Simulation: External Control of Swarm of Copters

Swarm Lab, WS 2015

Asema Hassan   Johannes Gätjen

March 11, 2016

Contents

1. SCOPE

The SwarmLab at the Otto-von-Guericke-University Magdeburg researches the applications of Swarm Intelligence using quadcopters, called FINken. The arena has a projector installed, with which images can be cast onto the floor. The FINken are equipped with a color sensor to detect the projected images. This way it is possible to give the FINken color-coded control inputs or simulate an environment.

Preliminary work with the copters is usually done in a simulation environment. A previous student project had already modelled the projector and color sensor for the simulation. The main objective of this project is to program simulated copters to react meaningfully to an input image, in particular to be able to follow a heightmap to its highest point. This way it is possible to indirectly control where a copter flies or to control the formation of a swarm of copters.

In order to be able to find the highest point in a landscape, the copter must be able to memorize multiple height values and their relative position. Then it can calculate a gradient, which will point in the direction of a higher point.

We want to be able to quantitatively evaluate how well the copter can find the high points in a map, so we control the simulation via an external script to easily run multiple simulations and gather data.

Finally, the code needs to be extended in such a way, that an entire swarm of copters can be simulated. Then it is also possible to let the copters interact in some ways.

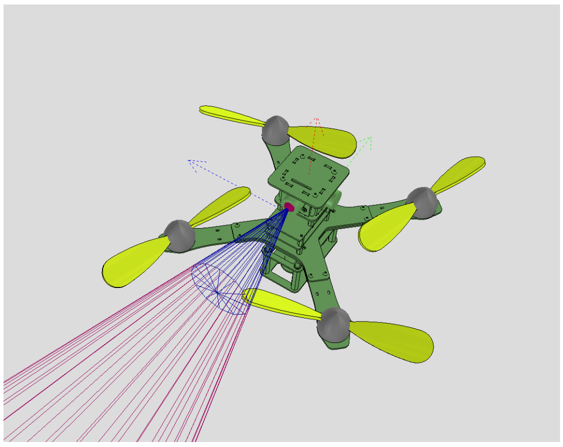


Figure 1: FINken model used in simulations of V-Rep

2. INTRODUCTION

Swarm Intelligence is defined as the collective behavior of decentralized systems whether natural or artificial [5].

The SwarmLab uses quadcopters to study Swarm Intelligence, but using real robots is expensive and time consuming. To eliminate the risk of breaking expensive equipment and avoid dealing with tricky problems caused by difficult to control variables preliminary studies are done in a simulation environment, before being transferred to the real copters.

2.1. V-Rep

We use V-Rep, a powerful 3D robotic simulator as our simulation environment. Its many features include inverse kinematics, physics/dynamics, collision detections, minimum distance calculations, path planning etc. It supports a distributed control architecture, i.e. an unlimited number of threaded or non threaded control scripts as well as plug-ins to set simulation parameters and signals. Within V-Rep the behavior of objects can be controlled with Lua scripts.[4] V-Rep can also be controlled via a remote API for several programming languages such as C++, Java and Python. Lastly, a feature the importance of which should not be underestimated is a very good documentation.

2.2. FINken

The FINken model shown in Figure 1 used in simulations was designed by the Chair of Intelligent Systems at OvGU. It has been created in V-Rep and resembles a real quadcopter in terms of propellers, static parts and sensors. The aerodynamics are done with particle simulation. The copter can be controlled by the four parameters throttle, pitch, roll and yaw, as shown in Figure 2.

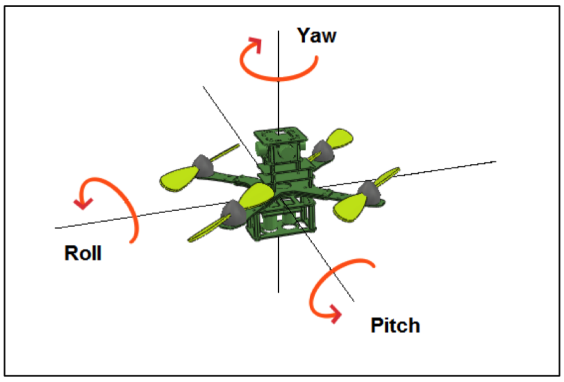


Figure 2: Quad-copter Angular parameters

2.3. Background

The project builds on previous work on simulation of copters. The arena is not modelled in detail, but only as a static floor and walls. To enable a simple control of the copters a previous group developed a PID controller. With this it is possible to specify a target and the PID controller will determine the pitch, roll and yaw values so that the FINken flies towards it. For more details refer to the documentation of the SS15 group directory [6]. The purpose of the projector is to provide an image input across the area of the arena. To model this in simulation an image is applied as a texture to the floor of the arena. The color sensor is then a one-pixel camera pointed towards the floor. This is a very idealized model, as several consequences of using a projector and color sensor, such as a dependence of the color value on the height of the copter, are completely ignored. A rudimentary gradient following behavior had been implemented by the SS15 group, but it was not very effective.

A first step was to generate possible height or gradient map images using MATLAB scripts. Initially, the gradient was read directly from the color values of the map image and set as a direction for the copter. Later only the height information was used to calculate a gradient during runtime. A major part of the project was to create a local map, where the copter can store information it has gained about its surroundings. By storing multiple values it is possible to calculate a gradient after enough information has been collected.

For evaluation it is necessary to run the simulation several times with different sets of parameters. We chose Python to create an external control script using the remote API and repeated the simulation several times for every different parameter setting. The parameters are set in the form of signals, which can be seen as global variables and are used for communication into and out of V-Rep [**VrepSignal**]. At the end of each simulation, a log file is created which saves the height value at each time step to analyze the behavior of copters with different parameter settings. In the next sections we cover the details of each milestone and the limitations.

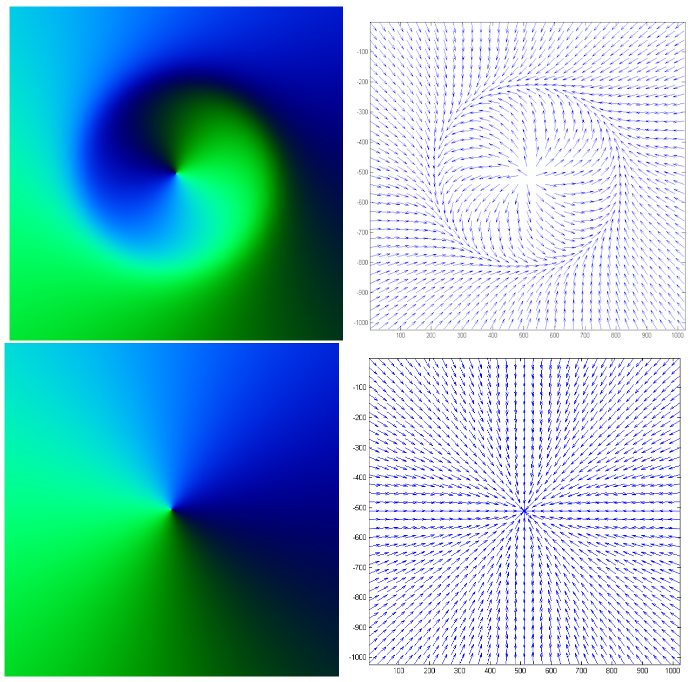


Figure 3: Left: Two examples of gradient maps using the values for green and blue. Right: Corresponding visualizations of the maps as vector fields.

3. MILESTONES

We have divided our project tasks into the following milestones.

3.1. Gradient and height maps

Several input images were needed to evaluate the behavior of the FINken. We chose to generate them with MATLAB scripts script[1], and encoding the height in the intensity of red, and the X and Y gradient in the intensity of green and blue intensity respectively. Having direct access to the gradient values is not realistic, but allowed us to start with a simple task to familiarize ourselves with V-Rep and Lua, where the gradient encoded in the color is used directly to set the direction of the FINken. Figure 3.1 shows two examples of gradient maps, where the height was actually not included, i.e. red has zero intensity. Figure 3.1 shows a map where height and gradient are included, and the same map without the gradient, i.e. only red is non-zero. Including the gradient encoding in principle also allows us to compare a gradient that is computed by the copter with the ground truth value, but we did not use this in practice.

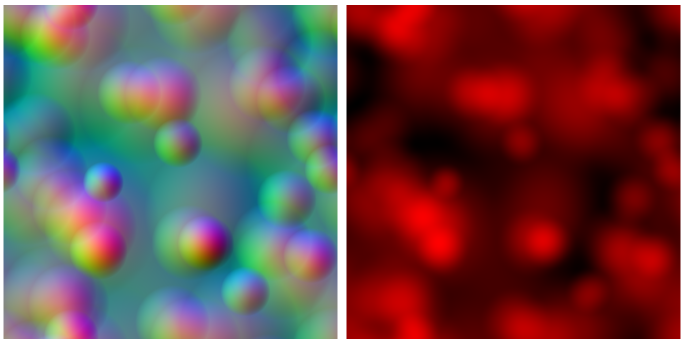


Figure 4: Left: An example of a heightmap with randomly placed hills. Right: The same heightmap, but not containing explicit gradient information.

3.2. Initial gradient following

A one pixel vision sensor is attached to the body of the FINken, pointing towards the floor, which returns the color value of the landscape directly below the FINken

For the initial gradient following behavior, the gradient is computed directly from the green and blue intensities and the target is set in the direction of the gradient. The PID controller then makes the copter fly in the direction of the target. If there are maxima implied by the gradient map, the copter will eventually reach one. However, since we do not necessarily include the height in the maps, it is possible to construct gradient maps where the copter will never converge. See e.g. the top image in Figure 3.1.

A demo video of the initial behavior of gradient following can be seen in the Simple-GradientFollow movie file in the project repository [3].

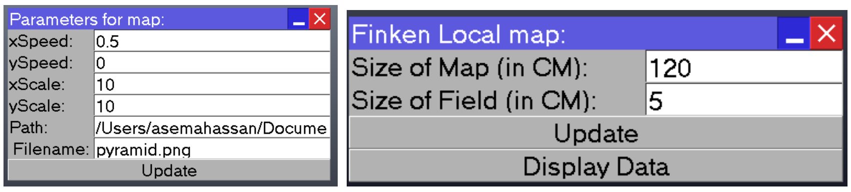
3.3. User Interface (UI)

V-Rep provides an editor to create custom user interfaces for better control of simulation parameters. It has many visual controls and can also be accessed via scripts. The UI consists of buttons, labels, input fields, images, background images, graphs and floating view of vision sensors [7].

The idea of creating a custom UI in this project was to give the user the ability to choose the parameters he wishes to change before starting the simulation. Such as the input image path, scale, speed. Figure 3.3 shows the two UI elements created in the project.

3.4. Dynamic input image

The idea of changing input images is to load the new input image at run time. This is controlled by the UI given for the map. The values from the UI are saved as signals and are used to create a new texture at run time and assign it to the landscape object (the floor of the arena).


Parameters for map:

xSpeed:	0.5
ySpeed:	0
xScale:	10
yScale:	10
Path:	/Users/asemahassan/Docume
Filename:	pyramid.png

Update

Finken Local map:

Size of Map (in CM):	120
Size of Field (in CM):	5

Update

Display Data

Figure 5: Left: Parameters for the landscape input image. The user can specify the path to the image and also set its scale and speed. The speed was removed in later versions. Right: Control of the local map (see also section 3.6). The user can specify the field size and total size of map from which the resolution of the map is computed. The display data button is used to print the data currently stored in the local map.

After the input image is being set, the main task is to change the input dynamically, that is it should be moving. To achieve this result, we started from changing the UV's of the texture. The UV's of the texture are basically its X and Y offset values, that when changed can shift the texture on the given shape handle mesh. When this is done repeatedly it gives an effect of a moving texture. The UV's can be changed either from texture properties in V-Rep or via script. The parameters shown in UI Map as xSpeed and ySpeed are used for the movement along U or V (can be both).[9]

In the final state of project, instead of only changing UV's of the texture the whole arena is being moved with the FINken. This gives the impression of the texture being inert with respect to the global coordinate system, but internally the UV's need to be updated constantly. The result can be seen in the Large_hills_smooth_gradient_map video file in the project repository [3].

3.5. Position sensor

Building a local map requires information of the FINken position. Two basic options to implement a position sensor are possible: A global positioning system (GPS) or an inertial measurement unit (IMU).

Every system has its own advantages and disadvantages. GPS gives the copter their absolute position but is not very accurate (compared to IMU). It limits the field of applications to be only used outside in an open environment where there is considerable room for error. However, it is easy to use and implement. On the other hand, an IMU gives the relative position of a copter and is only accurate for a limited amount of time. As the IMU is already installed in the real quad-copters and knowledge of the absolute position is not necessary for the local map we decided to implement an IMU in the simulation. We created a dummy object, which in principle could be attached to any simulation object. In the associated script for the object the velocities are read from the simulation. Because the simulation gives perfectly accurate velocity values, which is not the case in the real world, we added some white noise to the data and set these as simulation signals. To estimate a realistic value for the noise magnitude we also analyzed

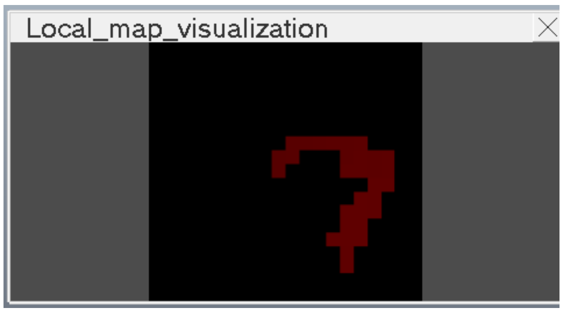


Figure 6: The visualization of a local map. The black parts are parts of the map that are not filled, or could in principle also mean the recorded height value at that point is zero.

some real copter IMU data, but the sampling rate of the data was too low to get any meaningful results.

3.6. Local map

At each time step, the height value that is read by the color sensor is saved into the local map of the FINken. The local map is needed to store multiple values as the FINken is moving, since the color sensor can only ever record a single value at a time. The local map is a data structure which can hold these values and moves them in relation to the FINken.

The local map is a virtual area around the FINken. It can be defined as a local boundary of the FINken where it is always staying at the center of the local map and values are updated when the FINken has moved a certain distance.

Technically the local map is a 2D Array with a defined resolution. The resolution is computed from the user input as UI parameters or from the signal values set via the remote API. The color sensor value, such as height, is stored in the center of the 2D Array at each time step. The data is then shifted within the 2D Array according to the FINken movement and direction.

A run time texture is created from the resolution of the local map and then data from the local map array is written on that texture at each time step. A vision sensor is used to look at the texture that is being updated for the local map, thus creating a visualization of the local map. Figure 3.6 shows an example of such a visualization.

3.7. Local map based gradient following

In order to calculate both the x and the y gradient from the local map there need to be at least three values arranged in an L-shape around the center, that is the copter position, as shown in Figure 3.7. A flowchart with the baseline gradient following behavior using the local map is shown in Figure 3.7. In the beginning, the FINken checks at every time step whether it is possible to calculate the local gradient. If the data is not sufficient it

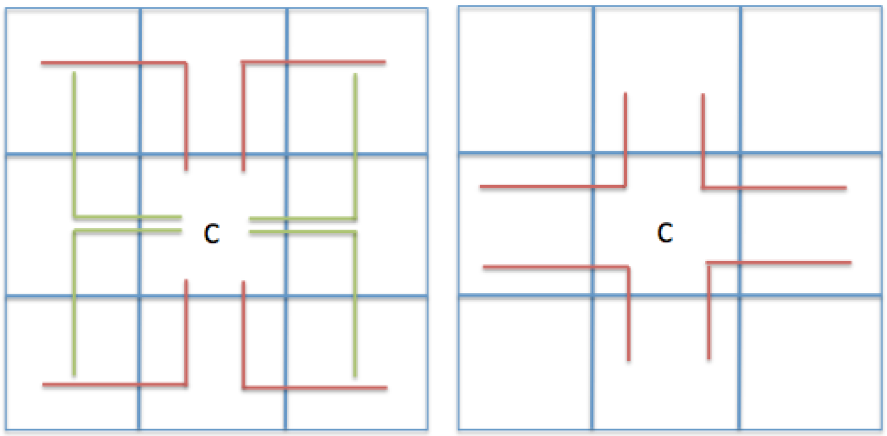


Figure 7: A visualization of the 12 different possible ways to estimate the gradient from the local map. We always know the center field has a stored value, because it is the position of the copter, so we always include this field in the gradient estimation. The eight possibilities on the left technically estimate the gradient of a neighboring field, but in practice this is accurate enough and it is much easier to ensure, that the copter flies in a way to leave such a pattern in the local map.

will use a simple exploration scheme, where it tries to fly in a direction perpendicular to the direction it was flying previously. This way it is likely to obtain an L-shape structure in the local map data. When the gradient can be calculated the target is set at a fixed distance in the gradient direction and the FINken will fly towards the target. Once it arrives in the proximity of the target it will try to calculate the gradient again and set a new target.

Based on the baseline behavior we implemented two modes for the gradient following, a straight mode and a drunk mode. In the straight mode the copter will fly directly towards the target once a gradient has been estimated. In the drunk mode the copter will set several checkpoint targets positioned alternating left and right of the path to the target. In effect the copter will fly in a sinuous line towards the target, thus achieving a better exploration of the map. A visualization of the modes is shown in Figure 3.7. We also implemented a random walk mode where any color sensor information is ignored, which we use as a baseline behavior in the evaluation.

3.8. Remote API

With the remote API the simulation can be started, paused and stopped at any time. Also, the parameters used in simulation which are in the form of signals can be set from the remote API easily. This way we can easily run a simulation many times with different parameter settings [8]. We decided to use Python for the remote API script [2].

Following is the list of parameters that are set as signals in the simulation and they

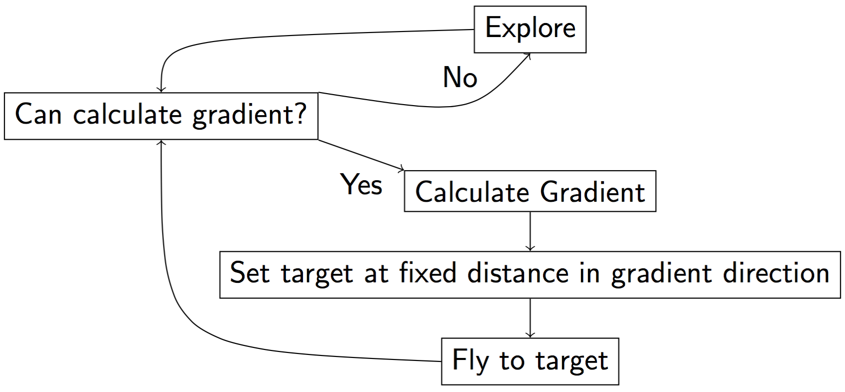


Figure 8: The basic scheme for the gradient following behavior using the local map.

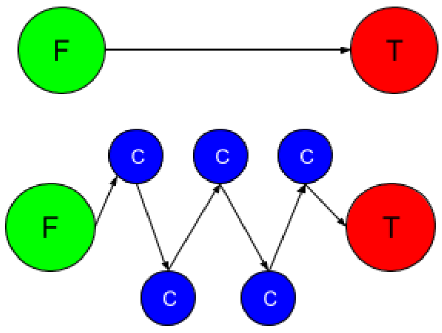


Figure 9: A comparison of the straight (top) and drunk (bottom) modes. F=FINken, T=Target, C=Checkpoint

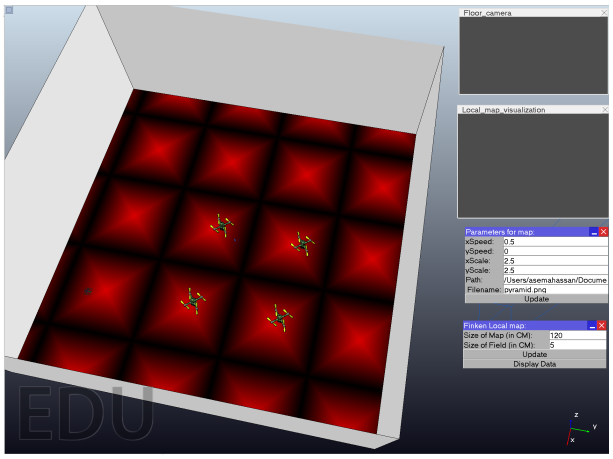
can also be set via remote API for analysis of FINken behavior. Otherwise if not set from the external control the simulation will still work with default values.

1. Height map image file and scale
2. Local map total size and field size (resolution implicit)
3. Mode (straight, drunk, random)
4. Gradient speed: How far away the target is set after the gradient has been calculated.
5. Explore speed: How quickly the copter will move during exploration.
6. Target epsilon: How close the copter has to get to the target before it is considered reached.
7. Width factor (drunk only): How far apart the checkpoints are from the center line.
8. Step factor: How far apart consecutive chekpoints are (in gradient direction)
9. Checkpoint epsilon ratio: How close the copter has to get to a checkpoint before it is considered reached.
10. IMU noise: The standard deviation of the white noise for the IMU velocities.
11. Map average: Whether to average or replace values written into the same field in the local map
12. Map replace: Upon entering a field in the local map with a preexisting value, whether to replace the value or keep it (for averaging)
13. Map alpha: A factor that controls how strongly new values in the local map are weighted during averaging, compared to old values

Unfortunately, this high number of parameters makes it difficult to evaluate different settings. We chose to focus on comparing different modes and noise levels, with parameter settings, that we knew to work fairly well. The demo video for the remote API titled RemoteApiController can be found in the project repository [3].

3.9. Swarm Behavior

A basic first idea for a swarm behavior is to set the landscape image such that, even without communication between copters and only using the gradient following behavior, every FINken stays within a restricted region and they do not collide. The code was adjusted such, that it is possible to simply duplicate the FINken object in the scene. Unfortunately, on our computers the simulation is quite slow with even a handful of copters and due to time constraints we were not able to do any formal evaluation of the swarming behavior.



Parameters for map:

xSpeed:	0.5
ySpeed:	0
xScale:	2.5
yScale:	2.5
Path:	/Users/asemahassan/Docume
Filename:	pyramid.png

Finken Local map:

Size of Map (in CM):	120
Size of Field (in CM):	5

Figure 10: Swarm behavior idea with 4 FINken

4. RESULTS

To evaluate the FINken behavior of following the gradient we implemented a logging behavior where we save the height value from the color sensor at each time step and write them to a file at the end of the simulation. A three minute long simulation is repeated 25 times with random starting positions for every set of parameters we want to test. For the details of the parameter settings and the plots refer to the appendix.

4.1. Test 1: Comparison of modes on Pyramid map

The first evaluation results shows the comparison of FINken behavior in different modes where modes are represented as 0,1 and 2 for straight, drunk and random respectively, with the pyramid map as an input image.

4.2. Test 2: Comparison of modes on Hills map

The second evaluation results shows the comparison of FINken behavior in different modes, using a map with randomly placed hills scattered across the map.

4.3. Test 3: Comparison of noise levels with drunk mode

The third evaluation results shows the comparison of FINken behavior in drunk mode only, using the pyramid map as an input image and the same parameter values as from the Test 1 with different variations of IMU noise starting from 0, medium as 0.05, high as 0.15 and very high as 0.5.

4.4. Test 4: Comparison of noise levels with straight mode

The fourth evaluation results shows the comparison of FINken behavior in straight mode only, using the pyramid map as an input image and same parameter values as from the Test 1 with different variations of IMU noise the same as in Test 3.

5. CONCLUSION

Based on the results and evaluation of FINken behavior, it can be said that overall results were satisfying. The FINken gradient following behavior is working effectively and the FINken is never leaving the arena boundaries.

We can generate random parameterized landscape images using MATLAB scripts. The white noise added in velocity sensor makes the result more realistic. The idea of building a local map around FINken in order to explore its surroundings, is working efficiently for all directions. The gradient calculated at runtime from local map serves the purpose of dynamically changing landscape and behavior of FINken in response to it.

From the evaluation, it is worth noticing that the straight mode is faster than the drunk flying mode because it doesn't take any detours. The drunk mode collects more information, but it does not make any use of the additional information. The straight mode is also more robust against higher IMU noise values.

Unfortunately, there are many different parameter settings possible to evaluate the FINken behavior. Because of the time limitation we did not simulate more parameter sets.

For further research, the color sensor value needs to be tweaked with some noise as at the moment it is unrealistically accurate. Also the velocity sensor noise should also be evaluated from the real data in future. The external control of swarm behavior can also be a possible research. There may also be applications where it is possible to incorporate more long-term memory or global information, which may also make the drunk mode more useful.

References

[1] Johannes Gätjen Asema Hassan. *External Control of Swarm of Copters, Matlab scripts.* URL: https://github.com/ovgu-FINken/Simulation/tree/ws16Ext/resources/gradient_maps.

[2] Johannes Gätjen Asema Hassan. *External Control of Swarm of Copters, Matlab scripts.* URL: https://github.com/ovgu-FINken/Simulation/tree/ws16Ext/finken_control/ExternalApiCall.

[3] Johannes Gätjen Asema Hassan. *External Control of Swarm of Copters, Videos.* URL: https://github.com/ovgu-FINken/Simulation/tree/ws16Ext/resources/presentation.

[4] *Robotics Simulator V-REP.* URL: http://www.k-team.com/mobile-robotics-products/v-rep.

[5] Prof. S.Mostaghim. *Swarm Intelligence - Lecture Slides.* URL: http://www.is.ovgu.de/Teaching/WS+2015_2016/Swarm+Intelligence.html.

[6] Andrei Stein et al SS15 Swarm Lab Project. *Flying in Formation.* URL: https://github.com/ovgu-FINken/Simulation/tree/ss15/finken_swarm/Documentation.

[7] *V-REP Custom User Interface.* URL: http://www.coppeliarobotics.com/helpFiles/en/customUserInterfacePropertiesDialog.htm.

[8] *V-REP Remote API.* URL: http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm.

[9] *V-REP Texture properties.* URL: http://www.coppeliarobotics.com/helpFiles/en/textureDialog.htm.

A. Evaluation parameters and plots

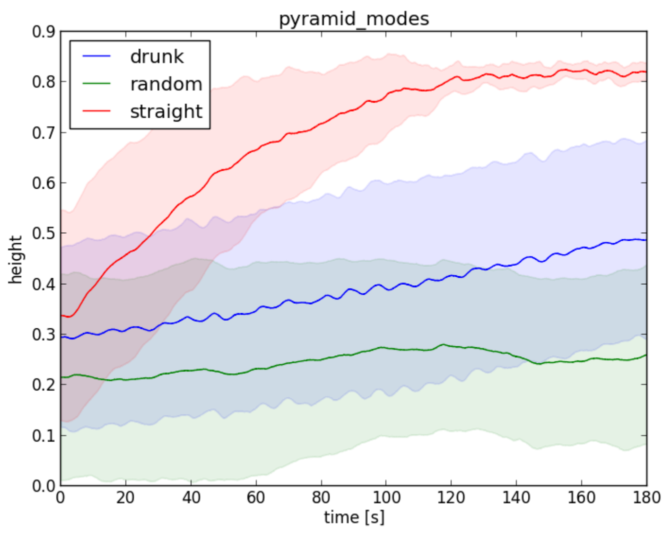
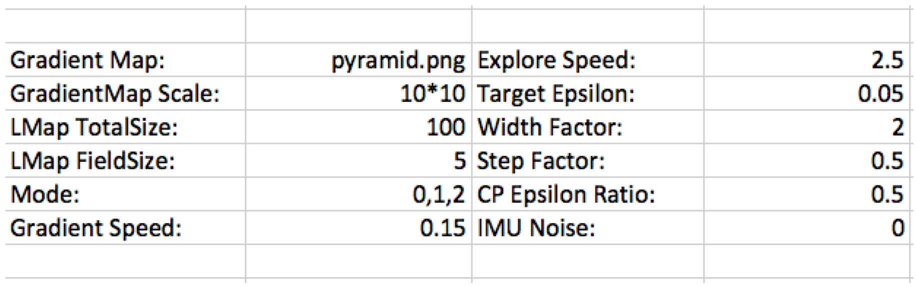


Figure 11: Comparison of modes on pyramid map, with average height over time. The shaded area shows the average ± one standard deviation. Both the straight and drunk mode are considerably better than the random mode, with the straight mode being fastest. Towards the end of the simulation the height for the straight mode stops rising and the standard deviation becomes smaller, indicating that a large fraction of the copters are close to the highest point in the map. For the drunk mode on the other hand, there is no clear decrease in the rise or the standard deviation, indicating, that this would continue to rise, given more time.



Gradient Map:	pyramid.png	Explore Speed:	2.5
GradientMap Scale:	10*10	Target Epsilon:	0.05
LMap TotalSize:	100	Width Factor:	2
LMap FieldSize:	5	Step Factor:	0.5
Mode:	0,1,2	CP Epsilon Ratio:	0.5
Gradient Speed:	0.15	IMU Noise:	0

Figure 12: Parameters set for the pyramid map

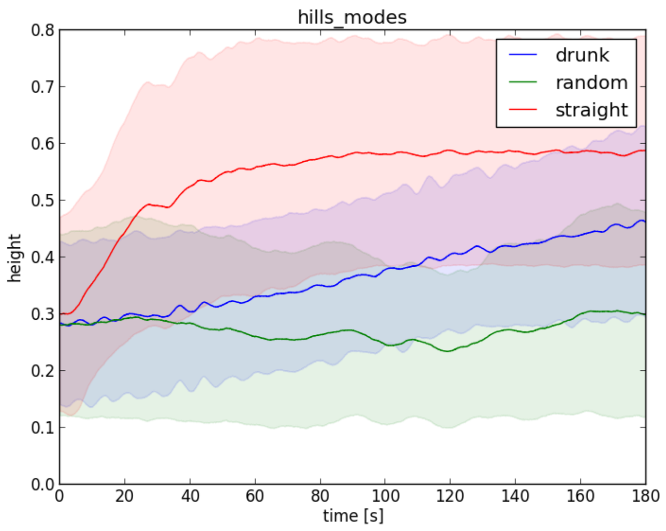
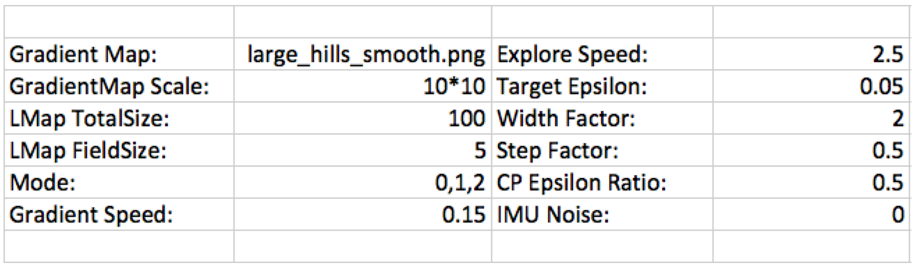


Figure 13: Comparison of modes on hills map, with average height over time. The shaded area shows the average ± one standard deviation. Again, the straight mode is best and drunk mode is better than random. In this case the standard deviation for the straight mode does not decrease towards the end of the simulation. This is probably because due to the random starting positions it is likely that some copters get stuck in a local maximum, while only some reach the global maximum.



Gradient Map:	large_hills_smooth.png	Explore Speed:	2.5
GradientMap Scale:	10*10	Target Epsilon:	0.05
LMap TotalSize:	100	Width Factor:	2
LMap FieldSize:	5	Step Factor:	0.5
Mode:	0,1,2	CP Epsilon Ratio:	0.5
Gradient Speed:	0.15	IMU Noise:	0

Figure 14: Parameters set for the hills map

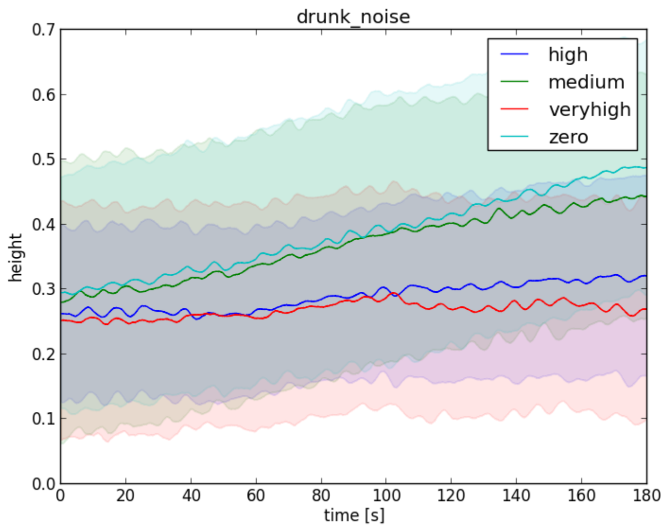
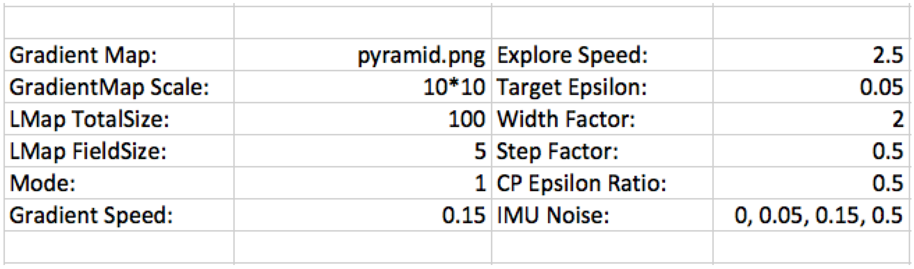


Figure 15: Comparison of IMU noise levels on pyramid map with **drunk** mode. As expected, the higher the noise is, the slower the copters can find higher points. With high and very high noise levels the performance is virtually indistinguishable from a random walk.



Gradient Map:	pyramid.png	Explore Speed:	2.5
GradientMap Scale:	10*10	Target Epsilon:	0.05
LMap TotalSize:	100	Width Factor:	2
LMap FieldSize:	5	Step Factor:	0.5
Mode:	1	CP Epsilon Ratio:	0.5
Gradient Speed:	0.15	IMU Noise:	0, 0.05, 0.15, 0.5

Figure 16: Parameters set for the pyramid drunk mode with IMU Noise levels

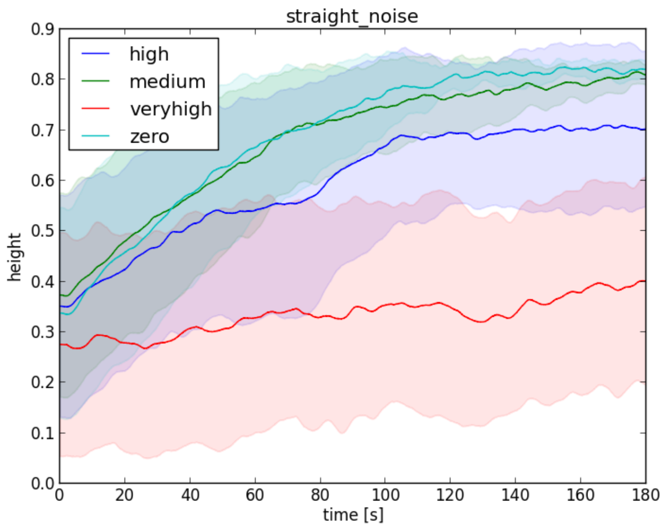
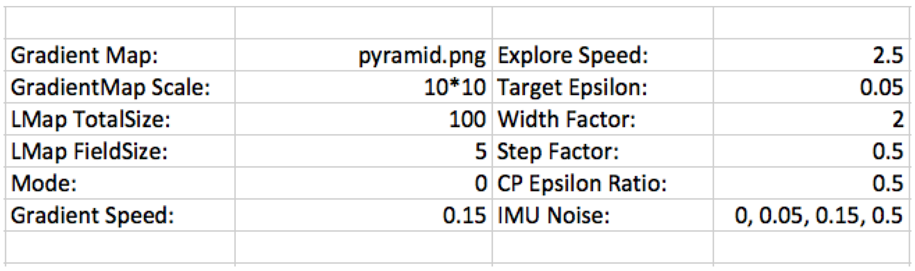


Figure 17: Comparison of IMU noise levels on pyramid map with **straight** mode. Again, higher noise values mean the higher parts of the map are reached more slowly. Compared to the drunk mode, the straight mode appears to be more robust against noise, as it will still reliably follow the gradient with high noise levels.



Gradient Map:	pyramid.png	Explore Speed:	2.5
GradientMap Scale:	10*10	Target Epsilon:	0.05
LMap TotalSize:	100	Width Factor:	2
LMap FieldSize:	5	Step Factor:	0.5
Mode:	0	CP Epsilon Ratio:	0.5
Gradient Speed:	0.15	IMU Noise:	0, 0.05, 0.15, 0.5

Figure 18: Parameters set for the pyramid straight mode with IMU Noise levels