



Chair of Intelligent Systems

Software Project : Swarm Robotics and Swarm Intelligence

Flying in Formation

Documentation

Supervisor : Prof.Dr. Sanaz Mostaghim
Co-supervisor : Dipl.-Inform. Christoph Steup

Team Members : Andrei Stein
Stefan Georgiev
Marcus Kamieth
Nithish Hulikanthe Math
Vladyslav Buriakovskiy

Abstract

It is expensive to experiment with real quadcopters, as they can get damaged. Hence, simulations are used to test implemented behaviors of quadcopters before they are applied to real quadcopters.

In this project, a simulation has been created using the V-REP software environment, which is able to simulate swarm behavior. The foundation of the project is an already existing model of a quadcopter called Finken. The Finken model is based on a real quadcopter, which is used by other teams in different projects of the Chair of Intelligent Systems at OVGU.

Two solutions have been implemented, which are targeting two different approaches. In the first solution the proximity sensors of the Finken are used to detect other objects. Attraction and repulsion have been added to the Finken to create a swarm behavior. With this solution, multiple Finkens are able to fly in a 3D-environment and form a formation without crashing.

The second solution uses a plane with a gradient texture, which represents a landscape. The Finken tries to find the minimum value of the landscape. For this purpose, a combination of a random-walk algorithm and a vision sensor is used.

Based on the achieved results, other principles from the area of Swarm Intelligence can be simulated. This includes particle-swarm optimization as well as communication between quadcopters that have different communication radius.

List Of Figures

Figure 2.1: Quadcopter angular parameters	2
Figure 3.1: Swarm behavior	5
Figure 3.2: Proximity sensors of quadcopter.....	6
Figure 3.3: Floor detection by quadcopter.....	7
Figure 3.4: Wall detection by quadcopter	8
Figure 3.5: Quadcopter detection and attraction-repulsion to each other.....	10
Figure 4.1: 3D image of the landscape function.....	12
Figure 6.1: Block diagram of a PID controller	15
Figure 6.2: Representation of the motion equations.	17
Figure 6.3: Plot of the attraction and repulsion functions.....	18
Figure 6.4: Motion of the swarm	18
Figure 6.5: Pattern formation and tracking	19

Content

1	Introduction.....	1
2	Copter Control.....	2
2.1	Manual Control.....	2
2.2	Control using PID Controller	3
2.3	PID Components.....	4
2.4	Parameter Constraints	4
3	Swarm Behavior	5
3.1	Sensing Basics.....	5
3.2	Height Measuring.....	6
3.3	Floor Detection.....	7
3.4	Wall Detection.....	8
3.5	Attraction and Repulsion.....	9
4	Landscape Searching	11
4.1	Image Creation.....	11
4.2	Integration in V-REP	12
4.3	Different Approaches	12
5	Conclusion	14
	Appendix.....	15
	A.1: PID Controller [2] [3]	15
	A.2: Swarm Intelligence (SI) [4]	17
	References.....	20

1 Introduction

The objective of this simulation project is to create a swarm behavior with different formations based on the existing Finken model. Each quadcopter should have an approximate real behavior, the formations should be stable and controllable. The quadcopters should have a specific distance to each other and should not collide. Hence, the simulation should resemble real conditions.

For the swarm simulation, V-REP software has been used. V-REP is a powerful 3D robot simulator, which features several versatile calculation modules like inverse kinematics, physics/dynamics, collision detections, minimum distance calculations, path planning and many more to pen. It supports a distributed control architecture i.e, an unlimited number of threaded or non threaded control scripts and several extension mechanisms which include plug-ins and custom client application. [1] In this project, the Lua script language is used in the control scripts to realize the desired functionality.

The quadcopter model has been created in V-REP under the Chair of Intelligent Systems at OVGU. It resembles a real quadcopter in terms of propellers, static parts and sensors. Also the aerodynamics has been done by particle simulation. The model can be controlled by the four parameters throttle, pitch, roll and yaw. Further on, a PID-Controller has been implemented to control the Finken. With this control mechanism, the Finken is able to fly stably in the environment.

Based on the Finken model, further implementations like multiple quadcopter swarm behavior and landscape searching have been realized.

2 Copter Control

This chapter describes two approaches to control the Finken quadcopter. In the first approach, the quadcopter is directly controlled by varying the yaw, roll, pitch and throttle parameters. In the second approach, these parameters are varied by a PID-controller.

2.1 Manual Control

Like a real quadcopter, the Finken model has the following parameters to control the quadcopter (ref. Figure 2.1 and Table 1):

Table 1: Control parameters of the quadcopter

Throttle	propeller speed, usually height control
Pitch	forward or backward inclination angle
Roll	sideways inclination angle, either left or right
Yaw	rotation of the quadcopter either clockwise or anti-clockwise

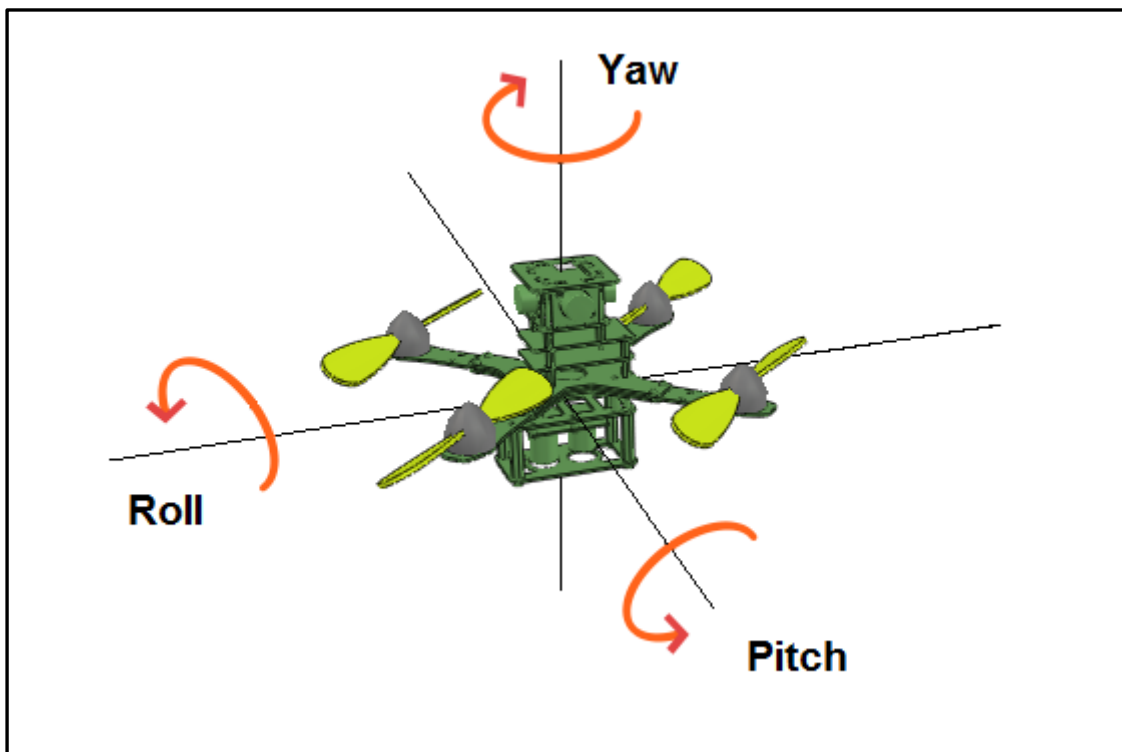


Figure 2.1: Quadcopter angular parameters

To evaluate the Finken model, simple movement patterns have been implemented. The different movement patterns are:

- 1) Landing behavior of the quadcopter
- 2) Swing down behavior, i.e. flight towards a target by first losing height and then gaining height again.
- 3) Swing up behavior, i.e. flight towards a target by first gaining and then losing the height.
- 4) Horizontal movement always keeping the same height.
- 5) Circular movement.

The implementation of the behaviors has been done by using either state-machines or mathematical functions. With the landing pattern, it is difficult to react to different initial heights due to the gravitation force. Big changes in roll or pitch angles lead to an unstable behavior of the quadcopter for the swing up and swing down movement patterns. The circular movement implementation uses sine and cosine functions for the control of pitch and roll. The required circular movement is achieved but because the throttle is constant, the height is not maintained. Moving the quadcopter horizontally on a straight line works good, but it is difficult to stop it at the target position due to the acceleration forces.

The experiments show that manual control does not yield the required results. Change in the angle value leads to a gradual change of the angle and thus there is no exact value at each timestep. Same for a parameter which shall be kept constant at an exact value, which is physically impossible. This is called steady-state-error. Therefore, it has been decided to control the quadcopter by a PID-controller, which is able to compensate such problems.

2.2 Control using PID Controller

Using only simple mathematical models to control a quadcopter does not work over time. There is the steady-state-error and inclinations, which do not follow a step function but are only performed gradually. The solution to this is to use a PID-controller which tries to minimize the error between setpoint and actual value. Further information about PID controllers can be found in Appendix A.1.

There are many approaches to do so. In one approach, cascaded P-controllers are used. In the final solution these have been replaced by a PD-controller.

In general, PD-controller works as follows: The quadcopter shall move towards a given target position. Thus, the distance from the quadcopter to the target is given by the euclidean distance between them. This distance is fed to the controller as the error which shall be minimized. The output is the desired parameter value and is applied to the quadcopter. For throttle, this is an acceleration, and for pitch and roll respectively, it is the desired angle. For calculating pitch and roll, only the distance in x-axis and y-axis between the quadcopter and target is needed. The height control is done using absolute height values. Because with relative coordinates the quadcopter's position would be the origin and practically, it's not possible to calculate the height. As a solution, a dedicated height sensor has been implemented to provide the distance to the floor as a sensor value.

Using a PD-controller, the quadcopter reaches its target after some time and stays there until the target is changed.

2.3 PID Components

The PID controller of the Finken only uses the proportional (P) and derivative (D) components. It does not use the integral (I) component because of frequently changing targets of the Finken. The integral component would calculate a cumulative error which is not appropriate and the Finken tries to compensate it and starts oscillating.

To find suitable parameters, following methods could be used:

1. Ziegler-Nichols Method
2. Chien-Hrones-Reswick Method

However, the P and D values have to be much smaller than proposed by these methods to guarantee a stable behavior. Therefore, the values for P and D have been obtained experimentally.

2.4 Parameter Constraints

The output parameters of the PID controller have to be limited to a certain value such that the Finken does not destabilize. Pitch and roll are limited to the interval $[-10^\circ, +10^\circ]$. On

the one hand, this ensures a stable flight and on the other hand, a relatively quick movement can be achieved. Throttle is limited to $[0,100]$, which represents the technical constraints for throttle.

3 Swarm Behavior

This chapter describes the implementation of a swarm simulation scene in V-Rep. The scene contains a group of quadcopters, which are able to detect other objects in their surrounding area by using sonar proximity sensors. Thereby, the quadcopters apply attraction and repulsion towards the detected objects to create a formation (ref. Figure 3.1). Single quadcopters without neighbors use a random-walk-behavior to explore the environment and find other quadcopters. The following paragraph explains the used principles in detail.

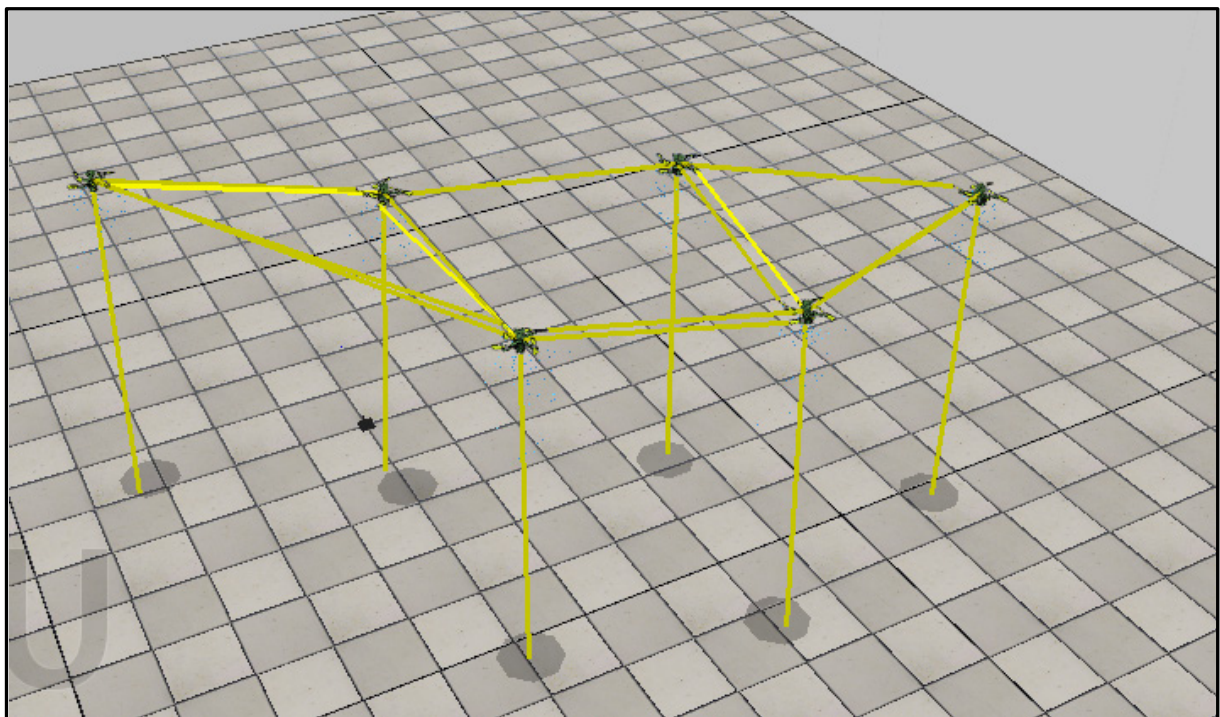


Figure 3.1: Swarm behavior

3.1 Sensing Basics

The quadcopter has to be able to react to the surrounding environment. Thus, it needs a sensing mechanism. The following paragraphs describe the sensing mechanism used in the quadcopter.

Each quadcopter has proximity sonar sensors pointing in different directions (front, back, left, right) referring to Figure 3.2. The sensors have the shape of a cone. The detection area has a range of 3 meters and an angle of 90° .

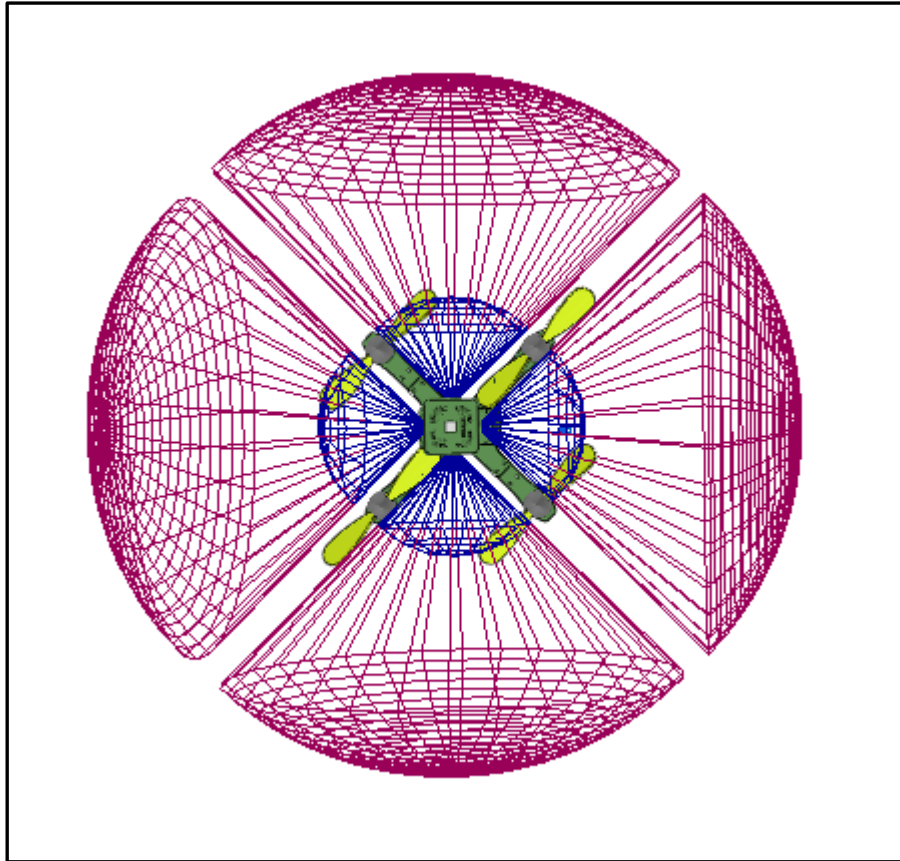


Figure 3.2: Proximity sensors of quadcopter

A sensor measures the distance to the nearest detected object. The detected object can be either a quadcopter, a wall or the floor. The exact direction of the detected object is not provided by the sensors. Only the approximate direction left, right, front or back is known depending on which sensor detects the object.

3.2 Height Measuring

The sensor at the bottom of the quadcopter has a beam shape and its purpose is to measure the distance to the ground. It is expected, that `bottomDistance` is perpendicular to the floor (ref. Figure 3.3). When pitch or roll are changed, the bottom sensor is not pointing perpendicularly to the floor. Theoretically, a real distance to the floor can be recalculated (based on similar approach as in the floor detection chapter, that uses tangent function, pitch and roll angles). This approach has been tested, but the behavior is not stable, because

sometimes the recalculated value is too big (without any logical reason). Therefore, a simplification is done. The distance to the floor is not recalculated, but it works because of the limitation of the roll and pitch angles with the range of $[-10^\circ, +10^\circ]$. In the future, the described problem should be investigated to increase the stability of the solution.

3.3 Floor Detection

A sensor at one of the four sides can have a long range. If a quadcopter is flying at a lower height, then it is possible that the floor is detected by the sensor (ref. Figure 3.3).

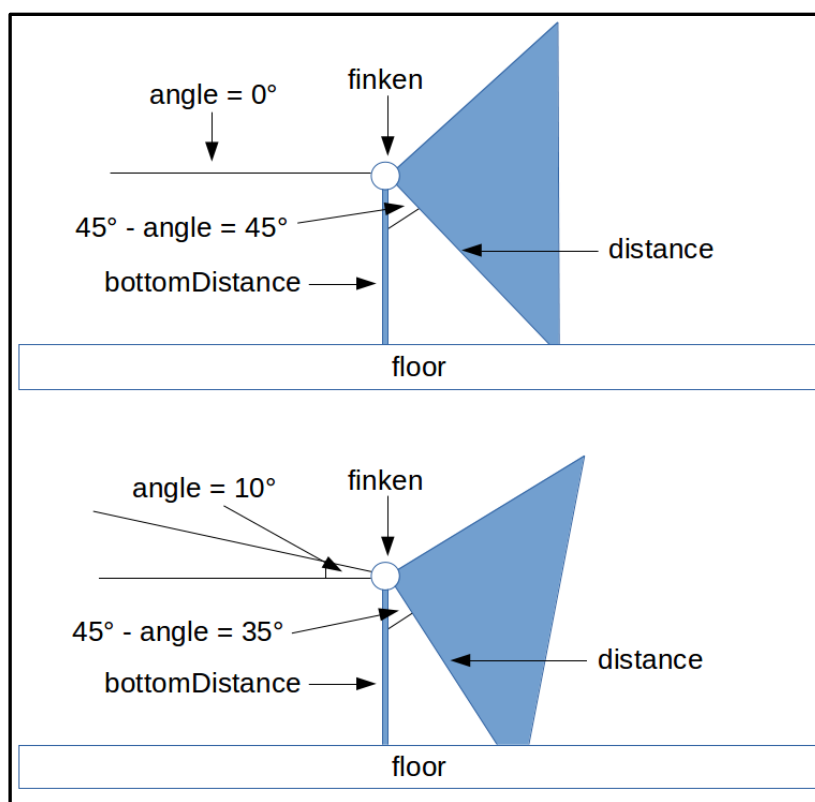


Figure 3.3: Floor detection by quadcopter

In this case, the quadcopter applies an attraction in the direction of the sensor. Thus, the quadcopter flies in that direction and that creates an unwanted behavior. To avoid this behavior, the implemented solution tries to identify this situation. Identified sensors, which detect the floor, are excluded from applying attraction and repulsion. The distance value of the sensor located at the side is analyzed. The *bottomDistance* is the distance to the floor. The tilt angle is based on pitch or roll of a quadcopter.

If the sensor detects the floor, then *bottomDistance*, distance of the sensor and the floor form a right-angled triangle. Based on the cosine function, a new possible distance is calculated and compared with the current distance of the sensor. If the difference between them is in a given range, then the floor is detected and the sensor is not taken into account for applying attraction and repulsion.

3.4 Wall Detection

The simulation is executed in a virtual room with walls. All quadcopters are moving. Therefore, it is possible, that a quadcopter is near a wall and its sensors are detecting it. All quadcopters are trying to keep a distance to all detected objects around them. If three of the sensors detect the same wall, the quadcopter gets unstable. The attraction and repulsion of each sensor tries to achieve the same distance for all three sensors. In this situation, this is not possible (ref. Figure 3.4).

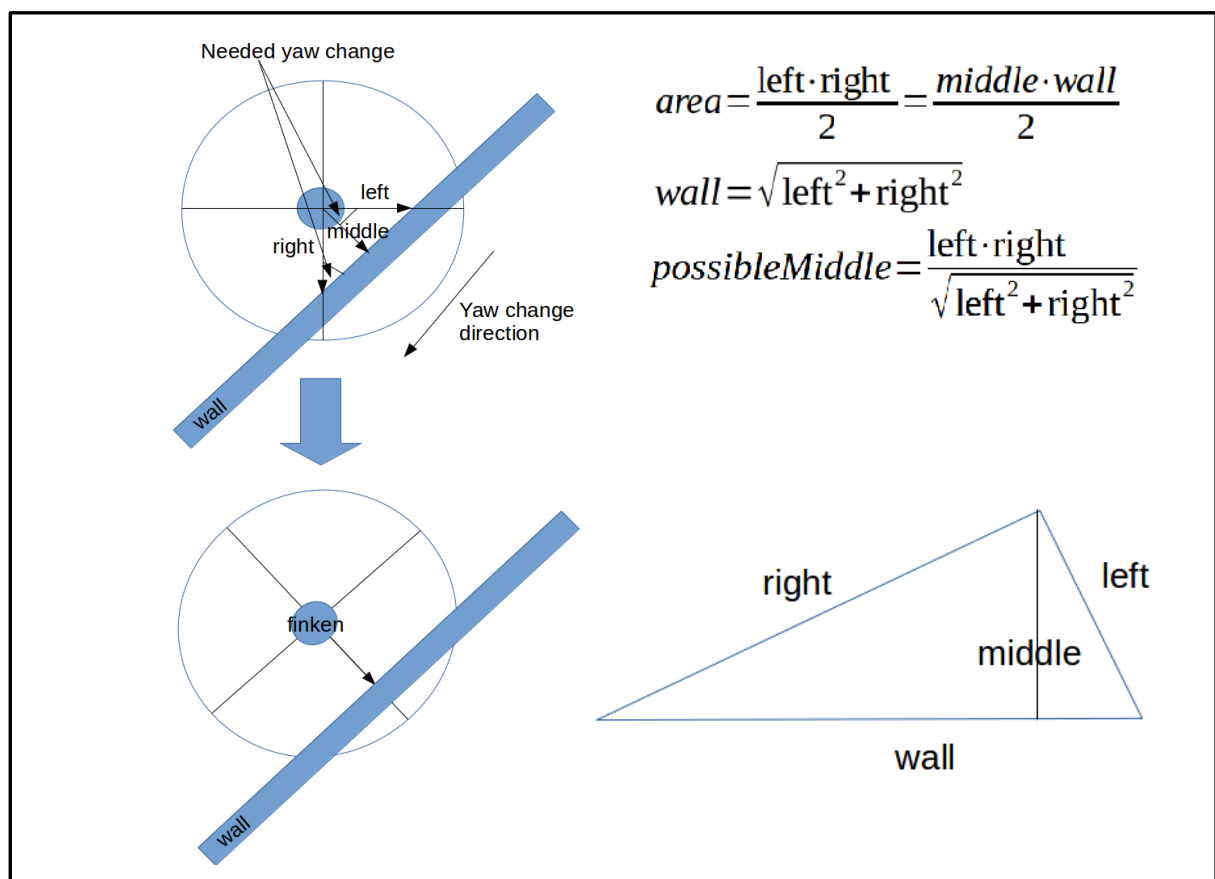


Figure 3.4: Wall detection by quadcopter

The implemented solution detects this situation and reacts to it. *middle* is the distance from the analyzed sensor, *left* and *right* are distances from the neighbor sensors. *left*, *right* and

the wall form a right angled triangle. *middle* is perpendicular towards the wall in this triangle. Based on the formulas for area of a triangle and the Pythagoras theorem, a *possibleMiddle* is calculated and compared with *middle*. If the difference between them is in a given range, then the wall is detected. An iteration-counter counts the amount of sequential simulation steps where a wall is detected for the analyzed sensor. If it reaches a given value, then the yaw angle is changed. The needed change value is determined by a function, that uses the tangent function between the two distances, left and right. After the yaw is changed, only two sensors are pointing to the wall and their detected distances have similar values.

3.5 Attraction and Repulsion

To achieve a swarm of quadcopters, a single quadcopter has to react to its neighborhood accordingly. Attraction and repulsion are used for that purpose. It is a commonly known principle from the area of Swarm Intelligence which describes the behavior of two individuals of a swarm. Appendix A.2 describes this principle in detail.

If a quadcopter detects some object in its neighborhood, attraction and repulsion is added in the approximate direction of the object. The attraction forces the quadcopter to fly towards the detected object. The repulsion prevents the quadcopter from crashing with that object. Attraction and repulsion affect each other, so the quadcopter converges at a specific distance to the object.

Attraction and repulsion can be applied to the quadcopter for each of the four directions (left, right, front, back). Each sensor just detects the nearest object, so the quadcopter is able to react to a maximum of four objects in its neighborhood. Once two or more quadcopters are attracted, the attraction and repulsion causes them to stay in a swarm (ref. Figure 3.5).

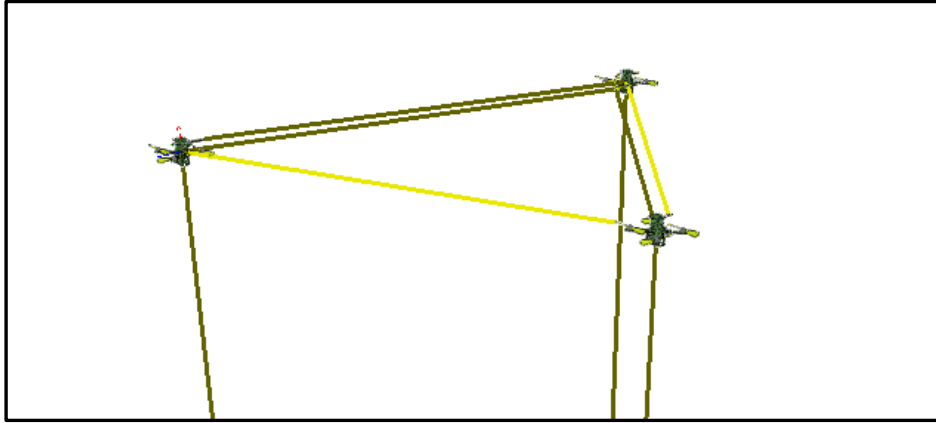


Figure 3.5: Quadcopter detection and attraction-repulsion to each other

The achieved results are satisfying. The quadcopters are able to fly in an environment. Thereby, they are creating different swarm formations without crashing into each other. The swarm convergence can be evaluated with an implemented error-measurement (sum of squared error, SSE). It measures the squared error between the desired distances caused by attraction and repulsion and the actual distances measured by the sensors.

Even though the simulation uses a model based on a real quadcopter, it does not take all aspects from the real world into account. As an instance, the sensors behave not exactly as the real sensors. Real sensors do not always measure exact values. This can be a task for future work. Another task is to find a better solution for sensing the height of the quadcopter as described in the chapter Height Measuring. The presented solution can be seen as a foundation to implement and simulate other topics from the area of Swarm Intelligence.

4 Landscape Searching

Another task is to make the Finken quadcopters follow the gradient of a landscape. On the one hand, this technique can be used to implement a particle swarm optimization scenario and on the other hand, it is possible to control the quadcopters via signals from outside.

In the simulation, the landscape is visualized by a texture on the floor, whereas in the SwarmLab, for real quadcopters the landscape could be projected from the ceiling. Here, the gradient of a landscape means that the quadcopter follows the lightness of the color and thus, simply a grayscale picture is applied. It is required to have a landscape where the gradient in the neighborhood of the minimum is towards this minimum, like a bowl. Thus, the quadcopter is able to stick to that minimum. Otherwise, the quadcopter could leave the goal area and is not expected to return.

4.1 Image Creation

It is possible to create a landscape picture in many ways. One approach is to use a paint program. Here, a line is drawn and afterwards a Gaussian blur is applied to it. This approach is very simple, but a more complicated canyon with only one optimal point can not be drawn that easily.

Thus, in the second approach mathematical formulas and a JAVA program to translate the respective function are used to create a grayscale bitmap. With this approach, any formula could be used and for each differentiable function the gradient is known, at least in which direction the quadcopter should fly. The JAVA-program itself is not complicated, first a new buffered image has to be created, then the color values for each pixel are set and afterwards the image has to be written to a file. Thus, the main part is to design a mathematical formula and fill the bit array with RGB-values between 0 and 255.

One formula has been designed that is a Gaussian curve on the x-axis and a linear function on the y-axis from 25% to 100% lightness. As the image has a resolution of 1024x1024, the formula reads as follows:

$$L(x, y) = a * e^{\frac{(x-b)^2}{2c^2}} = \left(\frac{191}{1024} y + 64 \right) e^{-\frac{(x-512)^2}{40000}} \quad (4.a)$$

Referring to the landscape function in (4.a), the peak is given by “a”, which is a linear function in y . The mean is given by “b”, and the standard deviation corresponds to “c”. Figure 4.1 depicts the graph of this function.

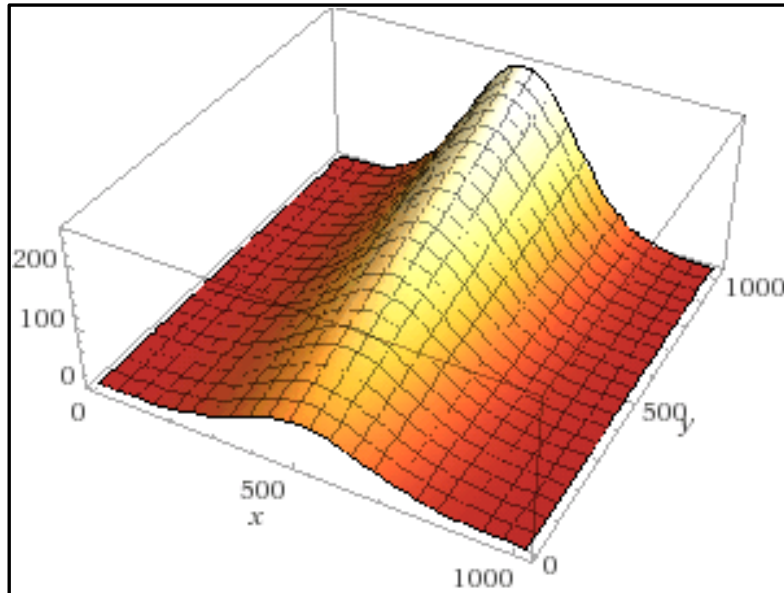


Figure 4.1: 3D image of the landscape function

4.2 Integration in V-REP

For this, a new plane has been created to replace the default tiled floor. The image is applied to the floor with a scaling factor of 1024, same as the image size. Then the image has to be scaled along U and V to match the actual floor size, e.g. 5. The ambient color of the floor has to be changed to white to show the real color of the texture.

4.3 Different Approaches

First of all, the quadcopter always moves randomly. This is achieved by a random walk algorithm. The defined goal color is white i.e., the quadcopter tries to move from black to white or from the dark to the light color, respectively. For consecutive simulation steps, the color gradient is calculated. If the gradient is negative, i.e. dark to light, the quadcopter sticks to its current direction, otherwise a random direction is assigned to the quadcopter.

The color sensing in all approaches is done by a vision sensor pointing to the floor. It measures the lightness with a sensor resolution of 1px. Thus, black results in 1 and 0 in white. This does not correspond to lightness but is chosen because white represents the minimum to be reached.

There are three different approaches to attach the sensor to the quadcopter. In the first approach the sensor is attached fixed to the quadcopter. Thus, the sensor tilts with the quadcopter resulting in a large color-position-error relative to the height of the quadcopter. To compensate this changes in color, the gradients of pitch and roll have to be considered. This means if the quadcopter accelerates in direction of the white color, it is still the right direction despite the measured color of the sensor gets darker.

As a second approach, a gimbal sensor has been implemented, i.e. a sensor that is independent of the quadcopter's rotation. For this sensor, the simulation of vision sensor rotation has to be executed more often than the quadcopter control loop. The problem with this approach is that there is still some delay in movement which also results in wrong color measurements as in the first approach.

In the third approach, a color sensor on top of the quadcopter is simulated. Hence, an additional shape is added to the top of the quadcopter which is only used as a reference point. The vision sensor is moved according to this reference point on top of the quadcopter and is not rotated. Since the sensor shall not rotate like the Finken quadcopter does, it has to be located outside of the Finken object hierarchy. This approach may even be realized in the SwarmLab where the projector projects the landscape image to the floor. The performance of the quadcopter depends on a random walk and thus, the quadcopter not always finds its way to the white color.

This performance could be enhanced by letting the quadcopter fly in a larger area, such that the randomness does not have such great influence on the performance. With smaller P and D values or smaller pitch and roll constraints, the quadcopter would move more slowly yielding the same effect. Further, it would be adequate to not have the minimum landscape value at the edge of the texture, such that the quadcopter is not that likely to leave the texture.

If the quality of this random walk is not sufficient at all, the resolution of the vision sensor could be increased, e.g. to 3x3 Pixels. Thus, the gradient could simply be calculated. For this solution, the quadcopter would only move to the nearest (local) minimum.

5 Conclusion

Using the theory of Swarm Intelligence, swarm behavior of Finken quadcopters has been simulated and the achieved results are very satisfying. The swarm behavior of multiple quadcopters can be witnessed. The purpose of this project, to experiment with the quadcopters' control and behavior in the simulation environment has been achieved. The results can be analyzed and adapted to the real environment.

For further research, physical parameters such as noise can be added to see how robust this swarm behavior is against imperfections. The landscape searching can be extended to support multiple quadcopters, such that the landscape helps the Finken quadcopters to form a swarm or let the swarm do particle swarm optimization.

Appendix

A.1: PID Controller [2] [3]

Proportional-Integral-Derivative control is a control algorithm widely used in industrial applications e.g, to control temperature, pressure, flow rate or velocity. It has a robust performance under different operational conditions and a simple functionality.

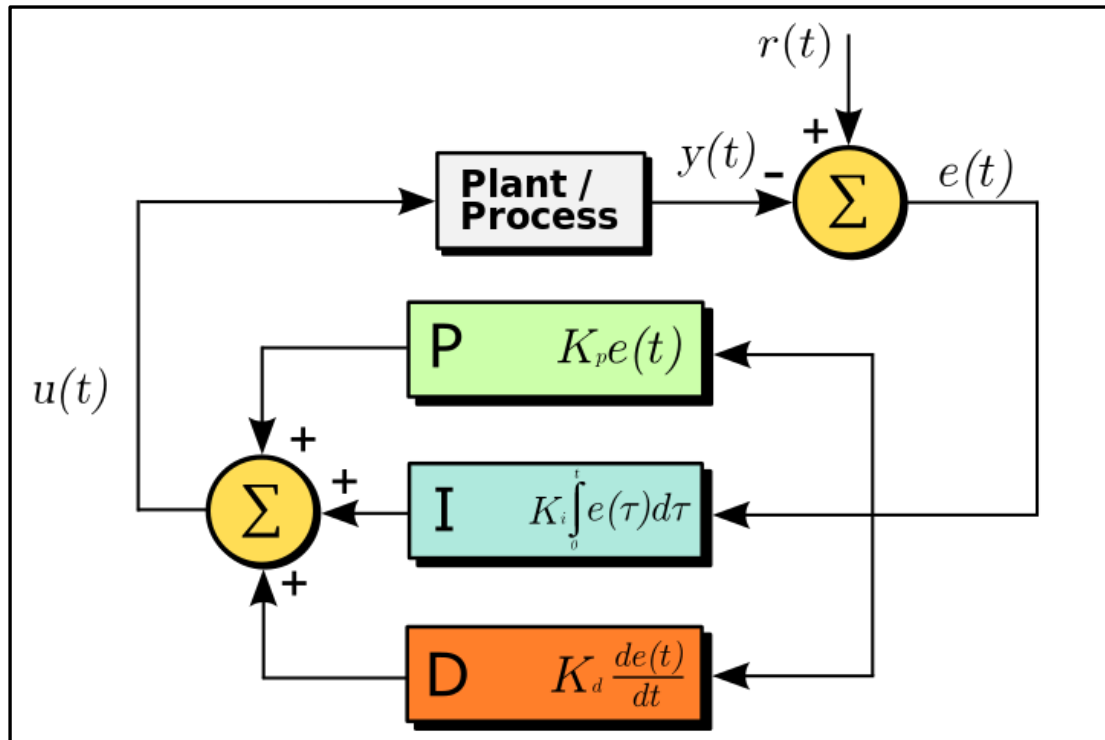


Figure 6.1: Block diagram of a PID controller

In a control system, the parameter to be controlled is called the process variable $y(t)$. It is measured by a sensor and fed back to the control system. The desired value of the system is referred to as setpoint $r(t)$. The difference between setpoint and process variable $e(t) = r(t) - y(t)$ is the error which is used to determine the actuator output to drive the system.

A PID controller uses the error signal $e(t)$ to compute the control signal $u(t)$. For this, derivative and integral of the error signal $e(t)$ are needed. The control signal $u(t)$ is computed as sum of following three coefficients:

- the error signal multiplied by the proportional gain constant K_p
- the integral of the error multiplied by the integral gain constant K_i
- the derivative of the error multiplied by the derivative gain constant K_d

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{d}{dt} e(t) \quad (6.a)$$

In discrete case, the integral of the error is calculated as sum over all previous errors and the derivative error as the difference of successive errors divided by timestep t .

$$u(t) = K_p e(t) + K_i \int_{j=1}^t e(t_j) dt + K_d \frac{e(t_k) - e(t_{k-1})}{dt} \quad (6.b)$$

To describe the effects of each component, some terminology has to be introduced. Most times, a step function is used to measure the response of the process variable. The time it takes to get from 10% to 90% of the final value is referred to as rise time. Overshoot is the percentual amount the process variable exceeds the final value. The time to settle to a value within a certain percentage in most cases 5% of the final value is called settling time. Steady-state error is the final deviation of the process variable from the set point.

Changing the different components has different effect to the output function. The proportional component has the effect of reducing the rise time, but it will also increase the overshoot and it will never eliminate the steady-state error. The integral component can eliminate the steady-state error, but will also slow down the response time. The derivative component can reduce overshooting and improves the response and settling time. Hence, changes in one component have an effect to the other components which makes it difficult to find the right parameters for an unknown system.

It is not necessary to always have all components, sometimes only one or two components are used, e.g. only P, PI, PD or I. The controller is then named accordingly.

A.2: Swarm Intelligence (SI) [4]

Swarm Intelligence is defined as a collective behavior of simple entities having simple rules with ability of local interactions. In this documentation, it is referred w.r.t Artificial Intelligent Systems.

From the theory of Swarm Intelligence, there are three basic rules for swarming which are Attraction, Repulsion and Alignment (ref. Table 2).

Table 2: Basic rules for swarming

Attraction(Cohesion)	To move towards average position of all local agents
Repulsion(Separation)	To move away from the agents
Alignment	To steer towards the average heading of the local agents

The Kinematic equations of motion for individual agents are:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1)$$
$$\vec{v}_i(t+1) = w\vec{v}_i(t) + \vec{f}_i$$

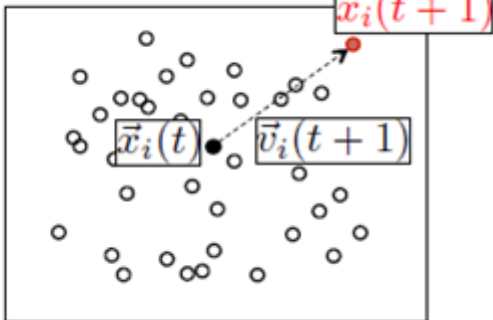


Figure 6.2: Representation of the motion equations.

$\vec{x}_i(t)$ is the position of an agent at time "t"

$\vec{x}_i(t+1)$ is the next position of the agent i.e at time "t+1"

$\vec{v}_i(t+1)$ is the velocity with which the agent moves at time "t+1"

$\vec{v}_i(t)$ is the velocity of the agent at time "t"

\vec{f}_i is the negative of the gradient of a potential function describing the interaction between the individuals.

For this project, the attraction and repulsion function-III is used i.e., linearly bounded attraction and unbounded repulsion (ref. Figure 6.3):

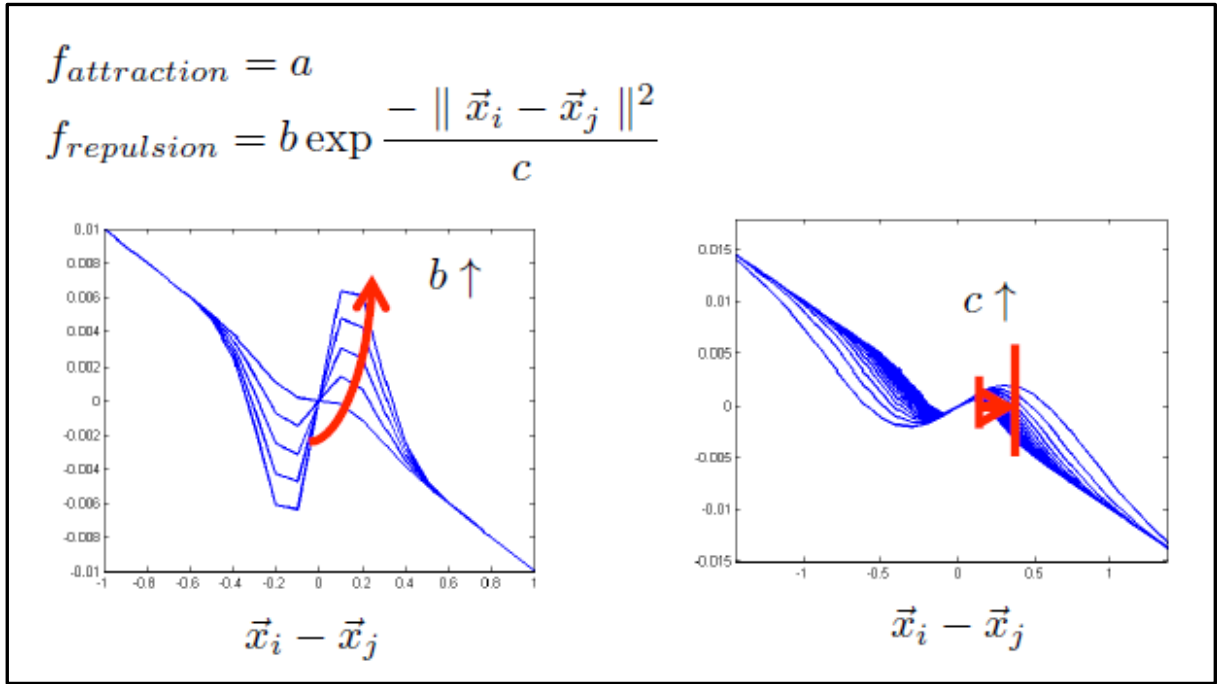


Figure 6.3: Plot of the attraction and repulsion functions

“a” describes the amount of attraction, “b” describes the rate of repulsion and “c” is a factor describing the breadth of the peaks.

For the motion of the swarm, the magnitude and direction of each agent is determined as sum of attraction and repulsion of all other agents (ref. Figure 6.4).

$$\vec{f}_i = \sum_{j=1, j \neq i}^M \vec{f}_i(i, j), \quad i = 1, \dots, M$$

Figure 6.4: Motion of the swarm

For pattern formation and tracking, the agents form a geometric pattern and catch or intercept a target. For each individual agent, a new position is computed according to the following equations (ref. Figure 6.5):

$$\vec{v}_i(t+1) = w\vec{v}_i(t) + w_{cohesion}\vec{f}_i + w_{target}\vec{f}_{target}$$

$$\vec{f}_{target} = f_{attraction}(i, target)$$

where $w_{cohesion} + w_{target} = 1$




Figure 6.5: Pattern formation and tracking

References

- [1] "<http://www.k-team.com/mobile-robotics-products/v-rep>" [Online].
- [2] "<http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=ControlPID>" 2015. [Online].
- [3] P. I. D. Explained, "<http://www.ni.com/white-paper/3782/en/>" March 2011. [Online].
- [4] P. S. Mostaghim, "Swarm Intelligence - Lecture Slides," 2014.