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

Abstract

The purpose of this paper is to reconstruct the recent findings about the navigation of deset ants. Desert ants need to move a long distance to find food. It is believed that they use vectors to find their way back to the nest. The animals continously approximate the vector pointing homewards. Furthermore the animals use landmarks for orientation in the desert. They set local vectors at these landmarks pointing to the nest. These thesis were implemented in MATLAB and it could be shown that these assumptions where correct. Anyway there are more factors that influece the navigation of desert ants which were not implemented.

Contents

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.

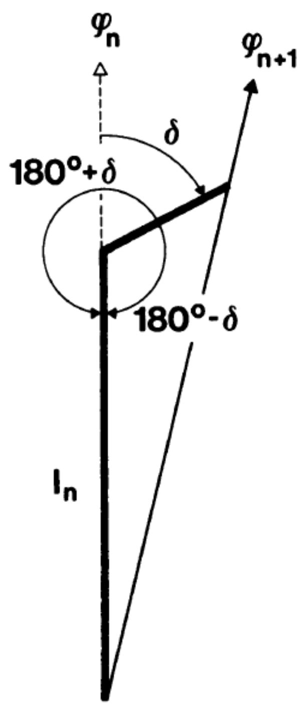

Figure 1: Illustration of the mean direction formula, as depicted in [MW88]

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 8).

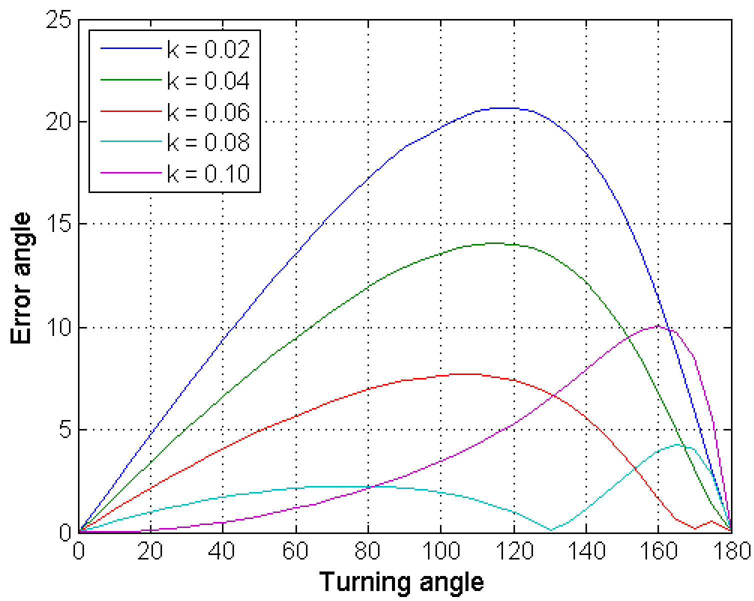

Figure 2: Results for k between 0.02 and 0.10

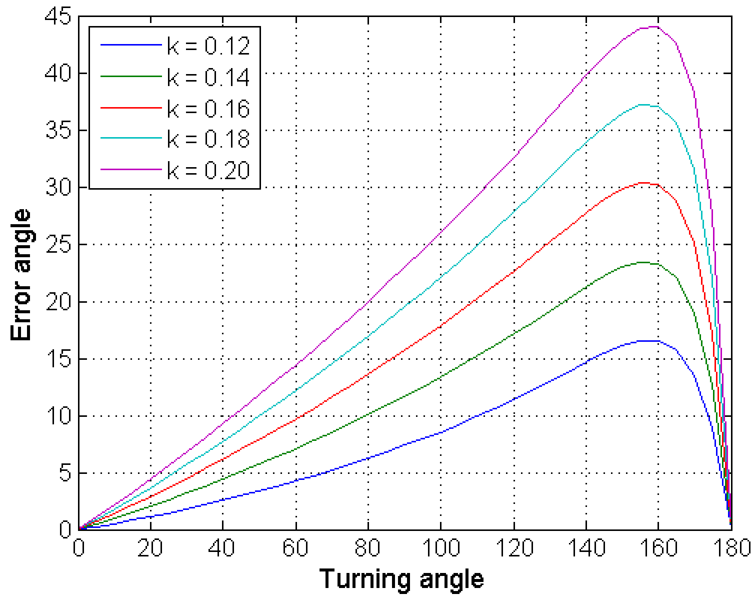

Figure 3: Results for k between 0.12 and 0.20

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.

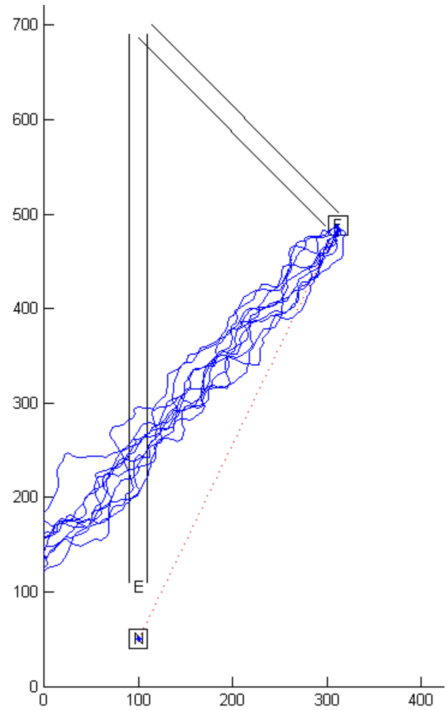

Figure 4: Trails of 10 ants with a = 6m, b = 3m, α = 135° and k = 0.13; N: Nest, E: Entrance, F: Feeder; Dotted red line indicates direct connection from feeder to nest

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.

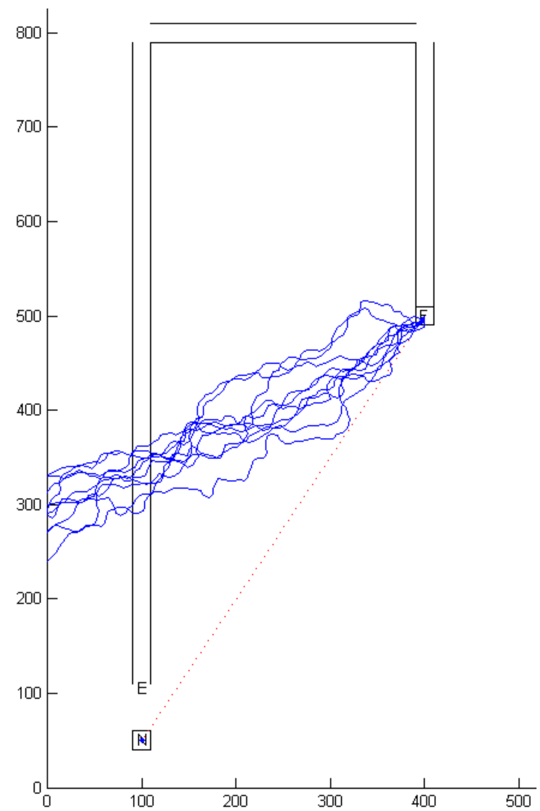

Figure 5: Trails of 10 ants with a = 7m, b = 3m, c = 3m, k = 0.13; N: Nest, E: Entrance, F: Feeder; Dotted red line indicates direct connection from feeder to nest

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.

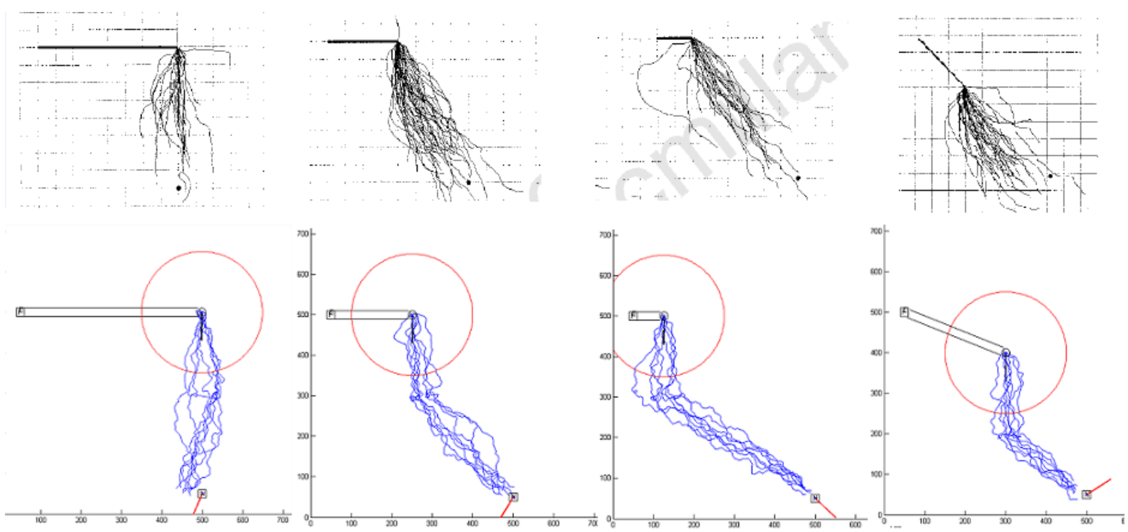

Figure 6: Resulting trails. Upper row taken from [CCBW98]; Fig. 1: Test scenario. Fig. 2: Channel half as long as training channel. Fig. 3: 1/4 of training channel Fig. 4: Rotated channel with half length of training channel. For all figures the parameters k=0.07, landmark.range=150, landmark.length=70 and random_params = [π/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

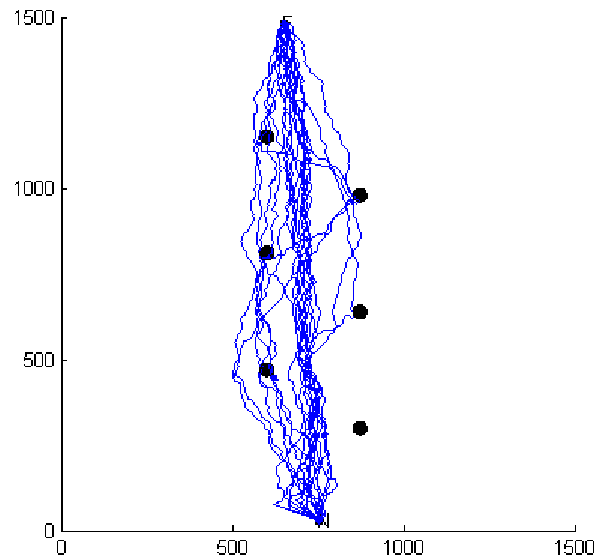

Figure 7: Trails of 10 ants N: Nest F: Feeder; Black circles indicate cylinders which are used as visual landmarks

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

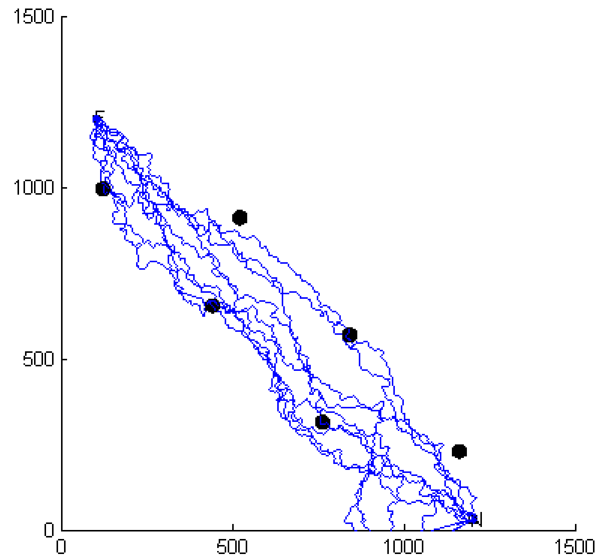

Figure 8: Trails of 10 ants N: Nest F: Feeder; Black circles indicate cylinders which are used as visual landmarks, same experiment as Fig. 7 but with a crooked corridor

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

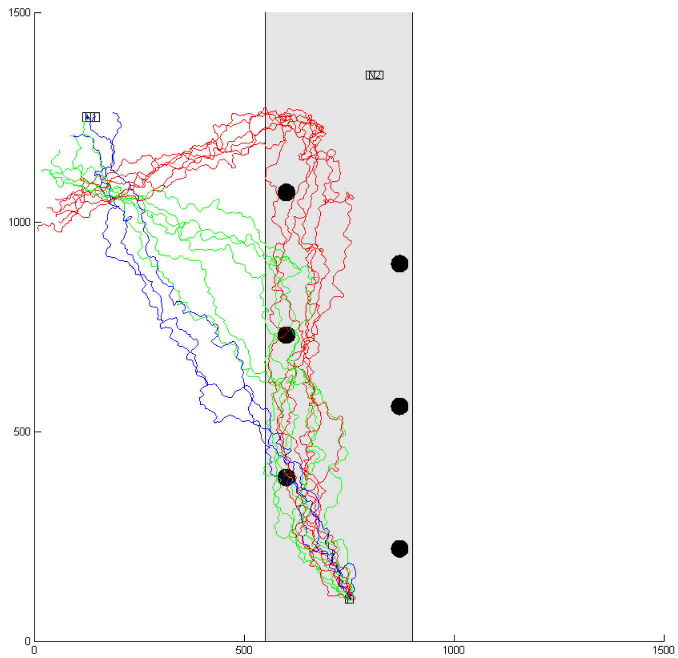

Figure 9: Trails of 15 ants N2: Nest where ants train(through landmark route) N1: New nest where ants get deposited after being catched at the feeder F: Feeder; Black circles indicate cylinders which are used as visual landmarks; Different trail colors: Red: Ants which follow the landmark route, Green: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

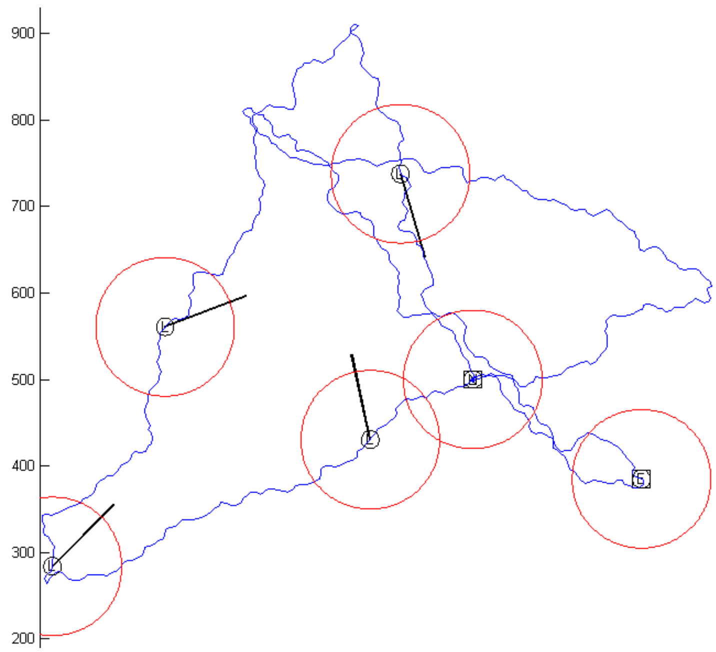

Figure 10: Example of a single foraging path with *lm_distance* = 70; N: Nest, F: Feeder, L: Landmarks with range as red circles and local vectors as black lines.

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 13, 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
```

```
            obj.status = 0;         % Set status to 'foraging'
            obj.reset_global_v;     % Reset the global vector
            obj.target = [0, 0];    % Reset current target
            obj.pos = pos;          % Put the ant back to 'pos'
        end
        % Moving
        function arrived = move_to(obj, target)
            % Lets the ant move to the point specified by target
            % Returns 1 if the ant has arrived at the point

            arrived = 0;
            dist = target - obj.pos;        % Distance vector between the ...
                current
                                            % position and the target
            e_dist = norm(dist);            % Euclidean distance

            if e_dist >= obj.speed          % If the distance is bigger ...
                than the ants' step
                                            % (i.e. the ant has not ...
                                                arrived yet
                % Move the ant
                old_ang = obj.ang;                          % Save old angle
                obj.ang = angle(dist(1) + dist(2)*1i);      % Compute new ...
                    angle
                delta = mod(obj.ang - old_ang, 2*pi);       % Compute ...
                    current turning angle
                if delta > pi                               % and adjust ...
                    the range
                    delta = delta -2*pi;                    % to -pi <= ...
                        delta <= pi
                end
                % Compute a random angle between -alpha <= ran <= alpha,
                % where alpha = random_params(1) is the random movement
                % angle
                ran = 2*obj.random_params(1)*rand-obj.random_params(1);
                % Compute the drift for random walk with random_params(2)
                drift = obj.random_params(2)*delta;
                % Apply the random values to the ants' old angle
                obj.ang = old_ang + ran + drift;

                % Compute the move vector for this step
                move = [cos(obj.ang)*obj.speed, sin(obj.ang)*obj.speed];
                % Advance position by the move vector
                obj.pos = obj.pos + move;

                obj.update_global_v();          % Update global vector
                obj.update_local_v();           % Update local vector

                if obj.show_trail
                    % Builds up the trail behind the ant
```

```
                    n = size(obj.trail);
                    if norm(obj.trail(n(1),:)-obj.pos) > 5
                        % Extend the trail with a new point
                        obj.trail = [obj.trail; obj.pos];
                    end
                else
                    if length(obj.trail) > 2
                        % Save the trail
                        obj.trails{length(obj.trails)+1} = obj.trail;
                    end
                    obj.trail = obj.pos;
                end
            else
                arrived = 1;
            end
        end

        % Landmarks
        function put_landmark(obj)
            % Puts a landmark at the current position with the current
            % global vector as local vector
            obj.landmarks = [obj.landmarks, Landmark(obj.pos(1), ...
                obj.pos(2), obj.global_v(1), obj.global_v(2))];
        end

        function put_landmark_at(obj, place)
            % Put a landmark at the given position with the current
            % global vector as local vector
            obj.landmarks = [obj.landmarks, Landmark(place(1), place(2), ...
                obj.global_v(1), obj.global_v(2))];
        end

        function del_landmark(obj, i)
            % Removes landmark with index i
            obj.landmarks = [obj.landmarks(1:i-1), ...
                obj.landmarks(i+1:length(obj.landmarks))];
        end
        function landmark = within_landmark(obj)
            % Returns a list of landmarks in whose range the ant is
            n = length(obj.landmarks);
            landmark = []; % Default case - the ant is not within a landmark
            for i=n:-1:1
                % Distance between the ant and landmark i
                dist = sqrt( (obj.pos(1)-obj.landmarks(i).pos(1))^2 + ...
                    ((obj.pos(2)-obj.landmarks(i).pos(2)))^2 );
                if dist < obj.landmarks(i).range
                    % If the ant lies within range, extend the list of
                    % landmarks
                    landmark = [landmark, obj.landmarks(i)];
                end
```

```
            end
        end
        % Vectors
        function update_global_v(obj)
            % Update the ant's global vector
            if strcmp(obj.phi, 'null')                   % Check for ...
                default value
                obj.phi = obj.ang;
            end
            delta = mod(obj.ang - obj.phi, 2*pi);        % Compute angle delta
            if delta > pi                                % Scale range to ...
                -pi<=delta<=pi
                delta = delta -2*pi;
            end
            % Compute covered distance for the current step
            obj.l = obj.l + (1 - abs(delta)/(pi/2));     % Mean distance
            % Compute mean direction for the current step
            % Version 1
            %obj.phi = (obj.l*obj.phi + obj.phi + delta)/(obj.l + 1);
            % Version 2
            obj.phi = mod(obj.phi,2*pi) + obj.k * ...
                ((pi+delta)*(pi-delta)*delta)/obj.l;

            if obj.phi > pi                              % Scale range to ...
                -pi<=phi<=pi
                obj.phi = obj.phi - 2*pi;
            end
            % Compute global vector
            obj.global_v = [70*cos(obj.phi + pi), 70*sin(obj.phi + pi)];
            % Compute mean vector
            obj.mean_v = [70*cos(obj.phi), 70*sin(obj.phi)];
        end
        function reset_global_v(obj)
            % Resets the global vector and its dependent properties
            obj.global_v = [0,0];
            obj.ang = 0;
            obj.phi = 'null';
            obj.l = 0;
        end
        function arrived = follow_global_v(obj)
            % Lets the ant follow the global vector
            obj.show_trail = 1;          % Enables the trail
            arrived = 1;                 % Default case
            if obj.l > 5
                % If there's still a distance to travel,
                % keep following the global vector
                obj.move_to(obj.pos+obj.global_v);
                arrived = 0;             % The ant is still on its way
            end
        end
```

```
        function update_local_v(obj)
            % Keeps the local vector associated with the last visited
            % landmark
            % Get all landmarks the ant walks on
            current_landmark = obj.within_landmark();
            if length(current_landmark) > 1
                % If there are multiple landmarks, pick the first
                current_landmark = current_landmark(1);
            end
            if current_landmark ~= 0
                % Get the local vector associated with the current landmark
                obj.local_v = current_landmark.local_v;
            else
                % The ant lost its local vector
                obj.local_v = [0, 0];
            end
        end
        function arrived = follow_local_v(obj)
            % Lets the ant follow the local vector (if there is any)
            obj.show_trail = 1;     % Enables trail
            if obj.within_landmark ~= 0
                % As long as the ant lies within the landmark, it keeps
                % the local vector and updates the place it points to
                obj.target = obj.pos + obj.local_v;
            end
            arrived = 1;
            if obj.target ~= 0
                % The local vector is not processed yet
                arrived = 0;
                % Plot local vector
                plot([obj.pos(1),obj.target(1)], [obj.pos(2), ...
                    obj.target(2)], 'magenta', 'LineWidth', 2);

                if obj.move_to(obj.target)
                    % The ant arrived at the point the local vector points
                    % to and resets this point (the target)
                    arrived = 1;
                    obj.target = [0, 0];
                end
            end
        end

        % Other
        function plot(obj)
            % Plot the ant as a point
            plot(obj.pos(1), obj.pos(2), '.');

            % Plot landmarks
            for i=1:length(obj.landmarks)
                obj.landmarks(i).plot();
```

```
            end
            % Plot mean vector
            % plot([obj.pos(1),obj.pos(1)+70*cos(obj.phi)],[obj.pos(2), ...
                obj.pos(2)+70*sin(obj.phi)], 'red');
            % Plot global vector
            plot([obj.pos(1),obj.pos(1)+obj.global_v(1)],[obj.pos(2), ...
                obj.pos(2)+obj.global_v(2)],'red', 'LineWidth', 2);

            % Plot current trail
            plot(obj.trail(:,1), obj.trail(:,2), obj.trail_color);
            % Plot previous trails
            for i=1:length(obj.trails)
                plot(obj.trails{i}(:,1), obj.trails{i}(:,2), ...
                    obj.trail_color);
            end
            obj.show_trail = 0;
        end
    end

end
```

A.3 Landmark.m

```
classdef Landmark < handle
    %LANDMARK class
    %   Describes abstract landmarks set by the foraging ant

    properties
        pos;            % Position
        range = 80;     % Covered range
        ang;            % Angle in which direction the local vector ...
            points to
        local_v;        % Local vector stored at the landmark
        length = 100;   % Length of local vector
        color = 'black';% Color of the Landmark
    end

    methods
        function obj = Landmark(varargin)
            % Constructor
            % Supports creation from both an angle and vector components
            if nargin == 3
                % Create with angle, e.g.: landmark = Landmark(x, y, phi);
                obj.pos = [varargin{1}, varargin{2}];
                obj.ang = varargin{3};
                obj.local_v = [obj.length*cos(obj.ang), ...
                    obj.length*sin(obj.ang)];
```

```
            elseif nargin == 4
                % Create with components, e.g.: landmark = Landmark(x, ...
                    y, u, v);
                obj.pos = [varargin{1}, varargin{2}];
                obj.local_v = [obj.length*varargin{3}/70, ...
                    obj.length*varargin{4}/70];
                obj.ang = angle(varargin{3} + varargin{4}*1i);
            end
        end

        function plot(obj)
            % Plots the Landmark with its corresponding local vector
            r = 10; % Radius of the inner circle
            % Plot small circle around 'L'
            rectangle('Position',[obj.pos(1)-r, obj.pos(2)-r, 2*r, 2*r], ...
                'Curvature', [1,1]);
            % Plot range as circle
            rectangle('Position',[obj.pos(1)-obj.range, ...
                obj.pos(2)-obj.range, 2*obj.range, 2*obj.range], ...
                'Curvature', [1,1], 'EdgeColor', 'r');
            % Plot local vector
            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);
            % Plot 'L'
            text(obj.pos(1)-5, obj.pos(2), 'L', 'Color', obj.color);
        end
    end

end
```

A.4 Area.m

```
classdef Area < handle
    %AREA class
    %   Describes the testing area and its basic elements

    properties
        size;
        feeder = [0, 0];
        nest = [0, 0];
    end

    methods
        function obj = Area(size_x, size_y)
            obj.size = [size_x, size_y];
        end
```

```
        function plot_nest(obj)
            text(obj.nest(1)-5, obj.nest(2), 'N');
            rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,20,20]);
        end

        function plot_feeder(obj)
            text(obj.feeder(1)-5, obj.feeder(2), 'F');
            rectangle('Position',[obj.feeder(1)-10,obj.feeder(2)-10,20,20]);
        end
    end

end
```

A.5 Cylinder_Area.m

```
classdef Cylinder_Area < handle
% Describes the area where all the experiments are held which include black
% cylinders as visual landmarks.

    properties
        amount;             % amount of cylinders
        cylinders;          % 2x6 Matrix, containing all the X,Y ...
            coordinates of the cylinders
        landmark_route;     % Displays the Training route
        size = [0,0];       % Area size
        feeder = [0, 0];    % Feeder position
        nest = [0, 0];      % Nest position
        nest_two = [0 0];   % Second Nest position ( used for ...
            "moved_nest" experiment.
    end

    methods

        % Creation procedure of the Cylinder Area
        function obj = Cylinder_Area(length_x, length_y, ...
            start_x,start_y,n_of_cylinders)
            xlim([0, length_x]); % Length of the area in X direction
            ylim([0, length_y]); % Length of the area in Y direction
            obj.size = [length_x, length_y];
            obj.cylinders = zeros(n_of_cylinders,2);
            obj.amount = n_of_cylinders;% amount of cylinders on the area


            % Sets the feeder position
            obj.feeder(1) = start_x-200;
            obj.feeder(2) = start_y + obj.amount*200;
```

```

            % Sets the nest position
            obj.nest(1) = start_x - 85;
            obj.nest(2) = start_y - 50;

            obj.cylinders(1,:) = [start_x start_y]; % Stores position of ...
                cylinder 1

            k = 2; % Starts storing at Cylinder 2 ( 1st one is stored ...
                allready

            % Places all Cylinders on the Map
            for i = 1:obj.amount
                start_y = start_y + 170;
            % Alternaterly left or right side of the corridor
                if mod(i,2) == 0
                    start_x = start_x + 270;
                else
                    start_x = start_x - 270;
                end

                % Stores The coordinates of the Cylinders
                if k  <= obj.amount
                    obj.cylinders(k,:) = [start_x start_y];
                    k = k+1;
                 end

                hold on
                axis square;
            end
        end

        % Plots all the Cylinders into the map
        function plot_cylinders(obj)
            for i = 1: obj.amount
                rectangle('Position',[obj.cylinders(i,:),40,40], ...
                    'Curvature', [1,1], 'FaceColor', 'k');
            end
        end

        %Plots the training/landmark route (moved_nest experiment)
        function plot_landmark_route(obj)
           obj.landmark_route = rectangle('Position',[550,0,350,1500], ...
               'Curvature', [0,0], 'FaceColor', [0.9, 0.9, 0.9]);
        end

        % Plots two nests that are used for "moved_nest" experiment
        function plot_new_nest(obj,n)
            if n == 1
                % plots nest two 'N2'
```

```
                text(obj.nest_two(1)-5, obj.nest_two(2), 'N2');
                rectangle('Position',[obj.nest_two(1)-10,obj.nest_two(2)-10,40,20]);
            elseif n == 2
                % plots nest one 'N1'
                obj.nest_two = [obj.nest(1)-5 obj.nest(2)];
                text(obj.nest(1)-5, obj.nest(2), 'N1');
                rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,40,20]);
            end
        end

        % Plots the regular nest
        function plot_nest(obj)
            text(obj.nest(1)-5, obj.nest(2), 'N');
            rectangle('Position',[obj.nest(1)-10,obj.nest(2)-10,20,20]);
        end

        % plots the feeder
        function plot_feeder(obj)
            text(obj.feeder(1)-5, obj.feeder(2), 'F');
            rectangle('Position',[obj.feeder(1)-10,obj.feeder(2)-10,20,20]);
        end
    end

end
```

A.6 Channel.m

```
classdef Channel < handle
    %CHANNEL Class
    %   Describes an n-leg channel

    properties
        n_of_legs;      % Number of legs
        entrance;       % Entrance coordinates
        exit;           % Exit coordinates
        nodes;          % Nodes of the channel
    end

    methods

        function obj = Channel(varargin)
            % Constructor
            % Provides inputs from 1 leg up to 3 legs Inputs are nodes
            if nargin == 4
                % Creates Channel with 2 nodes, e.g.: channel = ...
                    Channel(x1, y1, x2, y2);
                obj.n_of_legs = 1;
```

```
                obj.entrance = [varargin{1}, varargin{2}];
                obj.exit = [varargin{3}, varargin{4}];
                obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                    varargin{4}];
            elseif nargin == 6
                % Creates Channel with 3 nodes, e.g.: channel = ...
                    Channel(x1, y1, x2, y2, x3, y3);
                obj.n_of_legs = 2;
                obj.entrance = [varargin{1}, varargin{2}];
                obj.exit = [varargin{5}, varargin{6}];
                obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                    varargin{4}, varargin{5}, varargin{6}];
            elseif nargin == 8
                % Creates Channel with 4 nodes, e.g.: channel = ...
                    Channel(x1, y1, x2, y2, x3, y3, x4, y4);
                obj.n_of_legs = 3;
                obj.entrance = [varargin{1}, varargin{2}];
                obj.exit = [varargin{7}, varargin{8}];
                obj.nodes = [varargin{1}, varargin{2}, varargin{3}, ...
                    varargin{4}, varargin{5}, varargin{6}, varargin{7}, ...
                    varargin{8}];
            end
        end

        function plot(obj)
            % Plots the channel
            for i=1:2:2*obj.n_of_legs

                % Width of channel
                w = 10;
                % Direction vector
                v = [obj.nodes(i+2)-obj.nodes(i), ...
                    obj.nodes(i+3)-obj.nodes(i+1)];
                % Direction vector with length w
                vn = ...
                    w*(v/sqrt((obj.nodes(i+2)-obj.nodes(i))^2+(obj.nodes(i+3)-obj.nodes(i+1))^
                % Orthogonal vector to direction vector
                vr = ...
                    [-(obj.nodes(i+3)-obj.nodes(i+1)),obj.nodes(i+2)-obj.nodes(i)];
                % Orthogonal vector to direction vector with lenght w
                vrn = ...
                    w*(vr/sqrt((-obj.nodes(i+2)+obj.nodes(i))^2+(obj.nodes(i+3)-obj.nodes(i+1)

                % Wall points
                wp11 = [obj.nodes(i)+vn(1)+vrn(1), ...
                    obj.nodes(i+1)+vn(2)+vrn(2)];    % wall point 1.1
                wp12 = [obj.nodes(i+2)-vn(1)+vrn(1), ...
                    obj.nodes(i+3)-vn(2)+vrn(2)];  % wall point 1.2
                wp21 = [obj.nodes(i)+vn(1)-vrn(1), ...
                    obj.nodes(i+1)+vn(2)-vrn(2)];    % wall point 2.1
```

```
                wp22 = [obj.nodes(i+2)-vn(1)-vrn(1), ...
                    obj.nodes(i+3)-vn(2)-vrn(2)];  % wall point 2.2

                % Connect adjacent wall points
                line([wp11(1),wp12(1)],[wp11(2),wp12(2)], 'Color', [0,0,0]);
                line([wp21(1),wp22(1)],[wp21(2),wp22(2)], 'Color', [0,0,0]);

            end
            text(obj.nodes(1)-5,obj.nodes(2)+5,'E');
        end
    end
end
```

A.7 two_leg_trajectory.m

```
function epsilon = two_leg_trajectory( a, b, alpha, k, n_of_ants )
%TWO_LEG_TRAJECTORY
% Describes an experiment where the ant is forced to walk through a
% two-legged channel. At the end of the channel, it finds food and starts
% to head back for its nest. Due to the approximative way the ant keeps
% computing its mean direction (and thus the direction back to the nest),
% errors occur. These errors can be pointed out by manipulating the angle
% between the two elements of the channel.

% Output:
% epsilon: returns the error angle produced by the ant.
% Inputs:
% a, b: Lengths of the channel-legs.
% alpha: Angle between the two legs.
% k: Normalization constant used in the approximative computation.
% n_of_ants: Amount of ants that run the experiment with the given inputs.


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

    % Output
    epsilon = zeros(length(k),length(alpha),n_of_ants);

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

```

```
%%%%%%%%%% Main loops %%%%%%%%%%%%%%
for i=1:length(k) % Loops through the specified constants k
    for j=1:length(alpha) % Loops through the specified angles alpha
        % Create a new ant and place it to the nest
        ant = Ant(area.nest);

        % Channel
        rad = alpha(j)*(pi/180);    % Convert angle to radian
        % Create a channel with the given parameters
        channel = Channel(100, 100, 100, 100+a, ...
            100+cos(rad-(pi/2))*b, (100+a)-sin(rad-(pi/2))*b);
        % Place the feeder at the channel exit
        area.feeder = channel.exit;

        for ants=1:n_of_ants % Loops through the specified amount of ...
            ants
            % Set the constant k according to the input
            ant.k = k(i);
            % Temporary variables
            target = channel.entrance;
            done = 0;
            while ~done
                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.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
                            epsilon(i, j, ants) = ...
                                abs((ant.phi-pi)-eps_correct); % ...
                                Compute difference to the ant's ...
                                global vector
                            epsilon(i, j, ants) = epsilon(i, j, ...
                                ants)*(180/pi); % Convert to degrees
                        end
                    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
```

```
                        epsilon(ants) = abs((ant.phi-pi)-eps_correct); % ...
                            Compute difference to the ant's global vector
                        epsilon(ants) = epsilon(ants)*(180/pi); % ...
                            Convert to degrees
                    end
                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);
                    target = channel.entrance;
                    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
    end
end
```

A.9 one_leg_trajectory.m

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

```
% 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.

% Environment
area = Area(1000,1000);          % Create an area with size 10m^2
area.nest = [500, 50];           % Nest position
area.feeder = [420,500];         % Feeder position

% Ants
ant = Ant(area.nest);            % Create ant at nest
ant.random_params = [pi/4, 0.3]; % Random parameters

%%%%%%%%%% 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
    % Starting conditions
    ant.del_landmark(1);                         % Delete first landmark
    testing_channel = Channel(500,500,422,500);  % Set testing channel
    plotting_channel = testing_channel;          % Set plotting channel
    target = testing_channel.entrance;           % Set target
    done = 0;                                    % Ant not done yet

    while ~done
        cla;    % Clear axes

        % Foraging, ant goes to nest like trained
        if ant.status == 0
            if ant.move_to(target)
                if target == testing_channel.entrance;
                    % put landmark at channel entrance
```

```
                        ant.put_landmark();
                        target = testing_channel.exit;
                    elseif target == testing_channel.exit
                        target = area.feeder;
                    elseif target == area.feeder
                        % Ant walks back through new channel
                        plotting_channel = channel;
                        ant.landmarks(1).pos = channel.exit;
                        target = channel.entrance;
                    elseif target == channel.entrance
                        target = channel.exit;
                    elseif target == channel.exit
                        target = area.nest;
                        ant.status = 1;
                    end
                end

            % Returning to the nest
            elseif ant.status == 1
                 if ant.follow_local_v()
                       if ant.follow_global_v();
                           % Reset: Put ant to nest
                           ant.reset(area.nest);
                           target = [500,500];
                           done = 1;
                       end
                 end
            end

            % Plot
            area.plot_nest();
            area.plot_feeder();
            plotting_channel.plot();
            ant.plot();

            % Debugging: Different
            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);  % Pause 0.001 seconds
        end
    end
end
```

A.10 moved_nest.m

```matlab
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
```

```
            target = [area.cylinders(4) area.cylinders(10)];
        elseif a > 1/3 && a <= 2/3
            target = [area.cylinders(2)-80 area.cylinders(8)];
            ant{ants}.trail_color = 'red';
            red = 1;
        elseif a > 2/3
            target = area.feeder;
            ant{ants}.trail_color = 'green';
            green = 1;
        end

        done = 0;
        while ~done
            tic; % Start time measurement
            cla; % Clear axes

            % Foraging
            if ant{ants}.status == 0

%               %%%%%%%%%%% Trainings situation %%%%%%%%%%%%%
                if do_training % First targets gets reset.
                    if yes < 1
                        ant{ants}.pos = area.nest_two;
                        target = [ area.cylinders(6)+ 95 ...
                            area.cylinders(12) ];
                        yes = 1;
                    end
%                   % Ant goes through corridor in trainings situation
                    if ant{ants}.move_to(target)
                        if target == [area.cylinders(6)+ 95 ...
                            area.cylinders(12) ];
                            % The ants on the 'green' trail dont always ...
                                follow the local vector
                            % Therefore some landmarks aren't put while ...
                                trainings mode
                            if green ~= 1;
                                ant{ants}.put_landmark_at(area.cylinders(6,:));% ...
                                    Ant stores landmark at cylinder position
                                n = length(ant{ants}.landmarks);
                                ant{ants}.landmarks(n).range = 115; % ...
                                    Determines the Range of the landmark.
                            end
                            target = [area.cylinders(5)- 140 ...
                                area.cylinders(11)]; % next target gets ...
                                chosen.
                        elseif target == [area.cylinders(5)- 140 ...
                            area.cylinders(11)]
                            ant{ants}.put_landmark_at(area.cylinders(5,:) ...
                                + 20);
                            n = length(ant{ants}.landmarks);
```

```
                    ant{ants}.landmarks(n).range = 150;
                    target = [area.cylinders(4) + 85 ...
                        area.cylinders(10) ];
                elseif target == [area.cylinders(4) + 85 ...
                    area.cylinders(10)]
                        ant{ants}.put_landmark_at(area.cylinders(4,:) ...
                            + 20);
                        n = length(ant{ants}.landmarks);
                        % Range differs randomly for some Cylinders
                        if green ~= 1
                         ant{ants}.landmarks(n).range = 130;
                        else
                            ant{ants}.landmarks(n).range = 70;
                        end
                    target = [area.cylinders(3) - 75 ...
                        area.cylinders(9)];
                elseif target == [area.cylinders(3) - 75 ...
                    area.cylinders(9)]
                    ant{ants}.put_landmark_at(area.cylinders(3,:) ...
                        + 20);
                    n = length(ant{ants}.landmarks);
                    ant{ants}.landmarks(n).range = 90;
                    target = [area.cylinders(2) + 100 ...
                        area.cylinders(8)];
                elseif target == [area.cylinders(2) + 100 ...
                    area.cylinders(8)]
                    ant{ants}.put_landmark_at(area.cylinders(2,:) ...
                        + 20)
                    n = length(ant{ants}.landmarks);
                    ant{ants}.landmarks(n).range = 130;
                    target = [area.cylinders(1) - 85 ...
                        area.cylinders(7)];
                elseif target == [area.cylinders(1) - 85 ...
                    area.cylinders(7)]
                    ant{ants}.put_landmark_at(area.cylinders(1,:) ...
                        + 20)
                    n = length(ant{ants}.landmarks);
                    ant{ants}.landmarks(n).range = 110;
                    target = area.feeder;
                elseif target == area.feeder;
                     do_training = 0; % Finishes trainings situation
                     ant{ants}.reset(area.nest); % Ant are reset
                end
            end


        %%%%%%%%%%% Foraging from second nest %%%%%%%%%%%%%%%%%%
        else
            if ant{ants}.move_to(target)
                if target == [area.cylinders(4), area.cylinders(10)]
```

```
                target = [area.cylinders(2)+ 50 ...
                    area.cylinders(8)];
            elseif target == [area.cylinders(2)+ 50 ...
                area.cylinders(8)]
                target = area.feeder();
            elseif target == [area.cylinders(2)−80 ...
                area.cylinders(8)]
                target = area.feeder();
            elseif target == area.feeder();
                ant{ants}.status = 1;
            end
        end
    end
% Returning to Nest
elseif ant{ants}.status == 1
    if red == 1;
        if ant{ants}.follow_local_v();
            if  ~ant{ants}.follow_global_v()
            else
                ant{ants}.reset(area.nest);
                ant{ants}.landmarks = [];
                done = 1;
                do_training = 1;
                yes = 0;
            end
        end
    elseif green == 1;
        if ant{ants}.follow_local_v();
            if  ~ant{ants}.follow_global_v()
            else
                ant{ants}.reset(area.nest);
                ant{ants}.landmarks = [];
                done = 1;
                do_training = 1;
                yes = 0;
            end
        end
    else
       if ~ant{ants}.follow_global_v()
       else
                ant{ants}.reset(area.nest); % Ants are reset
                ant{ants}.landmarks = []; % Landmarks are reset
                done = 1;
                do_training = 1; % Trainings mode is ...
                    inserted for next ant
                yes = 0;
       end
    end
end

```

```
            %Plot
            area.plot_landmark_route;
            area.plot_new_nest(1);
            area.plot_new_nest(2);
            area.plot_feeder();
            area.plot_cylinders();
            for i=1:n_of_ants
                if ant{i} ~= 0
                    ant{i}.plot();
                end
            end
            area.nest_two = [800, 1350];

            pause(0.001);
        end
    end
end
```

A.11 crooked_corridor.m

```
function crooked_corridor(n_of_ants)
% Describes an experiment where the ant is forced to walk through a
% a corridor of black cylinders and return back to their nest.
% The difference here is, that the corridor is crooked about 45 .
% The trails are recorded to watch the ants behaviour under influence of
% the visual landmarks.

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

    % Environment
    area = Cylinder_Area(1500,1500,1300,300,6); % Create an area with ...
        size 15m^2 and constructs the Cylinders
    area.nest = [1200, 30];
    area.feeder = [100, 1200];

    %%%%%%%% Crook the Corridor %%%%%%%%%
    i = area.amount;
    j = area.amount*2;
    % Cylinders are shifted to built a crooked corridor
    while i > 0
        area.cylinders(i) = area.cylinders(i) - 160*(i);
        area.cylinders(j) = area.cylinders(j) - 90;
        i = i - 1;
        j = j - 1;
    end
    u = 6;
```

```
    v = 12;

    while u > 1
        area.cylinders(u) = area.cylinders(u) + 30;
        area.cylinders(v) = area.cylinders(v) - 85;
        v = v - 2;
        u = u - 2;
    end

    % Ants
    ant = Ant(area.nest);
    ant.random_params = [pi/3, 0.3];
    ant.k = 0.13;

    %%%%%%%%%% 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
        % Generates a random number in the intervall [-60,120]
        % Is used to determine how close the ant approaches to Cylinder.
        a = -60;
        b = 120;
        rand_n = a + (b-a).*rand(1);

        % After starting from the nest, which nearby cylinder gets chosen
        c = rand;
        % (Chances 50%)
        if c > 0.5
            target = [area.cylinders(1)-rand_n area.cylinders(7)];
        else
            target = [area.cylinders(2)-rand_n area.cylinders(8)];
        end

        done = 0;
           while ~done
                tic; % Start time measurement
                cla; % Clear axes
                % Foraging
                if ant.status == 0
                    if ant.move_to(target)
                        if target == [area.cylinders(1)-rand_n ...
                            area.cylinders(7)]
                            ant.put_landmark_at(area.cylinders(1,:)+ 20);
                            n = length(ant.landmarks);
```

```
                    ant.landmarks(n).range = 110; % Determines ...
                        the Range of the landmark.
                    rand_n = a + (b-a).*rand(1);  % Generates ...
                        new approach range( how close to come to ...
                        the cylinder.)
                    % Decides with a chance of 30% which of the ...
                        3 next cylinder to approach
                    c = rand;
                    if c <= 1/3
                        target = [area.cylinders(2) + rand_n ...
                            area.cylinders(8)];
                    elseif c > 1/3 && c <= 2/3
                        target = [area.cylinders(3) - rand_n ...
                            area.cylinders(9)];
                    else
                        target = [area.cylinders(4) + rand_n ...
                            area.cylinders(10)];
                    end
                elseif target == [area.cylinders(2) - rand_n ...
                    area.cylinders(8)]
                    ant.put_landmark_at(area.cylinders(2,:)+ 20);
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110; % Determines ...
                        the Range of the landmark.
                    rand_n = a + (b-a).*rand(1);
                    target = [area.cylinders(4) + rand_n ...
                        area.cylinders(10)];
                elseif target == [area.cylinders(2) + rand_n ...
                    area.cylinders(8)]
                    ant.put_landmark_at(area.cylinders(2,:)+ 20);
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110;
                    rand_n = a + (b-a).*rand(1);
                    % Decides with a chance of 25%/75% next ...
                        Cylinder to approach
                    % Cylinder which is more far away, has less ...
                        chance
                    c = rand;
                    if c <= 1/4
                        target  = [area.cylinders(3)-rand_n ...
                            area.cylinders(9)];
                    else
                        target = [area.cylinders(4)+rand_n ...
                            area.cylinders(10)];
                    end
                elseif target == [area.cylinders(3)-rand_n ...
                    area.cylinders(9)]
                    ant.put_landmark_at(area.cylinders(3,:) + 20);
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110;
```

```
                    rand_n = a + (b-a).*rand(1);
                    % Decides with a chance of 25%/75% next approach
                    c = rand;
                    if c <= 1/4
                        target = [area.cylinders(4)+rand_n ...
                            area.cylinders(10)];
                    else
                        target = [area.cylinders(5) - rand_n ...
                            area.cylinders(11)];
                    end
                elseif target == [area.cylinders(4)+rand_n ...
                    area.cylinders(10)]
                    ant.put_landmark_at(area.cylinders(4,:) + 20);
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110;
                    rand_n = a + (b-a).*rand(1);
                    % Decides with a chance of 25%/75% next approach
                    c = rand;
                    if c <= 1/4
                        target = [area.cylinders(5) - rand_n ...
                            area.cylinders(11)];
                    else
                        target = [area.cylinders(6) + rand_n ...
                            area.cylinders(12)];
                    end
                elseif target == [area.cylinders(5) - rand_n ...
                    area.cylinders(11)]
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110; % Determines ...
                        the Range of the landmark.
                    ant.put_landmark_at(area.cylinders(5,:) + 20);
                    rand_n = a + (b-a).*rand(1);
                    % Decides with a chance of 25%/75% next approach
                    if c <= 1/4
                        target = [area.cylinders(6) + rand_n ...
                            area.cylinders(12)];
                    else
                        target = area.feeder;
                    end
                elseif target == [area.cylinders(6) + rand_n ...
                    area.cylinders(12)]
                    n = length(ant.landmarks);
                    ant.landmarks(n).range = 110; % Determines ...
                        the Range of the landmark.
                    ant.put_landmark_at(area.cylinders(6,:) + 20);
                    target = area.feeder;
                elseif target == area.feeder
                    ant.status = 1;
                end
            end
```

```
                % Returning to Nest
                elseif ant.status == 1
                    if ant.follow_local_v();
                        if  ~ant.follow_global_v()
                        else
                            ant.reset(area.nest);
                            ant.landmarks = [];
                            done = 1;
                        end
                    end
                end
                % Plot
                area.plot_nest();
                area.plot_feeder();
                area.plot_cylinders();
                ant.plot();

                pause(0.001);
            end
    end
end
```

A.12 corridor_of_black_cylinders.m

```
function corridor_of_black_cylinders(n_of_ants)
% Describes an experiment where the ant is forced to walk through a
% a corridor of black cylinders and return back to their nest.
% The trails are recorded to watch the ants behaviour under influence of
% the visual landmarks.

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

    % Environment
    area = Cylinder_Area(1500,1500,850,280,6); % Create an area with ...
        size 15m^2 and constructs the Cylinders
    area.nest = [750, 30];
    area.feeder = [650, 1480];
    % Ants
    ant = Ant(area.nest);
    ant.random_params = [pi/4, 0.3];
    ant.k = 0.08;

    %%%%%%%%%% 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
        % Generates a random number in the intervall [-60,120]
        % Is used to determine how close the ant approaches to Cylinder.
        a = -60;
        b = 120;
        rand_n = a + (b-a).*rand(1);

        % After starting from the nest, which nearby cylinder gets chosen
        c = rand;

        % (Chances 50%)
        if c > 0.5
            target = [area.cylinders(1)-rand_n area.cylinders(7)];
        else
            target = [area.cylinders(2)-rand_n area.cylinders(8)];
        end

        done = 0;
            while ~done
                tic; % Start time measurement
                cla; % Clear axes
                % Foraging
                if ant.status == 0
                    if ant.move_to(target)
                        if target == [area.cylinders(1)-rand_n ...
                            area.cylinders(7)]
                            ant.put_landmark_at(area.cylinders(1,:)+ ...
                                20); % Ant stores landmark at cylinder ...
                                position
                            n = length(ant.landmarks);
                            ant.landmarks(n).range = 110; % Determines ...
                                the Range of the landmark.
                            rand_n = a + (b-a).*rand(1); % Generates new ...
                                approach range( how close to come to the ...
                                cylinder.)
                            % Decides with a chance of 30% which of the ...
                                3 next cylinder to approach
                            c = rand;
                            if c <= 1/3
                                target = [area.cylinders(2) + rand_n ...
                                    area.cylinders(8)];
                            elseif c > 1/3 && c <= 2/3
                                target = [area.cylinders(3) - rand_n ...
                                    area.cylinders(9)];
                            else
```

```
                    target = [area.cylinders(4) + rand_n ...
                        area.cylinders(10)];
                end
            elseif target == [area.cylinders(2) - rand_n ...
                    area.cylinders(8)]
                ant.put_landmark_at(area.cylinders(2,:)+ 20);
                n = length(ant.landmarks);
                ant.landmarks(n).range = 110;
                rand_n = a + (b-a).*rand(1);
                target = [area.cylinders(4) + rand_n ...
                    area.cylinders(10)];% next target gets ...
                    chosen.
            elseif target == [area.cylinders(2) + rand_n ...
                    area.cylinders(8)]
                ant.put_landmark_at(area.cylinders(2,:)+ 20);
                n = length(ant.landmarks);
                ant.landmarks(n).range = 110;
                rand_n = a + (b-a).*rand(1);
                % Decides with a chance of 25%/75% next ...
                    Cylinder to approach
                % Cylinder which is more far away, has less ...
                    chance
                c = rand;
                if c <= 1/4
                    target  = [area.cylinders(3)-rand_n ...
                        area.cylinders(9)];
                else
                    target = [area.cylinders(4)+rand_n ...
                        area.cylinders(10)];
                end
            elseif target == [area.cylinders(3)-rand_n ...
                    area.cylinders(9)]
                ant.put_landmark_at(area.cylinders(3,:) + 20);
                n = length(ant.landmarks);
                ant.landmarks(n).range = 110;
                rand_n = a + (b-a).*rand(1);
                % Decides with a chance of 25%/75% next approach
                c = rand;
                if c <= 1/4
                    target = [area.cylinders(4)+rand_n ...
                        area.cylinders(10)];
                else
                    target = [area.cylinders(5) - rand_n ...
                        area.cylinders(11)];
                end
            elseif target == [area.cylinders(4)+rand_n ...
                    area.cylinders(10)]
                ant.put_landmark_at(area.cylinders(4,:)+ 20);
                n = length(ant.landmarks);
                ant.landmarks(n).range = 110;
```

```
                        rand_n = a + (b-a).*rand(1);
                        % Decides with a chance of 25%/75% next approach
                        c = rand;
                        if c <= 1/4
                            target = [area.cylinders(5) - rand_n ...
                                area.cylinders(11)];
                        else
                            target = [area.cylinders(6) + rand_n ...
                                area.cylinders(12)];
                        end
                    elseif target == [area.cylinders(5) - rand_n ...
                        area.cylinders(11)]
                        ant.put_landmark_at(area.cylinders(5,:)+ 20);
                        n = length(ant.landmarks);
                        ant.landmarks(n).range = 110;
                        rand_n = a + (b-a).*rand(1);
                        % Decides with a chance of 25%/75% next ...
                            trajecetory next approach
                        if c <= 1/4
                            target = [area.cylinders(6) + rand_n ...
                                area.cylinders(12)];
                        else
                            target = area.feeder;
                        end
                    elseif target == [area.cylinders(6) + rand_n ...
                        area.cylinders(12)]
                        ant.put_landmark_at(area.cylinders(6,:) + 20);
                        n = length(ant.landmarks);
                        ant.landmarks(n).range = 110;
                        target = area.feeder;
                    elseif target == area.feeder
                        ant.status = 1;
                    end
                end
            % Returning to Nest
            elseif ant.status == 1
                if ant.follow_local_v();
                    if ~ant.follow_global_v()
                    else
                        ant.reset(area.nest); % Ants are reset
                        ant.landmarks = []; % Landmarks are reset
                        done = 1;
                    end
                end
            end
            % Plot
            area.plot_nest();
            area.plot_feeder();
            area.plot_cylinders();
            ant.plot();
```

```
                    pause(0.001);
                end
        end
    end
```

A.13 natural_foraging.m

```
function reached_nest = natural_foraging(n_of_ants, lm_distance)
    % NATURAL FORAGING
    % This experiment recreates roughly a natural foraging sitation where
    % all the implemented navigational modules 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 feeder
    % site is put on a random place. The ant picks random points on the
    % area and moves to them. As soon as it gets near to the feeder, its
    % walks towards the feeders 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, which additionally help the
    % ant when moving back to the nest.

    % Output:
    % reached_nest: Whether or not the ant found the way back to the nest

    % Input:
    % n_of_ants: Amount of ants that run the test
    % lm_distance: Landmark distance. Interval at which the ant sets
    % landmarks

    % Output
    reached_nest = zeros(1,n_of_ants);

    % Environment
    area = Area(1000,1000); % Create an area with size 10m^2
    area.nest = [500, 500];

    %%%%%%%%%% 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 % Loops through the specified amount of ants

```

```
        % Put the feeder at a randomly generated position between 100 and
        % 900 on the area
        area.feeder = [randi(800)+100, randi(800)+100];
        % Create an ant and put it at the nest
        ant = Ant(area.nest);
        % Create a special landmark (so that the ant later on 'sees' the
        % nest from a certain distance (the landmarks' range) and can
        % return to it
        ant.put_landmark_at(area.nest);
        ant.landmarks(1).length = 0;
        ant.landmarks(1).range = 80;

        % Create special landmark for the feeder site
        ant.put_landmark_at(area.feeder);
        ant.landmarks(2).length = 0;
        ant.landmarks(2).range = 80;

        % Set initial random target
        target = [randi(1000), randi(1000)];
        % Set parameters for random walk
        ant.random_params = [pi/4, 0.3];
        % Temporary variables
        done = 0;
        % Variable that holds the distance the last landmark was set
        put_after = 0;

        % Main loop
        while ~done
            cla; % Clear axes

            % Phase 1: Foraging
            ant.show_trail = 1;
            if ant.status == 0
                if max(ismember(ant.within_landmark, ant.landmarks(2)))
                    % The ant has crossed the feeder landmark and walks
                    % now towards the center
                    if ant.move_to(ant.landmarks(2).pos)
                        % The ant collected food and returns back to the
                        % nest
                        ant.status = 1;
                    end
                else
                    if abs(ant.l-put_after) > lm_distance
                        % The ant covered the required distance to set a
                        % landmark
                        put_after = ant.l;
                        ant.put_landmark();
                    end
                    % The ant walks towards its target
                    if ant.move_to(target)
```

```
                        % Arrived at target, set new random target
                        target = [randi(1000), randi(1000)];
                    end
                end

            % Phase 2: Returning to the nest
            elseif ant.status == 1
                if max(ismember(ant.within_landmark, ant.landmarks(1)))
                    % The ant has crossed the nest landmarks and walks now
                    % towards the center
                    if ant.move_to(ant.landmarks(1).pos)
                        % The ant arrived at the nest and completed the run
                        done = 1;
                        reached_nest(ants) = 1;
                    end
                else
                    % The ant first follows local vectors on its way home
                    if ant.follow_local_v()
                        % If the local vectors are used up, it follows the
                        % global vector
                        if ant.follow_global_v()
                            % The ant walked back its calculated covered
                            % distance but did not return at the nest.
                            % The and completed the run.
                            done = 1;
                            reached_nest(ants) = 0;
                        end
                    end
                end
            end

            % Plot
            ant.plot();
            area.plot_nest();
            area.plot_feeder();

            % 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
        % Reset the ants' properties and put it back to the nest for the
        % next run
        ant.reset(area.nest);
    end
end
```