ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Navigational systems of Desert Ants

Remo Breitenmoser & Anton Markaj & Dejan Mircic

Zurich
December 2011

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Remo Breitenmoser Anton Markaj Dejan Mircic

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

_____

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
_____	_____

Supervising lecturer

Last name	First name
_____	_____

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature
_____	_____

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Contents

1 Individual contributions **6**

2 Introduction and Motivations **6**

3 Description of the Model **7**
3.1 Path integration based on global vectors 7
3.2 Landmarks and local vectors 9
3.3 Interaction of the navigational tools 9

4 Implementation **10**
4.1 Ant 10
4.1.1 Global vector 10
4.1.2 Local vector 10
4.1.3 Landmarks 10
4.2 Testing Area 11
4.2.1 Channel 11
4.2.2 Cylinders 11
4.3 Experiments 11

5 Simulation Results and Discussion **12**
5.1 Two-leg trajectory 12
5.1.1 Three-leg trajectory 15
5.2 One-leg trajectory 16
5.3 Cylinder corridor 17
5.3.1 Crooked corridor 18
5.4 Moved nest 19
5.5 Natural foraging 20

6 Summary and Outlook **22**
6.1 Summary 22
6.2 Outlook 22

7 References **23**

A MATLAB Code **24**
A.1 main.m 24
A.2 Ant.m 25
A.3 Landmark.m 30
A.4 Area.m 31

A.5 Cylinder_Area.m 32
A.6 Channel.m 34
A.7 two_leg_trajectory.m 36
A.8 three_leg_trajectory.m 38
A.9 one_leg_trajectory.m 40
A.10 moved_nest.m 43
A.11 crooked_corridor.m 47
A.12 corridor_of_black_cylinders.m 51
A.13 natural_foraging.m 55

1 Individual contributions

The project was created in a cooperative manner.

2 Introduction and Motivations

Desert ants, *Cataglyphis fortis* have to walk dozens of meters to find food. To make sure they find the way back to their nest, they need some sort of navigational tools. Since the brain of a desert ant is quite simply built, scientists assume that the ant cannot have the whole visible area stored as an image within its mind. It is amazing to see that these small animals anyhow find their long way back home. It seemed interesting for us to study the behaviour of desert ants. Not only because of this astonishing performance, but also because of the fact that neurologists begin to use the understanding of the ants' navigational behaviour. This knowledge can be used for the research of humans' spatial representations. Recent research discovered that the desert ant makes use of various navigational aids while roaming through terrain, such as approximating the direct way back to the nest, and use the sun as a reference for the cardinal directions.
We asked ourselves how these different navigational aids collaborate, and if their modeling is able to describe the behaviour of real ants good enough. Our concrete questions are:

- What's the interaction between the different navigational tools employed by the desert ant?
- What happens if one of those tools is detained?
- How can landmark based navigation (one of the tools) be appropriately implemented?
- How do slight variations in amount and distribution of landmarks affect foraging efficiency?
- Is the model applicable to the ants' natural foraging behaviour?

3 Description of the Model

When it comes to navigational tasks, it has been shown that the desert ants' behaviour can be modelled by describing several essential mechanisms, so-called "navigational tools". The fundamental concept of this navigational toolkit is path integration based on global vectors as stated in the paper *Path integration in desert ants, Cataglyphis fortis* [MW88]. We extended this model by the concept of local vectors associated with landmarks, which are described in *Local and global vectors in desert ant navigation* [CCBW03]. Further aids used by navigating ants are the sun compass and the polarization compass which provide the ant spatial information such as cardinal directions (*Desert ant navigation: how miniature brains solve complex tasks*, [W03]). However, celestial compasses are not part of our model.

3.1 Path integration based on global vectors

Path integration is the ability to continuously compute one's current position based on previous movement and as a result to know the direct route back to the starting point at any given time. Many animals that travel over long distances have been shown to use path integration [MW88]. However, compared to these species, an ant possesses a much smaller brain which is not able to perform such tasks by using complex methods (i.e. vector addition), as they also produce navigational inaccuries. It is rather assumed that desert ants use a simple approximation to compute their mean direction and distance as they forage.
Such an approximation can be expressed by the following equations stated in [MW88]:

$$l_{n+1} = l_n + 1 - \frac{\delta}{90^\circ} \tag{1}$$

δ	Angle about which the ant has turned in the current step.
l_n	Covered distance in unit lengths after the n-th step.

For the mean direction, there are two different versions provided:

$$\varphi_{n+1} = \frac{l_n\varphi_n + \varphi_n + \delta}{l_n + 1} = \frac{\varphi_n(l_n + 1) + \delta}{l_n + 1} \tag{2}$$

$$\varphi_{n+1} = \varphi_n + k\frac{(180^\circ + \delta)(180^\circ - \delta)\delta}{l_n} \tag{3}$$

φ_n	Direction in which the ant has covered the distance δ.
k	Constant that normalizes the product.

In equation (2), $(\varphi_n + \delta)$ denotes the direction in which the ant proceeds for the following unit length. The approximation is distance-weighted. This means that the mean direction φ_n changes less the longer the covered distance l_n is. Thus φ_n gets multiplied with $(l_n + 1)$ (and later on divided).

Equation (3) describes an improved solution. In this version the ant is assumed to take its previous homeward course $(\varphi_n + 180°)$ as the reference vector. By multiplying the two possible angular differences $(180° + \delta)$ and $(180° - \delta)$, the deviation of the new direction $(\varphi_n + \delta)$ from the reference vector can be determined approximately. To normalize that product to not reach more than 90°, it is multiplied by k.

As equation (2) turned out to be very inaccurate under certain test conditions in [MW88], we use the approximation denoted by equation (3), which suited the measured results in [MW88] of real ants sufficiently well, for our model.
Since φ_n stands for the mean direction during the ants' foraging, its opposite direction $(\varphi_n + 180°)$ always points to the nest and is thus denoted as the global vector.

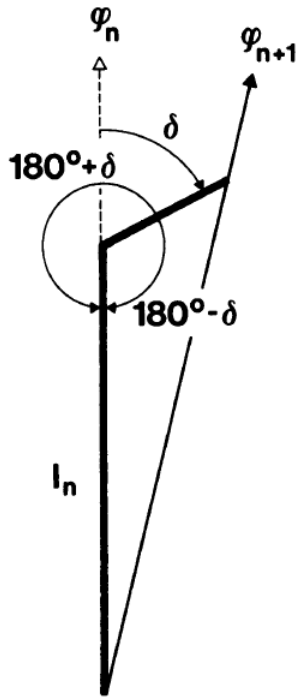

Figure 1: Illustration of the mean direction formula, as depicted in [MW88]

3.2 Landmarks and local vectors

Desert ants returning from a foraging trip to their nest navigate next to path integration also by visual landmarks.
[CCBW98] illustrates that such multi-segmental journeys are composed partly of stored local movement vectors, which are associated with landmarks and are recalled at the appropriate place.
It also shows that the expression of the global vector seems to be temporarily inhibited while the local vector is used.
There are several experiments executed as indicated in [CCBW98] to figure out what exactly happens to the global vector during landmark navigation by testing interactions between global and local vectors where ants are put in situations in which the two vectors point in different directions.
What actually has been discovered is, that the global vector is probably continually updated while it is suppressed, so that its value is always appropriate for guiding the ant home.

3.3 Interaction of the navigational tools

In real life the orchestration of the different navigation tools is rather complex. Cataglyphis fortis can store only a finite number of landmarks. Which landmarks and how long they are stored is not known. In our experiment we assumed that desert ants can store infinite landmarks. We also assumed that they favour local vectors over the global vector. Implemented ants always follow a local vector belonging to a landmark whenever they get into the range of a landmark. They follow this local vector until they leave the range of the aid to orientation.

4 Implementation

Basically MATLAB is intended primarily for numerical computing. But it is also possible to program in MATLAB as in an object-orientated language. We decided to use classes with properties and methods. Important for all experiments were the ants and their environment. Therefore we implemented the classes Ant, Area and Landmark. Furthermore it was handy to have a class Channel. For the experiments we used scripts which made use of the classes.

4.1 Ant

The class *Ant* describes a single represenation of an ant and is the underlying class for every experiment. It contains all relevant properties and methods which were required for the simulation. The properties as well as the methods are divided in different categories for consistency and readability reasons.
In the following, the implementations of the most important concepts are outlined.

4.1.1 Global vector

The computation of the ants' mean direction and mean length occurs according to equations (1) and (3) given in Chapter 3.1. Every angle is set in radian measure, as MATLAB originally works with this angular unit. The two-component mean and global vector are computed with sine and cosine of the current direction φ and its opposite $(\varphi + \pi)$, respectively.

4.1.2 Local vector

Local vectors are always corresponding to a specific landmark that has been set by the ant. The value of the local vector equals to the value of the global vector by the moment the ant created the landmark.

4.1.3 Landmarks

The concept of landmarks has been implemented as a separate class, since every landmark represents an autonomous object. Every ant maintains a list of landmarks, which are being created during the foraging trip. Mentionable properties of class *Landmark* are *length* and *range*. *length* defines the length of the corresponding local vector which determines the distance the ant moves in the respective direction, if it crosses the landmark. *range* defines the circular range within whose the ant "recognizes" its previously set landmark and acquires the corresponding local vector.

4.2 Testing Area

The area in which all the experiments are held has, depending on what kind of experiments the ants are passed through, either the size of $10m^2$ or $15m^2$ whereat 1m corresponds to 100 units in MATLAB.
The area generally consists of a feeder and a nest towards which the ants aspire while foraging and returning. Additionally there are several other components appended on the map on which some experiments are based.

4.2.1 Channel

Since channels were used in a couple experiments it was useful to have a class channel which was usable in all experiments. It is possible to define one-, two- or three-legged channels. They can be defined by their edge points. The first node is the entrance and the last point is the exit of the channel.

4.2.2 Cylinders

In the *Corridor of black Cylinders* experiment [CCBW98]. We used the greater area of $15m^2$ (1'500 units in MATLAB). The cylinders are created simultaneously with the cylinder area as the amount and position of the are managed by the input argument during creation of the cylinder area.
The size of each cylinders is corresponding to [CCBW98] 40 cm which equals to 40 pixel units. The cylinders are arranged as in the experiment [CCBW98, Fig.4] in form of a corridor whereat on each side of the corridor 3 cylinders are placed and slightly shifted parallel to one another. The range on which the cylinder can be seen by the ant varies by each cylinder and is determined at the same time the ant puts the landmark.

4.3 Experiments

We implemented various tests which have been documented in our references. Our main purpose was to reconstruct the given test results with the predefined model (global vectors) and our implementation of landmark based navigation. There are three types of experiments which we implemented: Channel based experiments, cylinder based experiments and a final natural foraging experiment which simulates roughly a natural situation (as of our thoughts).
Every experiment was implemented as a function with various inputs and ouputs. All of them are called one after another in the *main* script.

5 Simulation Results and Discussion

5.1 Two-leg trajectory

This experiment contains a two-legged channel, through which the ant is forced to move. The feeder is located at the end of the channel. As soon as the ant gets to the feeder, it collects food and returns to the nest on the open area, where its trail is being recorded. Due to the approximative way the ant calculates the global vector, errors can occur, which lead to homing directions that differ from the actual direction. This deviation is being captured at the exit of the channel. Below are the simulation results for different turning angles ($0° \leq \alpha \leq 180°$) and different values for k (see equation (3) on page 7).

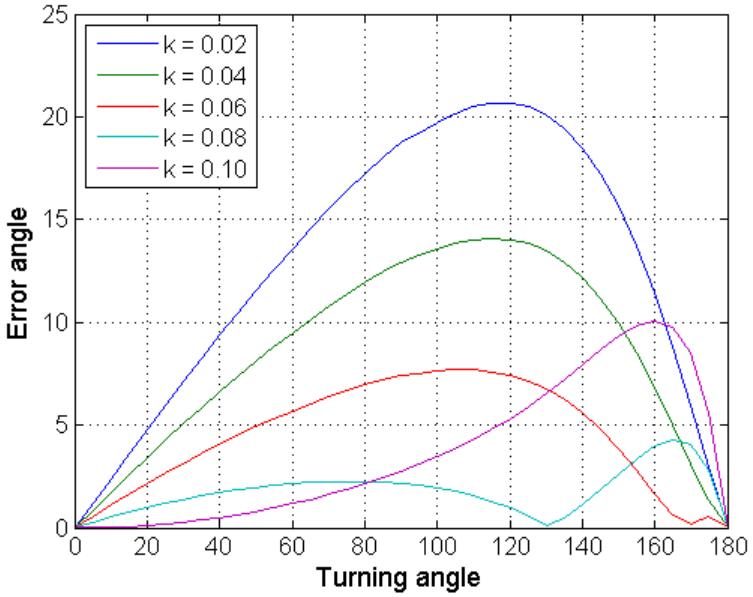

Figure 2: Results for k between 0.02 and 0.10

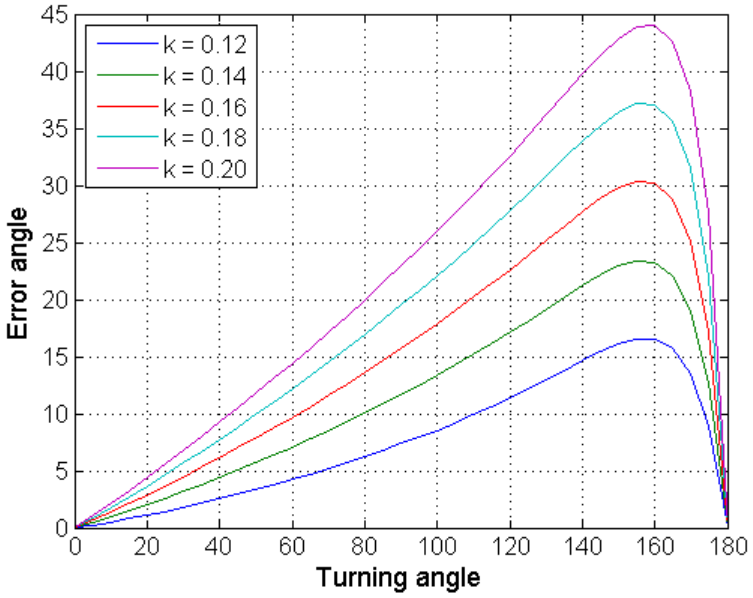

Figure 3: Results for k between 0.12 and 0.20

It can be seen that the curve changes its shape at around $k = 0.08$. This can be explained by the fact that the mean direction φ changes less the smaller the value for k is and vice versa. Thus, the error angle ϵ deviates more for smaller angles α when value k is small (Figure 2). On the other side, for larger values k, the deviation increases for greater angles α (Figure 3).

Compared to the results obtained with real ants in [MW88], one gets the best matching with a value for k at around 0.13. Therefore, the default value for k has been set to 0.13 for every experiment.

Apart from that, as already stated in [MW88], the results match the experimental data astonishing good, and show that the approximative way of computing the homeward course yields relatively large errors although it is being used by ants as an effective way of navigating. This seeming contradiction can be explained by the fact the experiment is set up. Because of the artificial foraging path generated by the channel, the ant is forced to take exactly one right turn. Therefore the path is biased towards the right direction, which generates an error. But surveys of ants' natural foraging paths have shown that the ant turns as often to the right as to the left, so foraging path are most often balanced [MW88]. In a natural situation, the error would minimize. Nevertheless, it still may happen that the ant walks more towards one side. In that case, it would have its other navigational aids as a backup.

This fact answers the question, what would happen if the ant is restricted to only use one of its navigational tools, in this instance, path integration. As it has been shown, the ant does not fully depend on the presence of all navigational tools, but if it is allowed to only use path integration, the navigation is not exact in situations, where the foraging path is biased to the left or to the right.

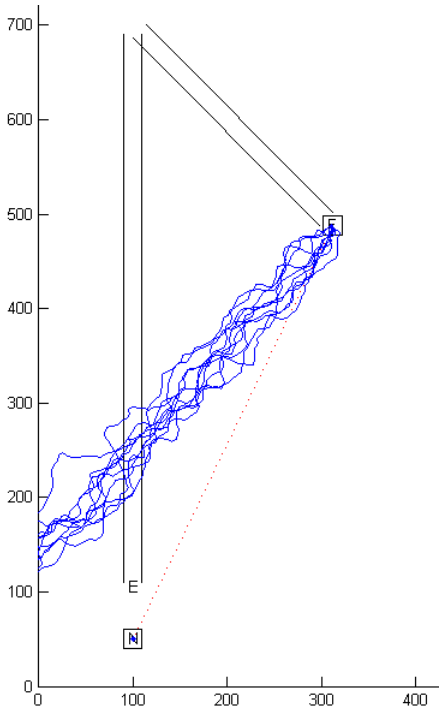

Figure 4: Trails of 10 ants with a = 6m, b = 3m, $\alpha = 135°$ and $k = 0.13$; N: Nest, E: Entrance, F: Feeder; Dotted red line indicates direct connection from feeder to nest

5.1.1 Three-leg trajectory

This experiment is an extension to the two-leg trajectory. In this simulation, the channel consists of three legs. The remaining parameters stay the same.

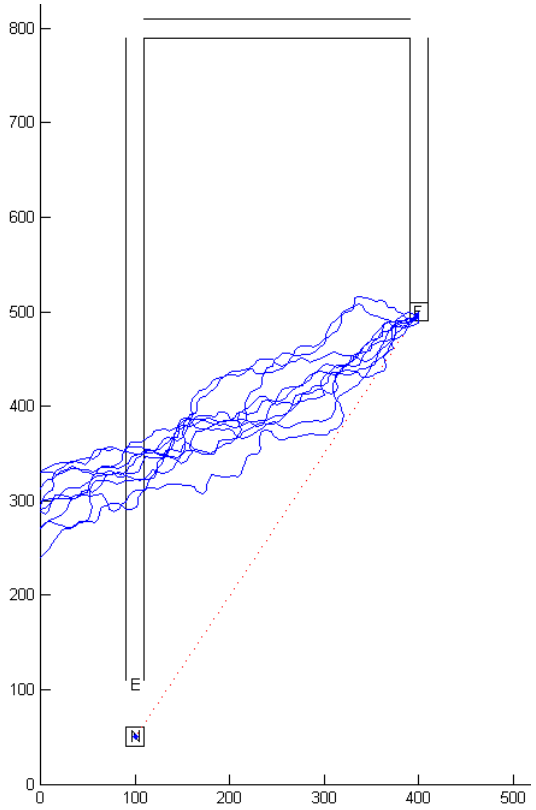

Figure 5: Trails of 10 ants with a = 7m, b = 3m, c = 3m, k = 0.13; N: Nest, E: Entrance, F: Feeder; Dotted red line indicates direct connection from feeder to nest

The result is in accord with the experimental data acquired in [MW88] and thus approves the conclusions made in chapter 5.1.

5.2 One-leg trajectory

In [CCBW98] an experiment is described where desert ants were trained to go to a tube, walk through it to a feeder, walk back through it and then return to their nest. The point of interest was on the returning after leaving the channel. The authors manipulated the channels from the training situation. The experts could witness that most animals at first still walked in the same direction as in the training situation, but then changed the direction towards the nest. They supposed that the desert ants could remember that at the end of the tunnel they had to walk in a certain direction to find their nest. After a certain time they started to follow the global vector.

In this experiment we supposed that at the one end of the channel, the ant puts a landmark which it first follows when heading homewards. After a certain time they rely on their global vector. For this experiment the classes ant, channel and landmark were useful. We defined an area with a nest, a channel and a feeder to simulate the trainings situation. Then the animal would walk to the channel entrance, put a landmark and transit the tunnel. When it reached the feeder, the channel was manipulated and the landmark moved to the entrance of the changed channel. In this new environment the ant traversed the channel again. After following the local landmark it had put before, the desert ant followed the global vector.

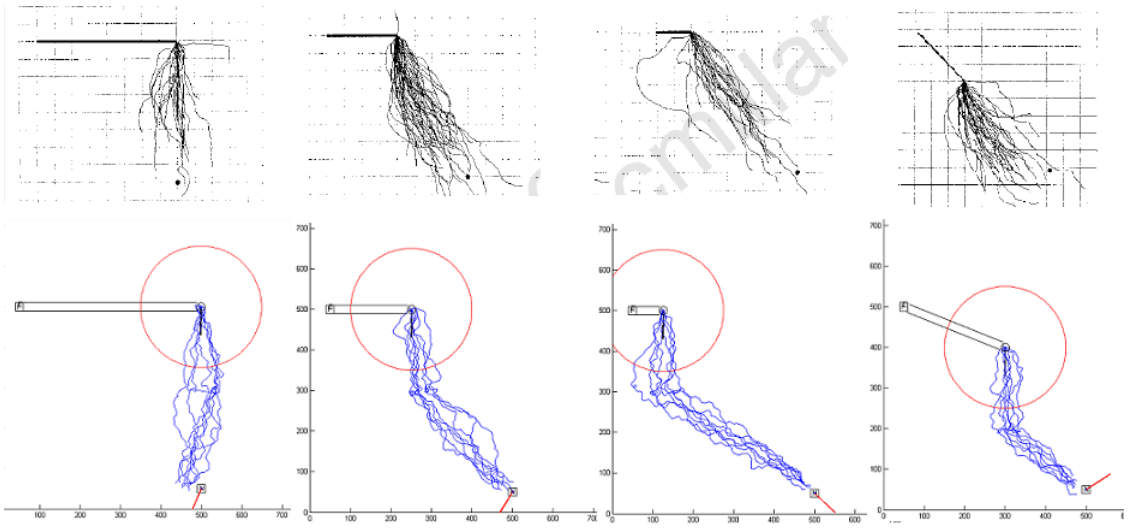

Figure 6: Resulting trails. Upper row taken from [CCBW98]; Fig. 1: Test scenario. Fig. 2: Channel half as long as training channel. Fig. 3: 1/4 of training channel Fig. 4: Rotated channel with half length of training channel. For all figures the parameters k=0.07, landmark.range=150, landmark.length=70 and random_params = $[\pi/4, 0.3]$ were used. Legend: E=Entrance, N=Nest, F=Feeder.

The first pictures show the training situation. As expected all ants find their way back home easily. In the second pictures the channel is half as long as the training channel. At first the ants follow the local vector and then start to follow the global vector. The simulation equates to the experiment results in text [CCBW98]. The simulation results in the third picture are not as similar to the behaviour of real desert ants. It seems that in reality the local vector is not as important as we supposed in our simulation. Still the animals reach the nest. This leads to the conclusion that the implementation for the global vector is quite good. For the last picture the channel got rotated. As in experiment three the insect follows the local vector for too long. But here also the ant is successful looking for the nest.

5.3 Cylinder corridor

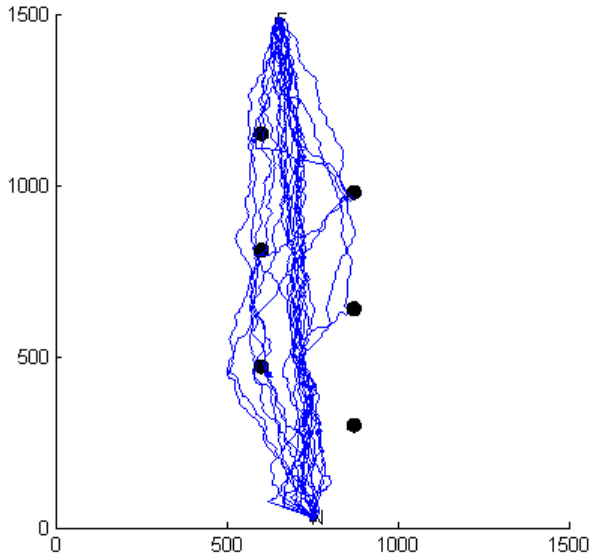

Figure 7: Trails of 10 ants N: Nest F: Feeder; Black circles indicate cylinders which are used as visual landmarks

Figure 7 shows the reproduction of the experiment [CCBW988, Fig.4 a)].
Trained ants were caught at the feeder, and taken to a test area where there was another corridor of cylinders. The Ants were collected at the feeder and released at the corridor entrance.
They walked up the corridor and continued in the same direction for a few metres after passing the last cylinder. In our reproduction the search path of the ants was

more the less predetermined in the programmed code. The ants were ordered by a random factor to approach the cylinders rage, after that the ants return path was influenced by landmarks respectively the local and global vectors.

5.3.1 Crooked corridor

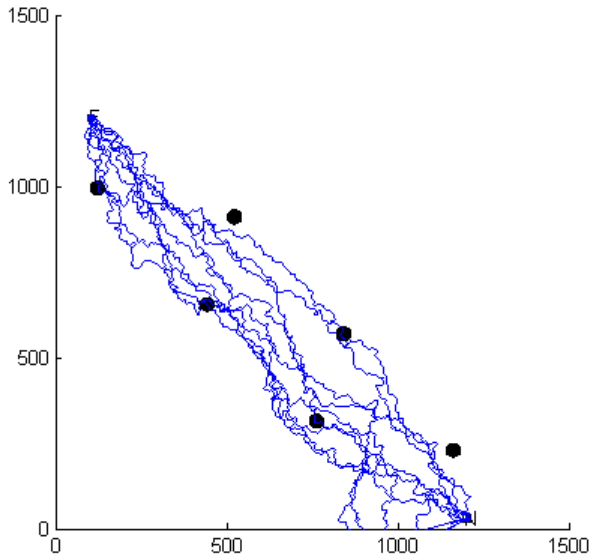

Figure 8: Trails of 10 ants N: Nest F: Feeder; Black circles indicate cylinders which are used as visual landmarks, same experiment as Fig. 7 but with a crooked corridor

A critical test to detect the local vector was to shift the corridor's orientation as shown in Figure 8. Despite being trained on a southwards respectively northwards showing corridor (Figure 7) the ants had no effort finding their nest in the crooked corridor.
Almost all ants walked along the corridor This answers also quite a lot the question about how variations in amount and distribution of the landmarks can affect the foraging efficiency. The landmarks help the ants to take the more direct way through the corridor.

5.4 Moved nest

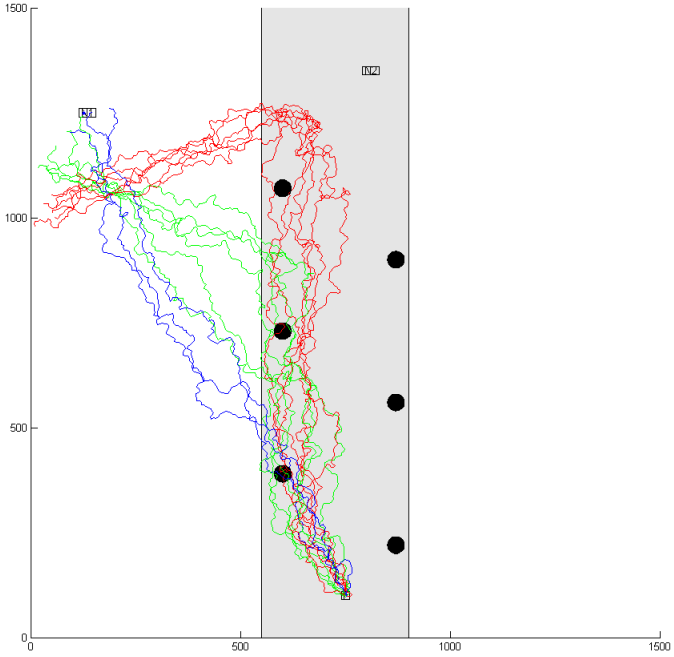

Figure 9: Trails of 15 ants N2: Nest where ants train(through landmark route) N1: New nest where ants get deposited after being catched at the feeder F: Feeder; Black circles indicate cylinders which are used as visual landmarks; Different trail colors: Red: Ants which follow the landmark route, Green: Ants which partly follow the landmarks, Blue: Ants which follow the direct path to the nest

The experiment in [W03, Fig.6] processes one of the most crucial tests to detect a compass-based local vector. The idea was to move the ants nest after they were trained walking down a corridor of black cylinders as shown in Figure 9.
The nest was shifted around 45°.
After their training the local vectors placed at the cylinders were straightened all north. Subsequent the ants were caught at the feeder and released at the moved nest.
After arriving at the feeder about a half of the ants select the global vector rather then the landmark. The other interesting point is that the ants which followed the landmark route continue to update their path integrator and, upon leaving the landmark route, choose the course leading directly to the nest. We see the clarification of the interaction between the two different navigational tools is still not taken for granted and probably needs more investigation.

5.5 Natural foraging

In our last experiment, we tested both navigational tools, path integration and landmark based navigation on an open area without constraints such as channels or fixed positions for landmarks. This setup comes closest to what we think is a natural situation, where all navigational aids are collaborating and thus guiding the ant throughout its trip in a best possible way.

The ant starts in the center of a $10m^2$ area. In every run, a feeding site is put at a random place. The ant picks random points on the area and moves to them. As soon as it gets near to the feeder, it walks towards its center to collect food, and then moves back to the nest, guided by the global vector. During the ants' trip it sets landmarks after certain distances specified by an input argument, which additionally help the ant when moving back to the nest.

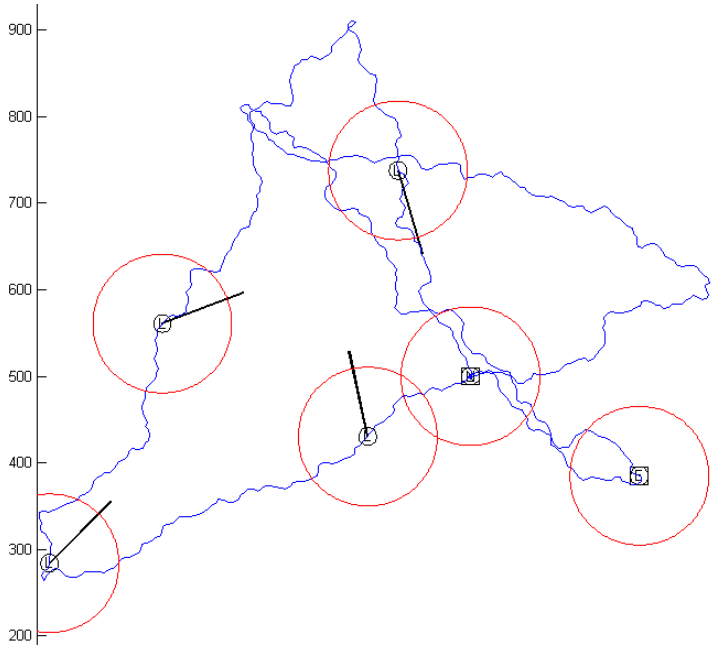

Figure 10: Example of a single foraging path with *lm_distance* = 70; N: Nest, F: Feeder, L: Landmarks with range as red circles and local vectors as black lines.

In this experiment, it happens often that the ant ends up in an infinite loop where the global and the local vectors are opposing to each other. This case occurs, when the homing ant follows its global vector and reaches a landmark which it had set previously. If the local vector points to the direction the ant came from, the local vector leads the ant back for a certain distance, whereupon it starts to follow the

global vector again to the same landmark with the opposing local vector, and so on.

The situation where a local vector points against the global vector arises due to inaccuracies of the ants' approximative path integration. This can be avoided by setting the normalization constant k to a lower value. As shown in chapter 5.1 on page 12, the error angle minimizes at $k = 0.08$. If this value is applied to the foraging ant, it does not end up in loops, as the global vector barely deviates from the actual direction to the nest, which results in local vectors that always point to the nest. Although this situation does not represent a natural situation, it allows to analyse the influence of different amounts and distributions of landmarks on homing ants.
Below are the results of the experiment run with 100 ants and different distributions of landmarks. Parameter *lm_distance* determines the interval, after which the ant puts landmarks on its path.

lm_distance	% of ants that reached nest
50	90%
60	81%
70	82%
80	79%
90	77%
100	73%
110	86%
120	80%
130	84%
140	83%

As it can be seen, the amount of ants that reached the nest is high for low values for *lm_distance*, which is not surprising since this means that the ant simply puts more landmarks on the area.

More interesting is the trend when *lm_distance* rises. For larger values, e.g. 140, the amount of ants that reached the nest is higher than with the smaller value 100 (which would mean more landmarks). This deviation may be explained by the random distribution of the landmarks. The results might simply be random and do not follow a certain rule, as the output of the experiment depends on a variety of parameters, such as the range of landmarks or their local vectors' length. Even the way random walk is implemented affects the result. Therefore, in this case, it can only be concluded that an increase in the amount of landmarks raises the rate of successful returning ants.

6 Summary and Outlook

6.1 Summary

In terms of path integration, we were able to reconstruct the results of [MW88] as can be seen from Experiment 5.1 (Two-leg trajectory). We argued that if the ant completely relies on path integration, its navigation contains error.

As for landmark based navigation, we acquired similar results for the remaining experiments with our implementation of local vectors. Problems we had to face were opposite global and local vectors which resulted in loops, from which the ant was unable to get out. Hence we concluded that our model of landmark based navigation combined with path integration cannot describe the ants' natural navigational behaviour thoroughly. Certainly our implementation of landmark based navigation has its weaknesses, but even if we would use a revised model where the loop-problem does not occur, it would be incomplete because of the fact that there are certainly more aspects took into consideration by the ant when navigating, such as using polarization patterns of skylight.

6.2 Outlook

Since the research of foraging behaviour of ants - especially desert ants - is still going on, we had little information about the foraging behaviour of the individuals. That is why more research is necessary to provide more data about this topic. An interesting point is the orchestration of the different navigation tools.

In our experiments there was almost always one ant foraging and finding a way back to the nest. The next step would be to let several ants search for food simultaneously. As an ant successfully finds the feeder, it would inform the others. In reality, antennas are used for communication. In the observed experiments only one feeder was used. What happens if more feeders were available?

Another interesting fact is that ants use odour to signal where they went. This is the case for ants in forests and humid environments but not for those in desert. That is the reason why it would also be interesting to compare the foraging strategies of different kinds of ants. While desert ants mainly use their global vector, ants in forests probably use more often landmarks and odour to navigate.

7 References

[MW88] MUELLER, Martin; WEHNER, Ruediger: *Path integration in desert ants, Cataglyphis fortis* in *Proc. Natl. Acad. Sci. USA, Vol. 85, pp. 5287-5290, July 1988*

[CCBW98] COLLETT M.; COLLETT T.S.; BISCH S.; WEHNER R.: *Local and global vectors in desert ant navigation* in *Nature, Vol 394, 16 July 1998*

[W03] WEHNER Ruediger: *Desert ant navigation: how miniature brains solve complex tasks* in *J Comp Physiol A (2003) 189: 579588*

A MATLAB Code

A.1 main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Main script     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% Contains experiments
% Entry point for program execution

clear;
path('experiments\',path);
% Set random seed
RandStream.setDefaultStream(RandStream('mt19937ar', 'seed', 56355));



%%%%%% Experiments %%%%%%

% Two-leg trajectory
alphas = 0:5:180;
k = [0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20];
epsilon_two_leg = two_leg_trajectory (600, 300, alphas, k, 1)
csvwrite('two-leg trajectory.csv', epsilon_two_leg);

% Three-leg trajectory
epsilon_three_leg = three_leg_trajectory(700, 300, 300, 10)
csvwrite('three-leg trajectory.csv', epsilon_three_leg);


% One-leg trajectory with default channel
channel = Channel(421,500,599,600);
one_leg_trajectory(channel,3);

% Crooked corridor
crooked_corridor(10)

% Corridor of black cylinders
corridor_of_black_cylinders(1);

% Moved nest
moved_nest(10)

% Natural foraging
natural_foraging(10, 80)
```

A.2 Ant.m

```
classdef Ant < handle
    %ANT class
    %   Describes the entity ant.
    properties
        % Basic
        status = 0;                 % Movement status (either 0 for ...
            foraging or 1 for returning or 2 for random search)
        pos;                        % Position on the area

        % Movement
        speed = 3;                  % Movement speed
        global_v = [0, 0];          % Global vector
        local_v = [0, 0];           % Local vector
        mean_v = [0, 0];            % Mean vector
        target = [0, 0];            % Target (used for directed and ...
            local vector movement)
        random_params = [0, 1];     % Angle (in radian) and drift factor ...
            for random walk
        ang = 0;                    % Current moving angle
        phi = 'null';               % Mean angle
        l = 0;                      % Mean distance
        k = 0.13;                   % Normalization constant for ...
            deriving the mean angle

        % Other
        show_trail = 0;             % Whether the ants trail will be plotted
        landmarks;                  % List of landmarks
        trail = [0, 0];             % Walking trail
        trails = cell(1);           % Container with all trails
        trail_color = 'blue';       % Color of the trails

        im = imread('ant.png');     % Image to plot
    end

    methods
        % Construction & Basic Operations
        function obj = Ant(pos)
            % Creates an ant at the specified position
            obj.pos = pos;          % Puts the ant to the given position
            obj.trail = pos;        % Sets the starting point of the ...
                trail to the ants' initial position
            obj.trails{1} = pos;    % Empty (dummy) trail
        end

        function reset(obj, pos)
            % Resets the ant's attributes
```

```
42              obj.status = 0;          % Set status to 'foraging'
43              obj.reset_global_v;      % Reset the global vector
44              obj.target = [0, 0];     % Reset current target
45              obj.pos = pos;           % Put the ant back to 'pos'
46          end
47          % Moving
48          function arrived = move_to(obj, target)
49              % Lets the ant move to the point specified by target
50              % Returns 1 if the ant has arrived at the point
51
52              arrived = 0;
53              dist = target - obj.pos;       % Distance vector between the ...
                    current
54                                             % position and the target
55              e_dist = norm(dist);           % Euclidean distance
56
57              if e_dist >= obj.speed         % If the distance is bigger ...
                    than the ants' step
58                                             % (i.e. the ant has not ...
                                                   arrived yet
59                  % Move the ant
60                  old_ang = obj.ang;                          % Save old angle
61                  obj.ang = angle(dist(1) + dist(2)*1i);      % Compute new ...
                        angle
62                  delta = mod(obj.ang - old_ang, 2*pi);       % Compute ...
                        current turning angle
63                  if delta > pi                               % and adjust ...
                        the range
64                      delta = delta -2*pi;                    % to -pi <= ...
                            delta <= pi
65                  end
66                  % Compute a random angle between -alpha <= ran <= alpha,
67                  % where alpha = random_params(1) is the random movement
68                  % angle
69                  ran = 2*obj.random_params(1)*rand-obj.random_params(1);
70                  % Compute the drift for random walk with random_params(2)
71                  drift = obj.random_params(2)*delta;
72                  % Apply the random values to the ants' old angle
73                  obj.ang = old_ang + ran + drift;
74
75                  % Compute the move vector for this step
76                  move = [cos(obj.ang)*obj.speed, sin(obj.ang)*obj.speed];
77                  % Advance position by the move vector
78                  obj.pos = obj.pos + move;
79
80                  obj.update_global_v();          % Update global vector
81                  obj.update_local_v();           % Update local vector
82
83                  if obj.show_trail
84                      % Builds up the trail behind the ant
```

```
85                      n = size(obj.trail);
86                      if norm(obj.trail(n(1),:)-obj.pos) > 5
87                          % Extend the trail with a new point
88                          obj.trail = [obj.trail; obj.pos];
89                      end
90                  else
91                      if length(obj.trail) > 2
92                          % Save the trail
93                          obj.trails{length(obj.trails)+1} = obj.trail;
94                      end
95                      obj.trail = obj.pos;
96                  end
97              else
98                  arrived = 1;
99              end
100         end
101
102         % Landmarks
103         function put_landmark(obj)
104             % Puts a landmark at the current position with the current
105             % global vector as local vector
106             obj.landmarks = [obj.landmarks, Landmark(obj.pos(1), ...
                    obj.pos(2), obj.global_v(1), obj.global_v(2))];
107         end
108
109         function put_landmark_at(obj, place)
110             % Put a landmark at the given position with the current
111             % global vector as local vector
112             obj.landmarks = [obj.landmarks, Landmark(place(1), place(2), ...
                    obj.global_v(1), obj.global_v(2))];
113         end
114
115         function del_landmark(obj, i)
116             % Removes landmark with index i
117             obj.landmarks = [obj.landmarks(1:i-1), ...
                    obj.landmarks(i+1:length(obj.landmarks))];
118         end
119         function landmark = within_landmark(obj)
120             % Returns a list of landmarks in whose range the ant is
121             n = length(obj.landmarks);
122             landmark = []; % Default case - the ant is not within a landmark
123             for i=n:-1:1
124                 % Distance between the ant and landmark i
125                 dist = sqrt( (obj.pos(1)-obj.landmarks(i).pos(1))^2 + ...
                        ((obj.pos(2)-obj.landmarks(i).pos(2)))^2 );
126                 if dist < obj.landmarks(i).range
127                     % If the ant lies within range, extend the list of
128                     % landmarks
129                     landmark = [landmark, obj.landmarks(i)];
130                 end
```

```
131             end
132         end
133         % Vectors
134         function update_global_v(obj)
135             % Update the ant's global vector
136             if strcmp(obj.phi, 'null')                  % Check for ...
                    default value
137                 obj.phi = obj.ang;
138             end
139             delta = mod(obj.ang - obj.phi, 2*pi);       % Compute angle delta
140             if delta > pi                               % Scale range to ...
                    -pi<=delta<=pi
141                 delta = delta -2*pi;
142             end
143             % Compute covered distance for the current step
144             obj.l = obj.l + (1 - abs(delta)/(pi/2));    % Mean distance
145             % Compute mean direction for the current step
146             % Version 1
147             %obj.phi = (obj.l*obj.phi + obj.phi + delta)/(obj.l + 1);
148             % Version 2
149             obj.phi = mod(obj.phi,2*pi) + obj.k * ...
                    ((pi+delta)*(pi-delta)*delta)/obj.l;
150
151             if obj.phi > pi                             % Scale range to ...
                    -pi<=phi<=pi
152                 obj.phi = obj.phi - 2*pi;
153             end
154             % Compute global vector
155             obj.global_v = [70*cos(obj.phi + pi), 70*sin(obj.phi + pi)];
156             % Compute mean vector
157             obj.mean_v = [70*cos(obj.phi), 70*sin(obj.phi)];
158         end
159         function reset_global_v(obj)
160             % Resets the global vector and its dependent properties
161             obj.global_v = [0,0];
162             obj.ang = 0;
163             obj.phi = 'null';
164             obj.l = 0;
165         end
166         function arrived = follow_global_v(obj)
167             % Lets the ant follow the global vector
168             obj.show_trail = 1;         % Enables the trail
169             arrived = 1;                % Default case
170             if obj.l > 5
171                 % If there's still a distance to travel,
172                 % keep following the global vector
173                 obj.move_to(obj.pos+obj.global_v);
174                 arrived = 0;            % The ant is still on its way
175             end
176         end
```

```
177         function update_local_v(obj)
178             % Keeps the local vector associated with the last visited
179             % landmark
180             % Get all landmarks the ant walks on
181             current_landmark = obj.within_landmark();
182             if length(current_landmark) > 1
183                 % If there are multiple landmarks, pick the first
184                 current_landmark = current_landmark(1);
185             end
186             if current_landmark ~= 0
187                 % Get the local vector associated with the current landmark
188                 obj.local_v = current_landmark.local_v;
189             else
190                 % The ant lost its local vector
191                 obj.local_v = [0, 0];
192             end
193         end
194         function arrived = follow_local_v(obj)
195             % Lets the ant follow the local vector (if there is any)
196             obj.show_trail = 1;     % Enables trail
197             if obj.within_landmark ~= 0
198                 % As long as the ant lies within the landmark, it keeps
199                 % the local vector and updates the place it points to
200                 obj.target = obj.pos + obj.local_v;
201             end
202             arrived = 1;
203             if obj.target ~= 0
204                 % The local vector is not processed yet
205                 arrived = 0;
206                 % Plot local vector
207                 plot([obj.pos(1),obj.target(1)], [obj.pos(2), ...
                        obj.target(2)], 'magenta', 'LineWidth', 2);
208
209                 if obj.move_to(obj.target)
210                     % The ant arrived at the point the local vector points
211                     % to and resets this point (the target)
212                     arrived = 1;
213                     obj.target = [0, 0];
214                 end
215             end
216         end
217
218         % Other
219         function plot(obj)
220             % Plot the ant as a point
221             plot(obj.pos(1), obj.pos(2), '.');
222
223             % Plot landmarks
224             for i=1:length(obj.landmarks)
225                 obj.landmarks(i).plot();
```

```
226             end
227             % Plot mean vector
228             % plot([obj.pos(1),obj.pos(1)+70*cos(obj.phi)],[obj.pos(2), ...
                    obj.pos(2)+70*sin(obj.phi)], 'red');
229             % Plot global vector
230             plot([obj.pos(1),obj.pos(1)+obj.global_v(1)],[obj.pos(2), ...
                    obj.pos(2)+obj.global_v(2)],'red', 'LineWidth', 2);
231
232             % Plot current trail
233             plot(obj.trail(:,1), obj.trail(:,2), obj.trail_color);
234             % Plot previous trails
235             for i=1:length(obj.trails)
236                 plot(obj.trails{i}(:,1), obj.trails{i}(:,2), ...
                        obj.trail_color);
237             end
238             obj.show_trail = 0;
239         end
240     end
241
242 end
```

A.3 Landmark.m

```
1  classdef Landmark < handle
2      %LANDMARK class
3      %   Describes abstract landmarks set by the foraging ant
4
5      properties
6          pos;            % Position
7          range = 80;     % Covered range
8          ang;            % Angle in which direction the local vector ...
                points to
9          local_v;        % Local vector stored at the landmark
10         length = 100;   % Length of local vector
11         color = 'black';% Color of the Landmark
12     end
13
14     methods
15         function obj = Landmark(varargin)
16             % Constructor
17             % Supports creation from both an angle and vector components
18             if nargin == 3
19                 % Create with angle, e.g.: landmark = Landmark(x, y, phi);
20                 obj.pos = [varargin{1}, varargin{2}];
21                 obj.ang = varargin{3};
22                 obj.local_v = [obj.length*cos(obj.ang), ...
                        obj.length*sin(obj.ang)];
```

```
23              elseif nargin == 4
24                  % Create with components, e.g.: landmark = Landmark(x, ...
                        y, u, v);
25                  obj.pos = [varargin{1}, varargin{2}];
26                  obj.local_v = [obj.length*varargin{3}/70, ...
                        obj.length*varargin{4}/70];
27                  obj.ang = angle(varargin{3} + varargin{4}*1i);
28              end
29          end
30
31          function plot(obj)
32              % Plots the Landmark with its corresponding local vector
33              r = 10; % Radius of the inner circle
34              % Plot small circle around 'L'
35              rectangle('Position',[obj.pos(1)-r, obj.pos(2)-r, 2*r, 2*r], ...
                    'Curvature', [1,1]);
36              % Plot range as circle
37              rectangle('Position',[obj.pos(1)-obj.range, ...
                    obj.pos(2)-obj.range, 2*obj.range, 2*obj.range], ...
                    'Curvature', [1,1], 'EdgeColor', 'r');
38              % Plot local vector
39              plot([obj.pos(1), obj.pos(1)+obj.length*cos(obj.ang)], ...
                    [obj.pos(2), obj.pos(2)+obj.length*sin(obj.ang)], ...
                    obj.color, 'LineWidth', 2);
40              % Plot 'L'
41              text(obj.pos(1)-5, obj.pos(2), 'L', 'Color', obj.color);
42          end
43      end
44
45  end
```

A.4 Area.m

```
1   classdef Area < handle
2       %AREA class
3       %   Describes the testing area and its basic elements
4
5       properties
6           size;
7           feeder = [0, 0];
8           nest = [0, 0];
9       end
10
11      methods
12          function obj = Area(size_x, size_y)
13              obj.size = [size_x, size_y];
14          end
```

```
15
16          function plot_nest(obj)
17              text(obj.nest(1)-5, obj.nest(2), 'N');
18              rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,20,20]);
19          end
20
21          function plot_feeder(obj)
22              text(obj.feeder(1)-5, obj.feeder(2), 'F');
23              rectangle('Position',[obj.feeder(1)-10,obj.feeder(2)-10,20,20]);
24          end
25      end
26
27  end
```

A.5 Cylinder_Area.m

```
1   classdef Cylinder_Area < handle
2   % Describes the area where all the experiments are held which include black
3   % cylinders as visual landmarks.
4
5       properties
6           amount;             % amount of cylinders
7           cylinders;          % 2x6 Matrix, containing all the X,Y ...
                coordinates of the cylinders
8           landmark_route;     % Displays the Training route
9           size = [0,0];       % Area size
10          feeder = [0, 0];    % Feeder position
11          nest = [0, 0];      % Nest position
12          nest_two = [0 0];   % Second Nest position ( used for ...
                "moved_nest" experiment.
13      end
14
15      methods
16
17          % Creation procedure of the Cylinder Area
18          function obj = Cylinder_Area(length_x, length_y, ...
                start_x,start_y,n_of_cylinders)
19              xlim([0, length_x]); % Length of the area in X direction
20              ylim([0, length_y]); % Length of the area in Y direction
21              obj.size = [length_x, length_y];
22              obj.cylinders = zeros(n_of_cylinders,2);
23              obj.amount = n_of_cylinders;% amount of cylinders on the area
24
25
26              % Sets the feeder position
27              obj.feeder(1) = start_x-200;
28              obj.feeder(2) = start_y + obj.amount*200;
```

```
29
30              % Sets the nest position
31              obj.nest(1) = start_x - 85;
32              obj.nest(2) = start_y - 50;
33
34              obj.cylinders(1,:) = [start_x start_y]; % Stores position of ...
                    cylinder 1
35
36              k = 2; % Starts storing at Cylinder 2 ( 1st one is stored ...
                    allready
37
38              % Places all Cylinders on the Map
39              for i = 1:obj.amount
40                  start_y = start_y + 170;
41              % Alternaterly left or right side of the corridor
42                  if mod(i,2) == 0
43                      start_x = start_x + 270;
44                  else
45                      start_x = start_x - 270;
46                  end
47
48                  % Stores The coordinates of the Cylinders
49                  if k  <= obj.amount
50                      obj.cylinders(k,:) = [start_x start_y];
51                      k = k+1;
52                   end
53
54                  hold on
55                  axis square;
56              end
57          end
58
59          % Plots all the Cylinders into the map
60          function plot_cylinders(obj)
61              for i = 1: obj.amount
62                  rectangle('Position',[obj.cylinders(i,:),40,40], ...
                        'Curvature', [1,1], 'FaceColor', 'k');
63              end
64          end
65
66          %Plots the training/landmark route (moved_nest experiment)
67          function plot_landmark_route(obj)
68             obj.landmark_route = rectangle('Position',[550,0,350,1500], ...
                  'Curvature', [0,0], 'FaceColor', [0.9, 0.9, 0.9]);
69          end
70
71          % Plots two nests that are used for "moved_nest" experiment
72          function plot_new_nest(obj,n)
73              if n == 1
74                  % plots nest two 'N2'
```

```
75                  text(obj.nest_two(1)-5, obj.nest_two(2), 'N2');
76                  rectangle('Position',[obj.nest_two(1)-10,obj.nest_two(2)-10,40,20]);
77              elseif n == 2
78                  % plots nest one 'N1'
79                  obj.nest_two = [obj.nest(1)-5 obj.nest(2)];
80                  text(obj.nest(1)-5, obj.nest(2), 'N1');
81                  rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,40,20]);
82              end
83          end
84
85          % Plots the regular nest
86          function plot_nest(obj)
87              text(obj.nest(1)-5, obj.nest(2), 'N');
88              rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,20,20]);
89          end
90
91          % plots the feeder
92          function plot_feeder(obj)
93              text(obj.feeder(1)-5, obj.feeder(2), 'F');
94              rectangle('Position',[obj.feeder(1)-10,obj.feeder(2)-10,20,20]);
95          end
96      end
97
98  end
```

A.6 Channel.m

```
1   classdef Channel < handle
2       %CHANNEL Class
3       %   Describes an n-leg channel
4
5       properties
6           n_of_legs;      % Number of legs
7           entrance;       % Entrance coordinates
8           exit;           % Exit coordinates
9           nodes;          % Nodes of the channel
10      end
11
12      methods
13
14          function obj = Channel(varargin)
15              % Constructor
16              % Provides inputs from 1 leg up to 3 legs Inputs are nodes
17              if nargin == 4
18                  % Creates Channel with 2 nodes, e.g.: channel = ...
                        Channel(x1, y1, x2, y2);
19                  obj.n_of_legs = 1;
```

```
20                  obj.entrance = [varargin{1}, varargin{2}];
21                  obj.exit = [varargin{3}, varargin{4}];
22                  obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                        varargin{4}];
23              elseif nargin == 6
24                  % Creates Channel with 3 nodes, e.g.: channel = ...
                        Channel(x1, y1, x2, y2, x3, y3);
25                  obj.n_of_legs = 2;
26                  obj.entrance = [varargin{1}, varargin{2}];
27                  obj.exit = [varargin{5}, varargin{6}];
28                  obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                        varargin{4}, varargin{5}, varargin{6}];
29              elseif nargin == 8
30                  % Creates Channel with 4 nodes, e.g.: channel = ...
                        Channel(x1, y1, x2, y2, x3, y3, x4, y4);
31                  obj.n_of_legs = 3;
32                  obj.entrance = [varargin{1}, varargin{2}];
33                  obj.exit = [varargin{7}, varargin{8}];
34                  obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                        varargin{4}, varargin{5}, varargin{6}, varargin{7}, ...
                        varargin{8}];
35              end
36          end
37
38          function plot(obj)
39              % Plots the channel
40              for i=1:2:2*obj.n_of_legs
41
42                  % Width of channel
43                  w = 10;
44                  % Direction vector
45                  v = [obj.nodes(i+2)-obj.nodes(i), ...
                        obj.nodes(i+3)-obj.nodes(i+1)];
46                  % Direction vector with length w
47                  vn = ...
                        w*(v/sqrt((obj.nodes(i+2)-obj.nodes(i))^2+(obj.nodes(i+3)-obj.nodes(i+1))^
48                  % Orthogonal vector to direction vector
49                  vr = ...
                        [-(obj.nodes(i+3)-obj.nodes(i+1)),obj.nodes(i+2)-obj.nodes(i)];
50                  % Orthogonal vector to direction vector with lenght w
51                  vrn = ...
                        w*(vr/sqrt((-obj.nodes(i+2)+obj.nodes(i))^2+(obj.nodes(i+3)-obj.nodes(i+1)
52
53                  % Wall points
54                  wp11 = [obj.nodes(i)+vn(1)+vrn(1), ...
                        obj.nodes(i+1)+vn(2)+vrn(2)];    % wall point 1.1
55                  wp12 = [obj.nodes(i+2)-vn(1)+vrn(1), ...
                        obj.nodes(i+3)-vn(2)+vrn(2)];  % wall point 1.2
56                  wp21 = [obj.nodes(i)+vn(1)-vrn(1), ...
                        obj.nodes(i+1)+vn(2)-vrn(2)];    % wall point 2.1
```

```
57                  wp22 = [obj.nodes(i+2)-vn(1)-vrn(1), ...
                        obj.nodes(i+3)-vn(2)-vrn(2)];  % wall point 2.2
58
59                  % Connect adjacent wall points
60                  line([wp11(1),wp12(1)],[wp11(2),wp12(2)], 'Color', [0,0,0]);
61                  line([wp21(1),wp22(1)],[wp21(2),wp22(2)], 'Color', [0,0,0]);
62
63              end
64              text(obj.nodes(1)-5,obj.nodes(2)+5,'E');
65          end
66      end
67  end
```

A.7 two_leg_trajectory.m

```
1  function epsilon = two_leg_trajectory( a, b, alpha, k, n_of_ants )
2  %TWO_LEG_TRAJECTORY
3  % Describes an experiment where the ant is forced to walk through a
4  % two-legged channel. At the end of the channel, it finds food and starts
5  % to head back for its nest. Due to the approximative way the ant keeps
6  % computing its mean direction (and thus the direction back to the nest),
7  % errors occur. These errors can be pointed out by manipulating the angle
8  % between the two elements of the channel.
9
10 % Output:
11 % epsilon: returns the error angle produced by the ant.
12 % Inputs:
13 % a, b: Lengths of the channel-legs.
14 % alpha: Angle between the two legs.
15 % k: Normalization constant used in the approximative computation.
16 % n_of_ants: Amount of ants that run the experiment with the given inputs.
17
18
19     %%%%%%%%%% Initialization %%%%%%%%%%
20     % Environment
21     area = Area(1000,1000); % Create an area with size 10m^2
22     area.nest = [100, 50];  % Place the nest at Position (100, 50)
23
24     % Output
25     epsilon = zeros(length(k),length(alpha),n_of_ants);
26
27     % Clear figure screen and hold the graphics
28     clf; hold on;
29     axis equal;
30     xlim([0, area.size(1)]);
31     ylim([0, area.size(2)]);
32
```

```
33      %%%%%%%%%% Main loops %%%%%%%%%%%%%%
34      for i=1:length(k) % Loops through the specified constants k
35          for j=1:length(alpha) % Loops through the specified angles alpha
36              % Create a new ant and place it to the nest
37              ant = Ant(area.nest);
38
39              % Channel
40              rad = alpha(j)*(pi/180);    % Convert angle to radian
41              % Create a channel with the given parameters
42              channel = Channel(100, 100, 100, 100+a, ...
                    100+cos(rad-(pi/2))*b, (100+a)-sin(rad-(pi/2))*b);
43              % Place the feeder at the channel exit
44              area.feeder = channel.exit;
45
46              for ants=1:n_of_ants % Loops through the specified amount of ...
                    ants
47                  % Set the constant k according to the input
48                  ant.k = k(i);
49                  % Temporary variables
50                  target = channel.entrance;
51                  done = 0;
52                  while ~done
53                      cla; % Clear axes
54
55                      % Phase 1: Foraging
56                      if ant.status == 0
57                          % Ant moves to target
58                          if ant.move_to(target)
59                              % Ant arrived at the target, set new target
60                              if target == channel.entrance;
61                                  target = channel.nodes(3:4);
62                              elseif target == channel.nodes(3:4)
63                                  target = channel.exit;
64                              elseif target == channel.exit
65                                  % Ant arrived at the feeder
66                                  ant.status = 1;
67                                  % Compute epsilon (error angle)
68                                  eps_correct = angle( ...
                                        (area.nest(1)-ant.pos(1)) + ...
                                        (area.nest(2)-ant.pos(2))*1i ); % ...
                                        Compute correct angle directed to ...
                                        the nest
69                                  epsilon(i, j, ants) = ...
                                        abs((ant.phi-pi)-eps_correct); % ...
                                        Compute difference to the ant's ...
                                        global vector
70                                  epsilon(i, j, ants) = epsilon(i, j, ...
                                        ants)*(180/pi); % Convert to degrees
71                              end
72                          end
```

```
                    % Phase 2: Returning to the nest
                    elseif ant.status == 1
                        % Set up parameters for random walk
                        ant.random_params = [pi/4, 0.3];
                        % Ant follows its global vector back to the nest
                        if ant.follow_global_v()
                            % Ant covered up its calculated distance and
                            % arrived at the nest (best case)
                            % Put the ant back to the nest to start the
                            % next run
                            ant.reset(area.nest);
                            done = 1;
                        end
                    end

                    % Plot
                    area.plot_nest();
                    area.plot_feeder();
                    channel.plot();
                    line([channel.exit(1), area.nest(1)], ...
                        [channel.exit(2), area.nest(2)], 'LineStyle', ...
                        ':', 'Color', [1,0,0]);
                    ant.plot();

                    % Debugging
                    text(10, area.size(2)-60, strcat('\phi = ', ...
                        int2str(ant.phi*180/pi), ' '));
                    text(10, area.size(2)-100, strcat('l = ', ...
                        int2str(ant.l)));

                    pause(0.001);
                end % While loop
            end % Ant loop
        end % Alpha loop
    end % k loop
end
```

A.8 three_leg_trajectory.m

```
function epsilon = three_leg_trajectory( a, b, c, n_of_ants )
%THREE_LEG_TRAJECTORY
% Same setup as two-leg trajectory but with three legs.

% Output:
% epsilon: returns the error angle produced by the ant.
% Inputs:
% a, b, c: Lenghts of the channel legs
```

```
% n_of_ants: Amount of ants that run the experiment with the given inputs.

    % Environment
    area = Area(1000,1000); % Create an area with size 10m^2
    area.nest = [100, 50];

    % Ants
    ant = Ant(area.nest);

    % Channel
    channel = Channel(100, 100, 100, 100+a, 100+b, 100+a, 100+b, (100+a)-c);
    area.feeder = channel.exit;

    % Temporary variables
    target = channel.entrance;

    % Output
    epsilon = zeros(1,n_of_ants);

    %%%%%%%%%% Initialization %%%%%%%%%%
    % Clear figure screen and hold the graphics
    clf; hold on;
    axis equal;
    xlim([0, area.size(1)]);
    ylim([0, area.size(2)]);

    %%%%%%%%%% Main loop %%%%%%%%%%%%%%%
    for ants=1:n_of_ants
        done = 0;   % Status of the current run
        while ~done
            tic; % Start time measurement
            cla; % Clear axes
            % Phase 1: Foraging
            if ant.status == 0
                % Ant moves to target
                if ant.move_to(target)
                    % Ant arrived at the target, set new target
                    if target == channel.entrance;
                        target = channel.nodes(3:4);
                    elseif target == channel.nodes(3:4)
                        target = channel.nodes(5:6);
                    elseif target == channel.nodes(5:6)
                        target = channel.exit;
                    elseif target == channel.exit
                        % Ant arrived at the feeder
                        ant.status = 1;
                        % Compute epsilon (error angle)
                        eps_correct = angle( (area.nest(1)-ant.pos(1)) + ...
                            (area.nest(2)-ant.pos(2))*1i ); % Compute ...
                            correct angle directed to the nest
```

```
57                          epsilon(ants) = abs((ant.phi-pi)-eps_correct); % ...
                                Compute difference to the ant's global vector
58                          epsilon(ants) = epsilon(ants)*(180/pi); % ...
                                Convert to degrees
59                      end
60                  end
61              % Phase 2: Returning to the nest
62              elseif ant.status == 1
63                  % Set up parameters for random walk
64                  ant.random_params = [pi/4, 0.3];
65                  % Ant follows its global vector back to the nest
66                  if ant.follow_global_v()
67                      % Ant covered up its calculated distance and
68                      % arrived at the nest (best case)
69                      % Put the ant back to the nest to start the
70                      % next run
71                      ant.reset(area.nest);
72                      target = channel.entrance;
73                      done = 1;
74                  end
75              end
76
77              % Plot
78              area.plot_nest();
79              area.plot_feeder();
80              channel.plot();
81              line([channel.exit(1), area.nest(1)], [channel.exit(2), ...
                    area.nest(2)], 'LineStyle', ':', 'Color', [1,0,0]);
82              ant.plot();
83
84              % Debugging
85              text(10, area.size(2)-60, strcat('\phi = ', ...
                    int2str(ant.phi*180/pi), ' '));
86              text(10, area.size(2)-100, strcat('l = ', int2str(ant.l)));
87
88
89              pause(0.001);
90          end
91      end
92  end
```

A.9 one_leg_trajectory.m

```
1  function epsilon = one_leg_trajectory( channel, n_of_ants )
2      %ONE_LEG_TRAJECTORY Experiment with one-legged channel and ...
           'n_of_ants' ants
```

```
3       % In [CCBW98] an experiment is described where desert ants were ...
            trained to go to a tube,
4       % walk through it to a feeder, walk back through it and then return ...
            to their nest.
5       % The point of interest was on the returning after leaving the ...
            channel. The authors
6       % manipulated the channels from the training situation. The experts ...
            could witness that
7       % most animals at first still walked in the same direction as in the ...
            training situation,
8       % but then changed the direction towards the nest. They supposed ...
            that the desert ants could
9       % remember that at the end of the tunnel they had to walk in a ...
            certain direction to find
10      % their nest. After a certain time they started to follow the global ...
            vector.
11
12      % Environment
13      area = Area(1000,1000);          % Create an area with size 10m^2
14      area.nest = [500, 50];           % Nest position
15      area.feeder = [420,500];         % Feeder position
16
17      % Ants
18      ant = Ant(area.nest);            % Create ant at nest
19      ant.random_params = [pi/4, 0.3]; % Random parameters
20
21      %%%%%%%%%% Initialization %%%%%%%%%%
22      % Clear figure screen and hold the graphics
23      clf; hold on;
24      axis equal;
25      xlim([0, area.size(1)]);
26      ylim([0, area.size(2)]);
27
28      %%%%%%%%%% Main loop %%%%%%%%%%%%%%%
29      for ants=1:n_of_ants
30          % Starting conditions
31          ant.del_landmark(1);                              % Delete first landmark
32          testing_channel = Channel(500,500,422,500);  % Set testing channel
33          plotting_channel = testing_channel;           % Set plotting channel
34          target = testing_channel.entrance;            % Set target
35          done = 0;                                     % Ant not done yet
36
37          while ~done
38              cla;    % Clear axes
39
40              % Foraging, ant goes to nest like trained
41              if ant.status == 0
42                  if ant.move_to(target)
43                      if target == testing_channel.entrance;
44                          % put landmark at channel entrance
```

```
45                          ant.put_landmark();
46                          target = testing_channel.exit;
47                      elseif target == testing_channel.exit
48                          target = area.feeder;
49                      elseif target == area.feeder
50                          % Ant walks back through new channel
51                          plotting_channel = channel;
52                          ant.landmarks(1).pos = channel.exit;
53                          target = channel.entrance;
54                      elseif target == channel.entrance
55                          target = channel.exit;
56                      elseif target == channel.exit
57                          target = area.nest;
58                          ant.status = 1;
59                      end
60                  end
61
62              % Returning to the nest
63              elseif ant.status == 1
64                   if ant.follow_local_v()
65                         if ant.follow_global_v();
66                             % Reset: Put ant to nest
67                             ant.reset(area.nest);
68                             target = [500,500];
69                             done = 1;
70                         end
71                   end
72              end
73
74              % Plot
75              area.plot_nest();
76              area.plot_feeder();
77              plotting_channel.plot();
78              ant.plot();
79
80              % Debugging: Different
81              text(10, area.size(2)-60, strcat('\phi = ', ...
                    int2str(ant.phi*180/pi), ' '));
82              text(10, area.size(2)-100, strcat('l = ', int2str(ant.l)));
83
84              pause(0.001);  % Pause 0.001 seconds
85          end
86      end
87  end
```

A.10 moved_nest.m

```
function moved_nest(n_of_ants)
% Describes an experiment where the ant is forced to walk through a
% landmark route containing a corridor of black cylinders.
% After being trained on that route the ants nest gets shifted to another
% position where the ant starts its foraging process once again.
% The landmark vectors are directed north as from the trainings procedure.
% After returning from the feeder some ants decide to follow the landmark
% route where as others decide to follow the global vector directly to the
% nest. The different trails are colored with different colors to show the
% difference of the trajectories.

%Input:
% n_of_ants: Amount of ants that run the experiment with the given inputs.

    % Environment
    area = Cylinder_Area(1500,1500,850,200,6); % Create an area with ...
        size 15m^2 and constructs the Cylinders
    area.nest = [125 1250];
    area.nest_two = [800, 1350];
    area.feeder = [750 100];

    % Ants
    ant = cell(1,n_of_ants);

    %%%%%%%%%% Initialization %%%%%%%%%%
    % Clear figure screen and hold the graphics
    clf; hold on;
    axis equal;
    xlim([0, area.size(1)]);
    ylim([0, area.size(2)]);

    do_training = 1;
    yes = 0;

    %%%%%%%%%% Main loop %%%%%%%%%%%%%%%
    for ants = 1:n_of_ants
        ant{ants} = Ant(area.nest);
        ant{ants}.random_params = [pi/3, 0.3];
        ant{ants}.k = 0.09;

        %%%%%%%%%%%% Wich trajectory the Ant chooses (red, green,blue) ...
            %%%%%%%%%%
        red = 0;
        green = 0;
        a = rand;
        if a <= 1/3
```

```
45              target = [area.cylinders(4) area.cylinders(10)];
46          elseif a > 1/3 && a <= 2/3
47              target = [area.cylinders(2)-80 area.cylinders(8)];
48              ant{ants}.trail_color = 'red';
49              red = 1;
50          elseif a > 2/3
51              target = area.feeder;
52              ant{ants}.trail_color = 'green';
53              green = 1;
54          end
55
56          done = 0;
57          while ~done
58              tic; % Start time measurement
59              cla; % Clear axes
60
61              % Foraging
62              if ant{ants}.status == 0
63
64  %               %%%%%%%%%%% Trainings situation %%%%%%%%%%%%%
65                  if do_training % First targets gets reset.
66                      if yes < 1
67                          ant{ants}.pos = area.nest_two;
68                          target = [ area.cylinders(6)+ 95 ...
                                area.cylinders(12) ];
69                          yes = 1;
70                      end
71  %                   % Ant goes through corridor in trainings situation
72                      if ant{ants}.move_to(target)
73                          if target == [area.cylinders(6)+ 95 ...
                                area.cylinders(12) ];
74                              % The ants on the 'green' trail dont always ...
                                    follow the local vector
75                              % Therefore some landmarks aren't put while ...
                                    trainings mode
76                              if green ~= 1;
77                                 ant{ants}.put_landmark_at(area.cylinders(6,:));% ...
                                       Ant stores landmark at cylinder position
78                                 n = length(ant{ants}.landmarks);
79                                 ant{ants}.landmarks(n).range = 115; % ...
                                       Determines the Range of the landmark.
80                              end
81                             target = [area.cylinders(5)- 140 ...
                                   area.cylinders(11)]; % next target gets ...
                                   chosen.
82                          elseif target == [area.cylinders(5)- 140 ...
                                area.cylinders(11)]
83                             ant{ants}.put_landmark_at(area.cylinders(5,:) ...
                                   + 20);
84                             n = length(ant{ants}.landmarks);
```

```
85                          ant{ants}.landmarks(n).range = 150;
86                          target = [area.cylinders(4) + 85 ...
                                area.cylinders(10) ];
87                      elseif target == [area.cylinders(4) + 85 ...
                            area.cylinders(10)]
88                          ant{ants}.put_landmark_at(area.cylinders(4,:) ...
                                + 20);
89                          n = length(ant{ants}.landmarks);
90                          % Range differs randomly for some Cylinders
91                          if green ~= 1
92                           ant{ants}.landmarks(n).range = 130;
93                          else
94                              ant{ants}.landmarks(n).range = 70;
95                          end
96                          target = [area.cylinders(3) - 75 ...
                                area.cylinders(9)];
97                      elseif target == [area.cylinders(3) - 75 ...
                            area.cylinders(9)]
98                          ant{ants}.put_landmark_at(area.cylinders(3,:) ...
                                + 20);
99                          n = length(ant{ants}.landmarks);
100                         ant{ants}.landmarks(n).range = 90;
101                         target = [area.cylinders(2) + 100 ...
                                area.cylinders(8)];
102                     elseif target == [area.cylinders(2) + 100 ...
                            area.cylinders(8)]
103                         ant{ants}.put_landmark_at(area.cylinders(2,:) ...
                                + 20)
104                         n = length(ant{ants}.landmarks);
105                         ant{ants}.landmarks(n).range = 130;
106                         target = [area.cylinders(1) - 85 ...
                                area.cylinders(7)];
107                     elseif target == [area.cylinders(1) - 85 ...
                            area.cylinders(7)]
108                         ant{ants}.put_landmark_at(area.cylinders(1,:) ...
                                + 20)
109                         n = length(ant{ants}.landmarks);
110                         ant{ants}.landmarks(n).range = 110;
111                         target = area.feeder;
112                     elseif target == area.feeder;
113                          do_training = 0; % Finishes trainings situation
114                          ant{ants}.reset(area.nest); % Ant are reset
115                     end
116                 end
117
118
119             %%%%%%%%%%% Foraging from second nest %%%%%%%%%%%%%%%%%
120             else
121                 if ant{ants}.move_to(target)
122                     if target == [area.cylinders(4), area.cylinders(10)]
```

```
123                         target = [area.cylinders(2)+ 50 ...
                                area.cylinders(8)];
124                     elseif target == [area.cylinders(2)+ 50 ...
                            area.cylinders(8)]
125                         target = area.feeder();
126                     elseif target == [area.cylinders(2)-80 ...
                            area.cylinders(8)]
127                         target = area.feeder();
128                     elseif target == area.feeder();
129                         ant{ants}.status = 1;
130                     end
131                 end
132             end
133         % Returning to Nest
134         elseif ant{ants}.status == 1
135             if red == 1;
136                 if ant{ants}.follow_local_v();
137                     if  ~ant{ants}.follow_global_v()
138                     else
139                         ant{ants}.reset(area.nest);
140                         ant{ants}.landmarks = [];
141                         done = 1;
142                         do_training = 1;
143                         yes = 0;
144                     end
145                 end
146             elseif green == 1;
147                 if ant{ants}.follow_local_v();
148                     if  ~ant{ants}.follow_global_v()
149                     else
150                         ant{ants}.reset(area.nest);
151                         ant{ants}.landmarks = [];
152                         done = 1;
153                         do_training = 1;
154                         yes = 0;
155                     end
156                 end
157             else
158                if ~ant{ants}.follow_global_v()
159                else
160                         ant{ants}.reset(area.nest); % Ants are reset
161                         ant{ants}.landmarks = []; % Landmarks are reset
162                         done = 1;
163                         do_training = 1; % Trainings mode is ...
                                inserted for next ant
164                         yes = 0;
165                end
166             end
167         end
168
```

```
169             %Plot
170             area.plot_landmark_route;
171             area.plot_new_nest(1);
172             area.plot_new_nest(2);
173             area.plot_feeder();
174             area.plot_cylinders();
175             for i=1:n_of_ants
176                 if ant{i} ~= 0
177                     ant{i}.plot();
178                 end
179             end
180             area.nest_two = [800, 1350];
181 
182             pause(0.001);
183         end
184     end
185 end
```

A.11 crooked_corridor.m

```
1  function crooked_corridor(n_of_ants)
2  % Describes an experiment where the ant is forced to walk through a
3  % a corridor of black cylinders and return back to their nest.
4  % The difference here is, that the corridor is crooked about 45 .
5  % The trails are recorded to watch the ants behaviour under influence of
6  % the visual landmarks.
7 
8  %Input:
9  % n_of_ants: Amount of ants that run the experiment with the given inputs.
10 
11     % Environment
12     area = Cylinder_Area(1500,1500,1300,300,6); % Create an area with ...
           size 15m^2 and constructs the Cylinders
13     area.nest = [1200, 30];
14     area.feeder = [100, 1200];
15 
16     %%%%%%%% Crook the Corridor %%%%%%%%%
17     i = area.amount;
18     j = area.amount*2;
19     % Cylinders are shifted to built a crooked corridor
20     while i > 0
21         area.cylinders(i) = area.cylinders(i) - 160*(i);
22         area.cylinders(j) = area.cylinders(j) - 90;
23         i = i - 1;
24         j = j - 1;
25     end
26     u = 6;
```

```
27      v = 12;
28
29      while u > 1
30          area.cylinders(u) = area.cylinders(u) + 30;
31          area.cylinders(v) = area.cylinders(v) - 85;
32          v = v - 2;
33          u = u - 2;
34      end
35
36      % Ants
37      ant = Ant(area.nest);
38      ant.random_params = [pi/3, 0.3];
39      ant.k = 0.13;
40
41      %%%%%%%%%% Initialization %%%%%%%%%%
42      % Clear figure screen and hold the graphics
43      clf; hold on;
44      axis equal;
45      xlim([0, area.size(1)]);
46      ylim([0, area.size(2)]);
47
48      %%%%%%%%%% Main loop %%%%%%%%%%%%%%%
49      for ants = 1: n_of_ants
50          % Generates a random number in the intervall [-60,120]
51          % Is used to determine how close the ant approaches to Cylinder.
52          a = -60;
53          b = 120;
54          rand_n = a + (b-a).*rand(1);
55
56          % After starting from the nest, which nearby cylinder gets chosen
57          c = rand;
58          % (Chances 50%)
59          if c > 0.5
60              target = [area.cylinders(1)-rand_n area.cylinders(7)];
61          else
62              target = [area.cylinders(2)-rand_n area.cylinders(8)];
63          end
64
65          done = 0;
66             while ~done
67                  tic; % Start time measurement
68                  cla; % Clear axes
69                  % Foraging
70                  if ant.status == 0
71                      if ant.move_to(target)
72                          if target == [area.cylinders(1)-rand_n ...
                                area.cylinders(7)]
73                              ant.put_landmark_at(area.cylinders(1,:)+ 20);
74                              n = length(ant.landmarks);
```

```
75                          ant.landmarks(n).range = 110; % Determines ...
                                the Range of the landmark.
76                          rand_n = a + (b-a).*rand(1);  % Generates ...
                                new approach range( how close to come to ...
                                the cylinder.)
77                          % Decides with a chance of 30% which of the ...
                                3 next cylinder to approach
78                          c = rand;
79                          if c <= 1/3
80                              target = [area.cylinders(2) + rand_n ...
                                    area.cylinders(8)];
81                          elseif c > 1/3 && c <= 2/3
82                              target = [area.cylinders(3) - rand_n ...
                                    area.cylinders(9)];
83                          else
84                              target = [area.cylinders(4) + rand_n ...
                                    area.cylinders(10)];
85                          end
86                      elseif target == [area.cylinders(2) - rand_n ...
                            area.cylinders(8)]
87                          ant.put_landmark_at(area.cylinders(2,:)+ 20);
88                          n = length(ant.landmarks);
89                          ant.landmarks(n).range = 110; % Determines ...
                                the Range of the landmark.
90                          rand_n = a + (b-a).*rand(1);
91                          target = [area.cylinders(4) + rand_n ...
                                area.cylinders(10)];
92                      elseif target == [area.cylinders(2) + rand_n ...
                            area.cylinders(8)]
93                          ant.put_landmark_at(area.cylinders(2,:)+ 20);
94                          n = length(ant.landmarks);
95                          ant.landmarks(n).range = 110;
96                          rand_n = a + (b-a).*rand(1);
97                          % Decides with a chance of 25%/75% next ...
                                Cylinder to approach
98                          % Cylinder which is more far away, has less ...
                                chance
99                          c = rand;
100                         if c <= 1/4
101                             target  = [area.cylinders(3)-rand_n ...
                                    area.cylinders(9)];
102                         else
103                             target = [area.cylinders(4)+rand_n ...
                                    area.cylinders(10)];
104                         end
105                     elseif target == [area.cylinders(3)-rand_n ...
                            area.cylinders(9)]
106                         ant.put_landmark_at(area.cylinders(3,:) + 20);
107                         n = length(ant.landmarks);
108                         ant.landmarks(n).range = 110;
```

```
109                         rand_n = a + (b-a).*rand(1);
110                         % Decides with a chance of 25%/75% next approach
111                         c = rand;
112                         if c <= 1/4
113                             target = [area.cylinders(4)+rand_n ...
                                    area.cylinders(10)];
114                         else
115                             target = [area.cylinders(5) - rand_n ...
                                    area.cylinders(11)];
116                         end
117                     elseif target == [area.cylinders(4)+rand_n ...
                            area.cylinders(10)]
118                         ant.put_landmark_at(area.cylinders(4,:) + 20);
119                         n = length(ant.landmarks);
120                         ant.landmarks(n).range = 110;
121                         rand_n = a + (b-a).*rand(1);
122                         % Decides with a chance of 25%/75% next approach
123                         c = rand;
124                         if c <= 1/4
125                             target = [area.cylinders(5) - rand_n ...
                                    area.cylinders(11)];
126                         else
127                             target = [area.cylinders(6) + rand_n ...
                                    area.cylinders(12)];
128                         end
129                     elseif target == [area.cylinders(5) - rand_n ...
                            area.cylinders(11)]
130                         n = length(ant.landmarks);
131                         ant.landmarks(n).range = 110; % Determines ...
                                the Range of the landmark.
132                         ant.put_landmark_at(area.cylinders(5,:) + 20);
133                         rand_n = a + (b-a).*rand(1);
134                         % Decides with a chance of 25%/75% next approach
135                         if c <= 1/4
136                             target = [area.cylinders(6) + rand_n ...
                                    area.cylinders(12)];
137                         else
138                             target = area.feeder;
139                         end
140                     elseif target == [area.cylinders(6) + rand_n ...
                            area.cylinders(12)]
141                         n = length(ant.landmarks);
142                         ant.landmarks(n).range = 110; % Determines ...
                                the Range of the landmark.
143                         ant.put_landmark_at(area.cylinders(6,:) + 20);
144                         target = area.feeder;
145                     elseif target == area.feeder
146                         ant.status = 1;
147                     end
148                 end
```

```
149                 % Returning to Nest
150                 elseif ant.status == 1
151                     if ant.follow_local_v();
152                         if  ~ant.follow_global_v()
153                         else
154                             ant.reset(area.nest);
155                             ant.landmarks = [];
156                             done = 1;
157                         end
158                     end
159                 end
160                 % Plot
161                 area.plot_nest();
162                 area.plot_feeder();
163                 area.plot_cylinders();
164                 ant.plot();
165 
166                 pause(0.001);
167             end
168     end
169 end
```

A.12 corridor_of_black_cylinders.m

```
1  function corridor_of_black_cylinders(n_of_ants)
2  % Describes an experiment where the ant is forced to walk through a
3  % a corridor of black cylinders and return back to their nest.
4  % The trails are recorded to watch the ants behaviour under influence of
5  % the visual landmarks.
6 
7  %Input:
8  % n_of_ants: Amount of ants that run the experiment with the given inputs.
9 
10     % Environment
11     area = Cylinder_Area(1500,1500,850,280,6); % Create an area with ...
           size 15m^2 and constructs the Cylinders
12     area.nest = [750, 30];
13     area.feeder = [650, 1480];
14     % Ants
15     ant = Ant(area.nest);
16     ant.random_params = [pi/4, 0.3];
17     ant.k = 0.08;
18 
19     %%%%%%%%%% Initialization %%%%%%%%%%
20     % Clear figure screen and hold the graphics
21     clf; hold on;
22     axis equal;
```

```
23      xlim([0, area.size(1)]);
24      ylim([0, area.size(2)]);
25
26      %%%%%%%%%% Main loop %%%%%%%%%%%%%%
27      for ants = 1: n_of_ants
28          % Generates a random number in the intervall [-60,120]
29          % Is used to determine how close the ant approaches to Cylinder.
30          a = -60;
31          b = 120;
32          rand_n = a + (b-a).*rand(1);
33
34          % After starting from the nest, which nearby cylinder gets chosen
35          c = rand;
36
37          % (Chances 50%)
38          if c > 0.5
39              target = [area.cylinders(1)-rand_n area.cylinders(7)];
40          else
41              target = [area.cylinders(2)-rand_n area.cylinders(8)];
42          end
43
44          done = 0;
45              while ~done
46                  tic; % Start time measurement
47                  cla; % Clear axes
48                  % Foraging
49                  if ant.status == 0
50                      if ant.move_to(target)
51                          if target == [area.cylinders(1)-rand_n ...
                                area.cylinders(7)]
52                              ant.put_landmark_at(area.cylinders(1,:)+ ...
                                    20); % Ant stores landmark at cylinder ...
                                    position
53                              n = length(ant.landmarks);
54                              ant.landmarks(n).range = 110; % Determines ...
                                    the Range of the landmark.
55                              rand_n = a + (b-a).*rand(1); % Generates new ...
                                    approach range( how close to come to the ...
                                    cylinder.)
56                              % Decides with a chance of 30% which of the ...
                                    3 next cylinder to approach
57                              c = rand;
58                              if c <= 1/3
59                                  target = [area.cylinders(2) + rand_n ...
                                        area.cylinders(8)];
60                              elseif c > 1/3 && c <= 2/3
61                                  target = [area.cylinders(3) - rand_n ...
                                        area.cylinders(9)];
62                              else
```

```
63                          target = [area.cylinders(4) + rand_n ...
                                area.cylinders(10)];
64                      end
65                  elseif target == [area.cylinders(2) - rand_n ...
                        area.cylinders(8)]
66                      ant.put_landmark_at(area.cylinders(2,:)+ 20);
67                      n = length(ant.landmarks);
68                      ant.landmarks(n).range = 110;
69                      rand_n = a + (b-a).*rand(1);
70                      target = [area.cylinders(4) + rand_n ...
                            area.cylinders(10)];% next target gets ...
                            chosen.
71                  elseif target == [area.cylinders(2) + rand_n ...
                        area.cylinders(8)]
72                      ant.put_landmark_at(area.cylinders(2,:)+ 20);
73                      n = length(ant.landmarks);
74                      ant.landmarks(n).range = 110;
75                      rand_n = a + (b-a).*rand(1);
76                      % Decides with a chance of 25%/75% next ...
                            Cylinder to approach
77                      % Cylinder which is more far away, has less ...
                            chance
78                      c = rand;
79                      if c <= 1/4
80                          target  = [area.cylinders(3)-rand_n ...
                                area.cylinders(9)];
81                      else
82                          target = [area.cylinders(4)+rand_n ...
                                area.cylinders(10)];
83                      end
84                  elseif target == [area.cylinders(3)-rand_n ...
                        area.cylinders(9)]
85                      ant.put_landmark_at(area.cylinders(3,:) + 20);
86                      n = length(ant.landmarks);
87                      ant.landmarks(n).range = 110;
88                      rand_n = a + (b-a).*rand(1);
89                      % Decides with a chance of 25%/75% next approach
90                      c = rand;
91                      if c <= 1/4
92                          target = [area.cylinders(4)+rand_n ...
                                area.cylinders(10)];
93                      else
94                          target = [area.cylinders(5) - rand_n ...
                                area.cylinders(11)];
95                      end
96                  elseif target == [area.cylinders(4)+rand_n ...
                        area.cylinders(10)]
97                      ant.put_landmark_at(area.cylinders(4,:)+ 20);
98                      n = length(ant.landmarks);
99                      ant.landmarks(n).range = 110;
```

```
100                         rand_n = a + (b-a).*rand(1);
101                         % Decides with a chance of 25%/75% next approach
102                         c = rand;
103                         if c <= 1/4
104                             target = [area.cylinders(5) - rand_n ...
                                    area.cylinders(11)];
105                         else
106                             target = [area.cylinders(6) + rand_n ...
                                    area.cylinders(12)];
107                         end
108                     elseif target == [area.cylinders(5) - rand_n ...
                            area.cylinders(11)]
109                         ant.put_landmark_at(area.cylinders(5,:)+ 20);
110                         n = length(ant.landmarks);
111                         ant.landmarks(n).range = 110;
112                         rand_n = a + (b-a).*rand(1);
113                         % Decides with a chance of 25%/75% next ...
                                trajecetory next approach
114                         if c <= 1/4
115                             target = [area.cylinders(6) + rand_n ...
                                    area.cylinders(12)];
116                         else
117                             target = area.feeder;
118                         end
119                     elseif target == [area.cylinders(6) + rand_n ...
                            area.cylinders(12)]
120                         ant.put_landmark_at(area.cylinders(6,:) + 20);
121                         n = length(ant.landmarks);
122                         ant.landmarks(n).range = 110;
123                         target = area.feeder;
124                     elseif target == area.feeder
125                         ant.status = 1;
126                     end
127                 end
128             % Returning to Nest
129             elseif ant.status == 1
130                 if ant.follow_local_v();
131                     if  ~ant.follow_global_v()
132                     else
133                         ant.reset(area.nest); % Ants are reset
134                         ant.landmarks = []; % Landmarks are reset
135                         done = 1;
136                     end
137                 end
138             end
139             % Plot
140             area.plot_nest();
141             area.plot_feeder();
142             area.plot_cylinders();
143             ant.plot();
```

```
144
145                 pause(0.001);
146             end
147     end
148 end
```

A.13 natural_foraging.m

```
1  function reached_nest = natural_foraging(n_of_ants, lm_distance)
2      % NATURAL FORAGING
3      % This experiment recreates roughly a natural foraging sitation where
4      % all the implemented navigational modules are collaborating and thus
5      % 'guiding' the ant throughout its trip in a best possible way.
6      % The ant starts in the center of a 10m^2 area. In every run, a feeder
7      % site is put on a random place. The ant picks random points on the
8      % area and moves to them. As soon as it gets near to the feeder, its
9      % walks towards the feeders center to collect food, and then moves back
10     % to the nest, guided by the global vector. During the ants' trip it
11     % sets landmarks after certain distances, which additionally help the
12     % ant when moving back to the nest.
13
14     % Output:
15     % reached_nest: Whether or not the ant found the way back to the nest
16
17     % Input:
18     % n_of_ants: Amount of ants that run the test
19     % lm_distance: Landmark distance. Interval at which the ant sets
20     % landmarks
21
22     % Output
23     reached_nest = zeros(1,n_of_ants);
24
25     % Environment
26     area = Area(1000,1000); % Create an area with size 10m^2
27     area.nest = [500, 500];
28
29     %%%%%%%%%% Initialization %%%%%%%%%%
30     % Clear figure screen and hold the graphics
31     clf; hold on;
32     axis equal;
33     xlim([0, area.size(1)]);
34     ylim([0, area.size(2)]);
35
36     %%%%%%%%%% Main loop %%%%%%%%%%%%%%%
37
38     for ants=1:n_of_ants % Loops through the specified amount of ants
39
```

```
40          % Put the feeder at a randomly generated position between 100 and
41          % 900 on the area
42          area.feeder = [randi(800)+100, randi(800)+100];
43          % Create an ant and put it at the nest
44          ant = Ant(area.nest);
45          % Create a special landmark (so that the ant later on 'sees' the
46          % nest from a certain distance (the landmarks' range) and can
47          % return to it
48          ant.put_landmark_at(area.nest);
49          ant.landmarks(1).length = 0;
50          ant.landmarks(1).range = 80;
51
52          % Create special landmark for the feeder site
53          ant.put_landmark_at(area.feeder);
54          ant.landmarks(2).length = 0;
55          ant.landmarks(2).range = 80;
56
57          % Set initial random target
58          target = [randi(1000), randi(1000)];
59          % Set parameters for random walk
60          ant.random_params = [pi/4, 0.3];
61          % Temporary variables
62          done = 0;
63          % Variable that holds the distance the last landmark was set
64          put_after = 0;
65
66          % Main loop
67          while ~done
68              cla; % Clear axes
69
70              % Phase 1: Foraging
71              ant.show_trail = 1;
72              if ant.status == 0
73                  if max(ismember(ant.within_landmark, ant.landmarks(2)))
74                      % The ant has crossed the feeder landmark and walks
75                      % now towards the center
76                      if ant.move_to(ant.landmarks(2).pos)
77                          % The ant collected food and returns back to the
78                          % nest
79                          ant.status = 1;
80                      end
81                  else
82                      if abs(ant.l-put_after) > lm_distance
83                          % The ant covered the required distance to set a
84                          % landmark
85                          put_after = ant.l;
86                          ant.put_landmark();
87                      end
88                      % The ant walks towards its target
89                      if ant.move_to(target)
```

```
90                          % Arrived at target, set new random target
91                          target = [randi(1000), randi(1000)];
92                      end
93                  end
94
95              % Phase 2: Returning to the nest
96              elseif ant.status == 1
97                  if max(ismember(ant.within_landmark, ant.landmarks(1)))
98                      % The ant has crossed the nest landmarks and walks now
99                      % towards the center
100                     if ant.move_to(ant.landmarks(1).pos)
101                         % The ant arrived at the nest and completed the run
102                         done = 1;
103                         reached_nest(ants) = 1;
104                     end
105                 else
106                     % The ant first follows local vectors on its way home
107                     if ant.follow_local_v()
108                         % If the local vectors are used up, it follows the
109                         % global vector
110                         if ant.follow_global_v()
111                             % The ant walked back its calculated covered
112                             % distance but did not return at the nest.
113                             % The and completed the run.
114                             done = 1;
115                             reached_nest(ants) = 0;
116                         end
117                     end
118                 end
119             end
120
121             % Plot
122             ant.plot();
123             area.plot_nest();
124             area.plot_feeder();
125
126             % Debugging
127             text(10, area.size(2)-60, strcat('\phi = ', ...
                    int2str(ant.phi*180/pi), ' '));
128             text(10, area.size(2)-100, strcat('l = ', int2str(ant.l)));
129
130             pause(0.001);
131         end
132         % Reset the ants' properties and put it back to the nest for the
133         % next run
134         ant.reset(area.nest);
135     end
136 end
```