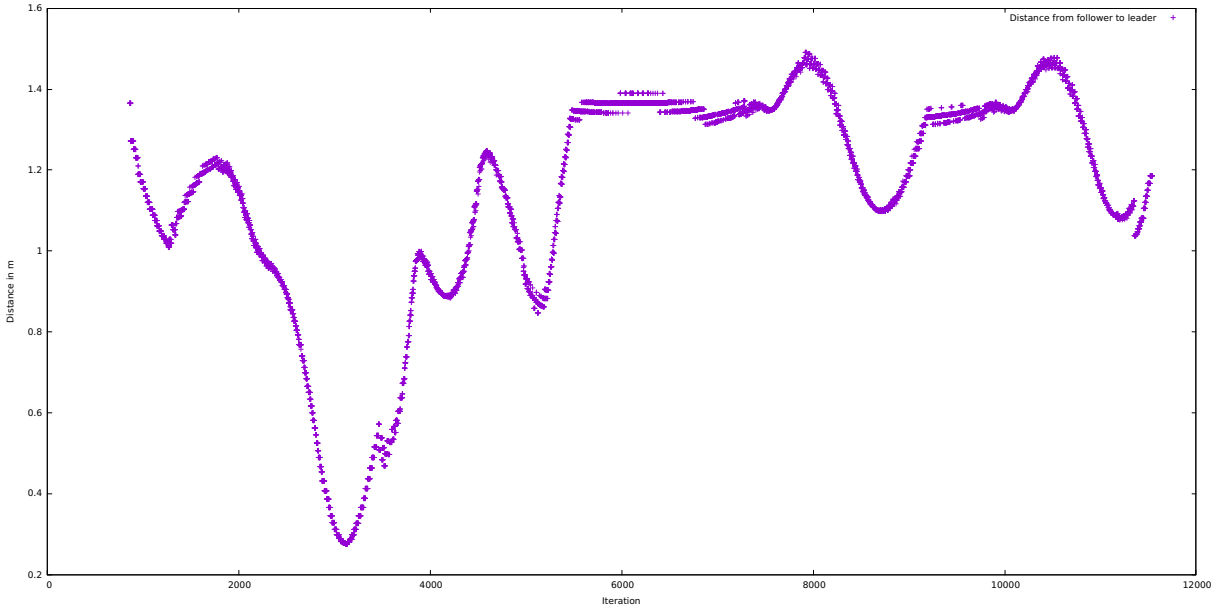


Figure 3.15

Chapter 4

Conclusion

The aim of this work was to show that simple methods are sufficient enough to create robust multi-system environments and complex systems. Only local information which can either be gathered through communication or sensors, is required to build them. With a modular structure, existing systems can be combined in order to build formations which are capable of performing complex tasks. In this work, the original Giovanni Controller's implementation is modified with a single interface, which enables the controller to set path points dynamically from external sources. Afterwards, the formation controller is able to meet decisions to maintain a formation by the use of a simple automate. Although it is not perfect, several locations are mentioned that can be modified with more complex algorithms, without requiring more information, just as the motto "No system is perfect" suggests. By adopting improvements to these algorithms, a better formation controller can be build. With the advancing automation in todays' society, formations will find its way into industries and our daily life.

Bibliography

[1] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, Mar 2013.

[2] Yang Quan Chen and Zhongmin Wang. Formation control: a review and a new consideration. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3181–3186, Aug 2005.

[3] William W. Cohen. Adaptive mapping and navigation by teams of simple robots. *Robotics and Autonomous Systems*, 18(4):411 – 434, 1996.

[4] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, Oct 2002.

[5] M. Egerstedt and Xiaoming Hu. Formation constrained multi-agent control. *IEEE Transactions on Robotics and Automation*, 17(6):947–951, Dec 2001.

[6] Guoyu Fu, Jin Zhang, Wenyuan Chen, Fengchao Peng, Pei Yang, and Chunlin Chen. Precise Localization of Mobile Robots via Odometry and Wireless Sensor Network. *International Journal of Advanced Robotic Systems*, 10(4):203, 2013.

[7] Hermann Helgert and Thomas Schmitt. Verwendung des Giovanni-Reglers zur Automatisierten Pfadverfolgung.

[8] Giovanni Indiveri. Kinematic Time-invariant Control of a 2D Nonholonomic Vehicle. 02 1970.

[9] Soheil Keshmiri and Shahram Payandeh. A Centralized Framework to Multi-robots Formation Control: Theory and Application, 01 2011.

[10] H. G. Tanner, G. J. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE Transactions on Robotics and Automation*, 20(3):443–455, June 2004.

[11] Barry Brian Werger. Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams. *Artificial Intelligence*, 110(2):293 – 320, 1999.

[12] Chi C Y. Automatic safety driving distance control device for a vehicle.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, March 2018