ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Desert ant navigational behaviour (Cataglyphis Fortis)

Florian Hasler, Matthias Heinzmann,
Andreas Urech, Dominik Werner

Zürich
December 2015

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.

Florian Hasler

Matthias Heinzmann

Andreas Urech

Dominik Werner

Inhaltsverzeichnis

1 Abstract **1**

2 Individual contributions **2**

3 Introduction and Motivations **2**

4 Description of the Model **3**
4.1 Orientation . . . 3
4.1.1 Pathintegration . . . 3
4.2 Local landmark Orientation . . . 3
4.3 Combination . . . 4

5 Implementation **5**
5.1 UML-Diagram . . . 5
5.2 Source Code . . . 6
5.2.1 run.m . . . 7
5.2.2 distanceBetweenTwoPoints.m . . . 7
5.2.3 vector2angle.m . . . 7
5.2.4 updateGround.m . . . 7
5.2.5 Landmark.m . . . 8
5.2.6 Ground.m . . . 8
5.2.7 Ant.m . . . 8

6 Simulation Results and Discussion **9**
6.1 Discussion of the Pathintegrator . . . 9
6.2 Discussion of the ant's random walk . . . 11
6.3 Verification of the Results . . . 12

7 Summary and Outlook **13**

A Appendix **i**
A.1 run.m . . . i
A.2 distanceBetweenTwoPoints.m . . . iv
A.3 vector2angle.m . . . iv
A.4 updateGround.m . . . iv
A.5 Landmark.m . . . v
A.6 Ground.m . . . vi
A.7 Ant.m . . . vii

B Correspondence **xiii**

C References **xiv**

1 Abstract

The Saharan Desert ant (Cataglyphis Fortis) lives under harsh environmental conditions which make a sophisticated way of orientation necessary.
German biologist Wehner [1],[2], [3] has proposed two such means. Namely path integration and orientation by landmarks. In his final conclusion he considers both methods important and assumes that the ant has a way of adopting the priority of each mechanism according to the current situation.

In previous years a group called gordonteam[5] already implemented a model based on the before mentioned articles. In their report they noted that their path integration did not work and that they were not able to combine all methods. Our plan is to extend upon their work and fix the problem with the path integrator as well as to bind the two methods together.

2 Individual contributions

We split up the work among the members of our group based upon their past experience in various topics and skills needed to develop the project. In order for every member to understand the project in its entirety and can make additions if needed we had meetings once a week where every member presented his work to the rest of the group. In those meetings we also discussed how to proceed with the project. We divided the work as follows:

- Path integrator: Florian Hasler, Matthias Heinzmann
- Landmarks: Matthias Heinzmann
- Multi foodsource support: Dominik Werner
- Software Architecture: Florian Hasler, Matthias Heinzmann,Dominik Werner
- Timer: Andreas Urech
- Ant looking around behaviour: Florian Hasler
- Project report: Andreas Urech, Florian Hasler, Matthias Heinzmann, Dominik Werner
- LaTeX- version of report and presentation: Andreas Urech

3 Introduction and Motivations

Due to the sparse Saharan resources the ants have to use them economically. Due to this fact the weight of the ant's brain is only around 0.1mg. It is not possible for a brain this small to execute calculations of high complexity. They have to rely on simple but efficient ways to be able to keep track of their individual position. Our goal of implementing a model of the ants navigation system will hopefully show if the proposed model can reproduce the behavior observed in real ants. Our main question is:
Is the path integrator-model proposed by Wehner1988 [2] to describe the ant's behaviour in one specific scenario applicable for more general situations?
Further questions are:

- Are we able to predict what happens if we alter the environment? (depends on the question beforehand)
- Can the ants survive if we rid the environment completely of any landmarks?

4 Description of the Model

4.1 Orientation

As mentioned the desert ant's main means of orientation are:

- pathintegration
- local landmark orientation

The cataglyphis doesn't communicate with its conspecifics on a foraging walk. This is because the prey usally consists of small parts of insects, that can be carried by one ant, and no need for sharing information arises. [Appendix B]

4.1.1 Pathintegration

No matter the zig-zag way out of its nest, the ant is able to return in a more or less straight line. This remarkable ability is reached by pathintegration. The ant iteratively computes the distance l to the next L ant the angle φ between the vector pointing from the nest to the ant's position and a predefined basis vector. Therefore the ant is always aware in which direction its nest is located. These calculations are executed with imperfections which accumulate with growing path length. In extreme cases pathintegration causes the ant to miss its nest and other means of orientation are necessary.
In Wehner1988[2][1] the following formulas have been derived, which give a rough approximation of the ants pathintegration.

$$\varphi_(n+1) = \varphi(n) + k \cdot \frac{(\pi+\delta)\cdot(\pi-\delta)\cdot\delta}{l(n)}$$
$$l(n+1) = l(n) + 1 - \frac{|\delta|}{\pi}$$

where $k = 0.1316$ is a fitting constant Wehner1988 [2], δ is the angle with which the ant is turning its current direction and the step width is assumed to be 1.

4.2 Local landmark Orientation

Desert ants can associate familiar landmarks with a local vector, which can lead them directly from one place to the visual catchment area of the next one (Wehner2003

[1] on page 5288

[1]).
Therefore, as soon as an ant recognizes a landmark it knows where to go to reach the next landmark or the nest respectively given that it has already familiarized with this landmark. For our implementation, we simplified this ability, i.e. the local vector, in the following way. As soon as the ant sees a landmark it goes there. When arrived, the ant walks N steps in a direction d where N and d are determined by the landmark's properties. Like that, the local vector is implicitly defined by N and d. The ant reaches the next landmark or the nest respectively after having walked along the local vector. For simplicity's sake, we assumed that all landmarks are familiar to all ants at the start of our simulation.

4.3 Combination

The results gained R. Wehner's experiments indicate that ants use predominantly pathintegration, but if available using local landmarks is more preferable to using the global vector-navigation via pathintegration. It has been concluded that the ant transiently inhibits the global vector when being in familiar territory. The global vector only reemerges after the local vector ceases. Whilst navigating via the local vector the global vector is continuously updated.

```
Algorithm ReturnToMyNest()
    while not at nest do
        execute global vector;
        update global vector;
        if local vector recognised then
            while local vector > 0 do
                execute local vector;
                update local vector;
                update global vector;
            end
        end
    end
    return
```

Algorithm 1: Returning to the nest

For our simulation we will use an agent based model where every agent will be representing an ant. To be able to compare our model with the real world data we have to keep track of several different variables:

nestLocation: This is a 2D vector which indicates the position of the nest inside a sand pan.

foodSourceLocation(s): These 2D vectors symbolize the location of a food source which the ant will have to travel to. It should be possible to implement an ephemeral (non-renewable) food source so the ant has to look for a new one if the first does not yield food anymore.

ants: The agents themselves who contain their position data and other information to track the simulation.

landmarks: Objects that symbolize visual orientation points for the ants.

5 Implementation

To implement our agent based model we decided to use object orientated programming because it seemed more intuitive than a procedural approach in this case.

5.1 UML-Diagram

The following graphic shows the UML-Class diagram that describes the data model of our project. The files run.m, updateGround.m, vector2angle.m and distanceBetweenTwoPoints.m are not part of this diagram since they are procedural functions and not classes.
This diagram shows the classes of the project and how they are related. Since Matlab does not support information hiding the public/private information is missing in the diagram. Because Matlab uses matrices in many cases all data-types with the exceptions of bool, string, Ant, Ground and Landmark are matrices of type double [2] as entries:

- vector2: double(2,1)
- vector3: double(3,1)
- vector2[]: double(2, n)
- number: double(1,1)

[2] Mathworks2015[4]

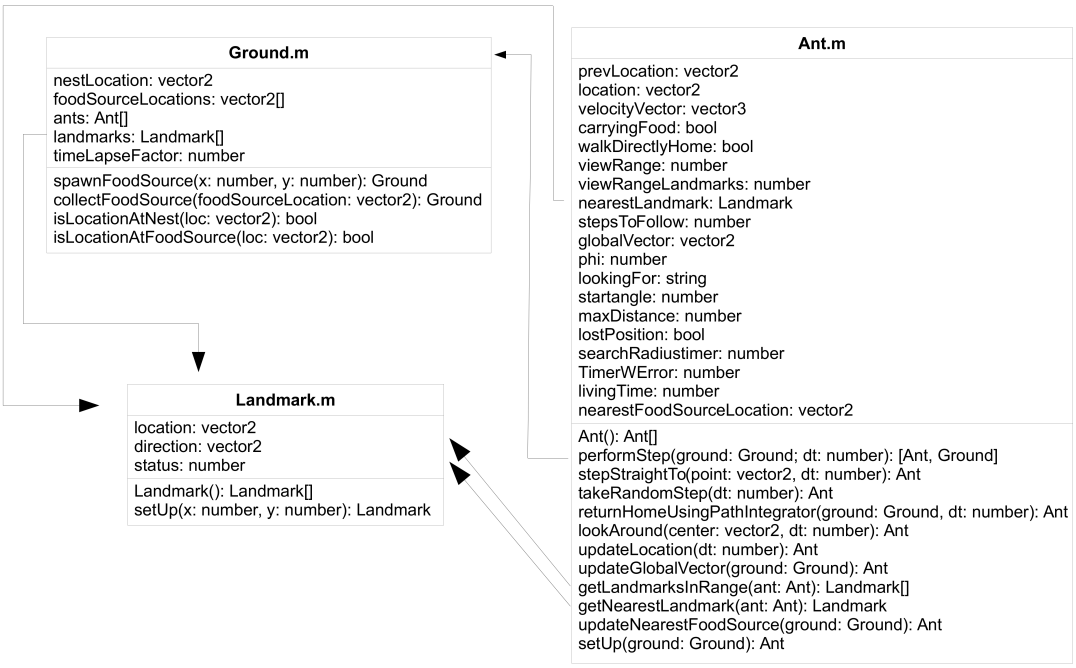

Figure 1: UML

Something else to be mentioned is that Matlab does not make use of pointers but instead copies instances of objects and does not alter the original. This means that in order for a method that is supposed to alter an object the original object has to be overwritten. This can be done by letting methods return the object that is containing them (this).

```
ground.spawnFoodSource(xCoord(k),yCoord(k));          % WRONG: Has no effect
ground=ground.spawnFoodSource(xCoord(k),yCoord(k));   % CORRECT: Update gets written
```

Listing 1: copies

For this reason nearly every method does return the object it belongs to as a result.

5.2 Source Code

To shed some light on how we implemented the model we will quickly describe the source code. The object orientated approach made it possible to implement new ideas relatively fast at the correct location.

5.2.1 run.m

[Listing 2 in A.1].This file does all the parametrization and executes the setup methods of the various classes utilized in this project. It is supposed to be used as an easy way to alter the setup if the need arises.
The file contains the before mentioned setup section, followed by the execution section where the main-loop does update the ground and the ants objects. Also inside of the main-loop one can find the time handling. At the beginning of each iteration a timestamp is saved and then used at the end of the iteration to pause the execution in order to have a uniform speed throughout the running time of the program.
The last section of the file does handle callback-functions from the user interface. The user interface in our case is the plot that opens and displays the various agents moving in the world. The user interface provides the following functions:

- **Spawn new food source:** New sources of food for the ants to collect can be spawned by simply clicking at the location in the world were the source should appear.
- **Close:** In order for the program to terminate without errors we used a callback-function from closing event of our plot-figure to terminate the program.

5.2.2 distanceBetweenTwoPoints.m

[Listing 3 in A.2]. A simple helper function which calculates the distance between two points.

5.2.3 vector2angle.m

[Listing 4 in A.3]. This helper function calculates the angle of a given vector in radians. The value of the result ranges from 0 to 2 pi. This is important in order for the path integrator to work correctly. Additional corrections (to use the acute and not obtuse angle) are done in Ant.m.

5.2.4 updateGround.m

[Listing 5 in A.4]. This function does not change anything on the Ground object as its name might suggest. This method does update the plot and redraws all ants, food sources, the nest and the landmarks.

5.2.5 Landmark.m

[Listing 6 in A.5]. This class represents a visual landmark. It is defined as having a location, a direction vector and a status which is represented by a number. The direction vector is used by incoming ants to leave the landmark in the direction provided by the landmark. This is supposed to mimic the ants procedural memory.

5.2.6 Ground.m

[Listing 7 in A.6]. This class serves as container for all agents and the objects they can interact with. Besides its function as a container it also provides methods to spawn new food sources and landmarks in the world as well as various methods which can determine the proximity of a certain target.

5.2.7 Ant.m

[Listing 8 in A.7].This class represents our agent and implements the mathematical model proposed by R. Wehner [2]. The model is implemented pretty straight forward with a few exceptions. Because our ants aren't real and do not have a limited field of view but can see 360 degrees we had to create a property which tells the ant if it is leaving the nest so it doesn't immediately go back to the nest after it left it. This is due to the assumption that foraging ants that come across their nest during their searches, reset their global vector to correct errors or in other words, they will use the local vector rather than the global vector (as described in 4.3) . Furthermore we have a much more mathematical problem when we are near the nest. Because we divide by the length of our global vector we cause a division by zero in our update method once the global vectors length becomes zero. To avoid this we actually wait until the nest is outside of the ants view distance and set the length of the global vector to the view distance. It is reasonable to assume that the real ants do something similar because of the before mentioned local vector.

Also important is the method that takes a random step. The math behind this method is discussed in detail in another chapter here we just describe the implementation. The implementation is basically the math plus the calling of the update method to set the ants position after the calculation is done.

As described above the ants vision was simplified for the implementation and a field of view is not implemented. Once the ant is within view distance of an object (food source, landmark, nest) it steps straight towards it (local vector). The decision if the ant has reached its target is made by checking the length of the difference vector between the ant's location and the target. Once the length is smaller than

a certain threshold value the ant has reached its target. Since it is possible to have many different food sources as well as landmarks we implemented methods which update the properties nearestFoodSourceLocation and nearestLandmark. Those two properties are then used for the comparison. It is also important to note that the landmarks are visible from further away than food sources or the nest.

Real ants cannot stay outside for too long because they would die in the heat of the desert. We implemented a timer that every ant updates (with a small, random error) which tells the ant when it has to return to the nest during its search for food. If an ant does not reach the nest in time it will stop moving thus being dead.

6 Simulation Results and Discussion

6.1 Discussion of the Pathintegrator

The general idea of the path integrator is described in section 4.1.1. In the following we will discuss the quality of this specific path integrator model. Let's assume that the ant is currently at a position $l = L$ and $\varphi = 0$ (we can assume $\varphi = 0$ without loss of generality because if $phi \approx 0$ we would simply rotate our coordinate system to obtain $\varphi' = 0$ again). Furthermore, the global vector at this point is assumed to be perfect, which means that it directly points from the nest to the ant's position.
Now the ant does one step with a specific step width l. The direction is given by the turning angle delta which is assumed to be normal distributed with expectation value μ and standard deviation $\sigma = \frac{\pi}{16}$ where μ corresponds to the direction the ant has been walking in the previous step. Of course, the path integrator critically depends on this μ as can be seen in figure 3 (plots for $\mu = 0, 40, 80, 120°$). In figure 3 , the first row illustrates the absolute expected error of the path integrator while the second row illustrates the expected error relative to the step width l.

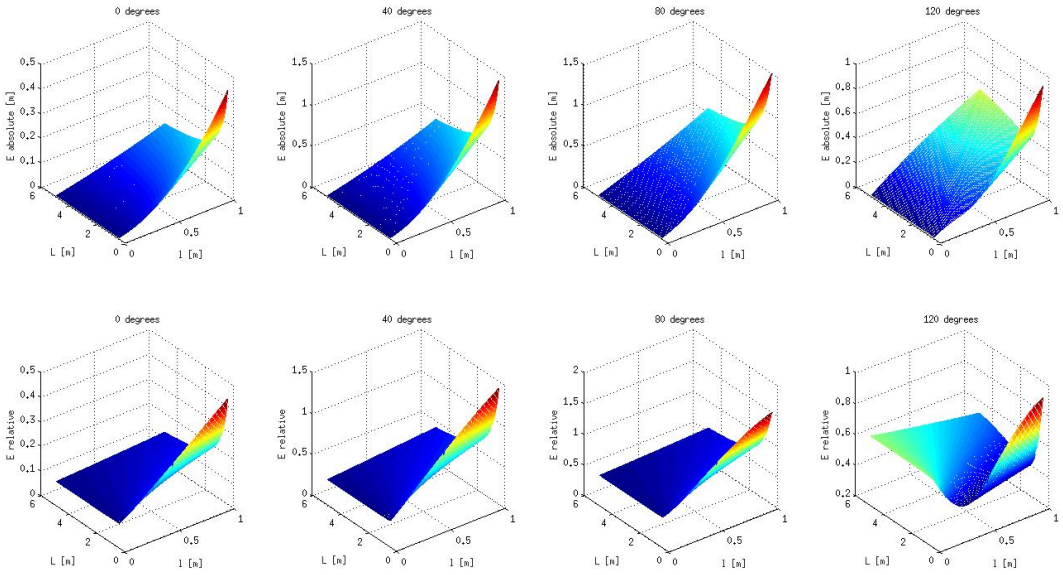

Figure 2: Pathintegrator error

One can easily observe figure 2 that the expected error is quite low for small turning angles delta but increases drastically for bigger turning angles. Thus, if the ant 'wants' to be safely lead back to the nest by using this path integrator model the expected value of the distribution mu should always be near 0. This means that in principle the ant would have to walk radially out of the nest. There are three possibilities to solve this issue:

- Deserts ants do actually walk radially out in principle. Here, 'in principle' means that they can walk to the left or to right respectively but afterwards they have to walk in the other direction which leads to a zig-zag walk around a path pointing radially out of the nest.
- Desert ants can't find home without the use of landmarks to a relatively large amount. One would have to do further experiments to prove or disprove these two possibilities. If they are wrong, we consequently have to consider a third and last possibility.
- The path integrator model has to be extended for that it works in the situation we wanted to apply it in our project.

6.2 Discussion of the ant's random walk

To improve the performance of the path integrator as well as to make to walk of the ant more detailed one would be able to vary the step width l. But by varying l we end up with a new problem. The path the ant walks along is strongly influenced by the choice of l. What we would like to have though is that the ant walks in a similar way no matter what the step size is. To address this problem we set the variance in the distribution of the turning angle ϑ in relation to dt (step width l of the ant is determined by l = dt*antSpeed) . We made the following ansatz: $\sigma(dt) = dt^c \cdot \sigma_0$ with $c \in (0, 1]$ and σ_0 the standard deviation for $dt = 1$. Then we let the ant do $n1 = 1$ steps with $dt = 1$ and $n2 = \frac{1}{dt}$ steps with some $dt \in (0, 1]$.Finally we correlated the resulting positions of the two situations and looked for the best correlation, i.e. c such that the difference between the two situations is lowest. The result for a normal distribution with $\mu = 0$ and sigma0 = pi/12 and dt = 0.1 can be seen in the figure below. Afterwards, we tried out several different $dt's$ and $\sigma_0's$ but the result for c was always similar to the one in the left side of the figure. The value for the best fitting c can be seen in figure 3 as being approximately 0.3.

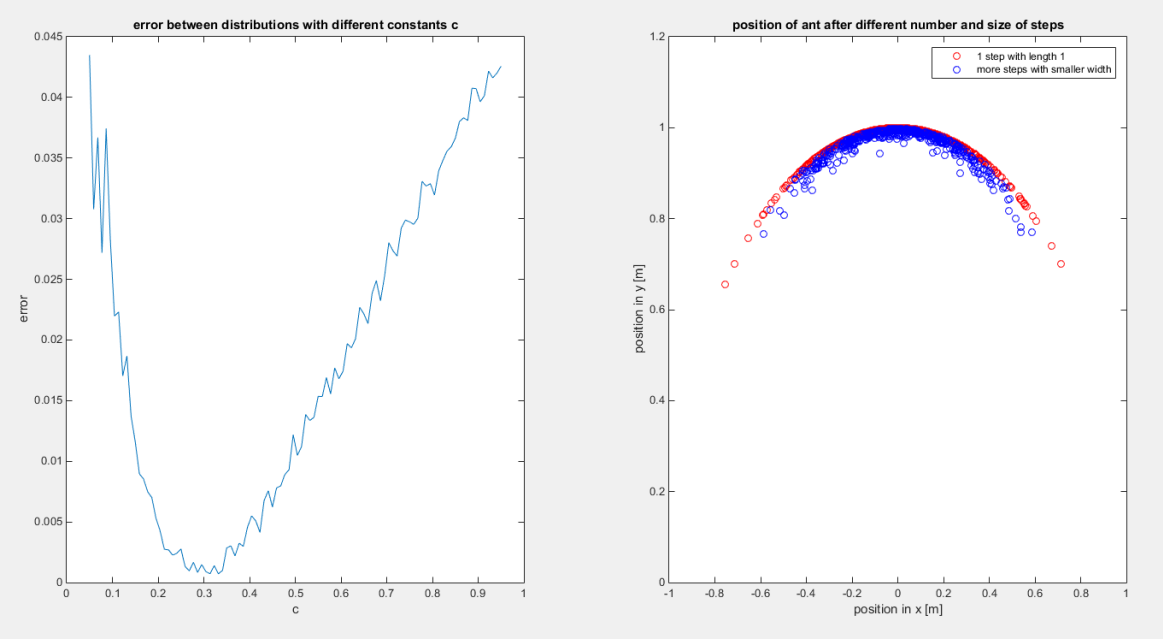

Figure 3: Variance for stepwidth

6.3 Verification of the Results

To verify, if we implemented the path integrator correctly, we let the ants do the same experiment as described in Wehner1988 [2]. The ant starts at the nest and walks 12 meters from the nest in a fixed direction. After that it turns an angle α and walks another 5 meters in this direction, where it finds food. At this point Wehner and Müller measured the angle the ants would take to find home and calculated the error in comparison to the exact position of the nest. Our results produced look exactly the same, what proves, that we implemented the path integrator in the way Wehner and Müller think it is correct.

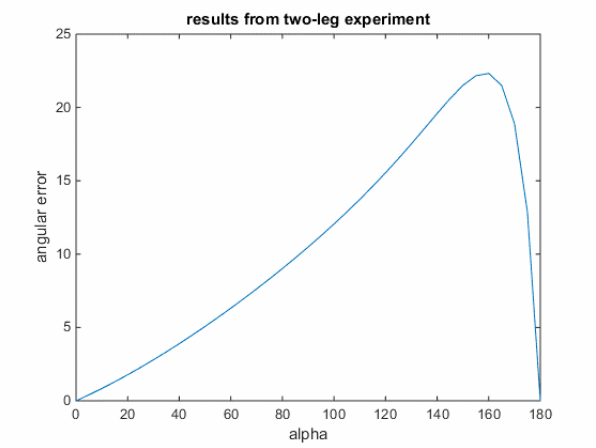

Figure 4: Angular Error produced

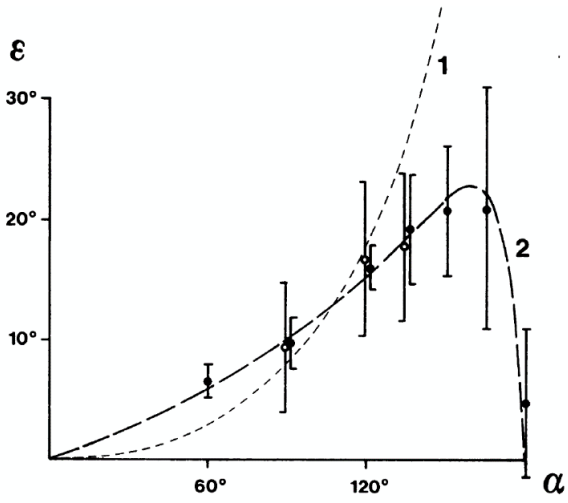

Figure 5: Angular Error according to [2]

In figure 4, we see the angular error of our Matlab simulations in figure 5 we see the plot the paper Wehner1988[2]. Our implementation of the path integrator varies in the following points to the implementation of GordonTeam2008 [5]

- δ is meant as the angle between the angle of the global vector and the actual direction instead of the actual change in direction
- the fitting constant k has to be rescaled if we use angles in radians instead of degrees.
- they miscalculated the end of the second leg in the two-leg experiment.

7 Summary and Outlook

In the following section we would like to discuss our findings and provide an outlook on what could also be looked at in future projects.

Are we able to predict what happens if we alter the environment? (depends on the question beforehand)

We were able to run the simulation in different environments (different landmarks and variable food source locations). As a result we saw that the ants navigational behaviour strongly depends on the setup of the landmarks.

Can the ants survive if we rid the environment completely of any landmarks?

When the ant remains in radius of about ten meters around the nest it has no trouble to find back to the nest without the aid of visual landmarks. If the ants foraging radius reaches further, which in reality is often the case many ants will get lost along the way which makes landmarks essential at those distances. However, we do not know whether this also applies to real ants or if this effect is just an artifact of the employed model. This possibility cannot be dismissed since we used the model for a different purpose than it was originally designed for.

In terms of the implemented model we were able to reproduce the error profile given by the experimental data in Wehner1988 [2] almost perfectly.

A Appendix

A.1 run.m

```
function run(dt,printFlag)
    if nargin == 0
        dt = 0.3;
        printFlag = false;
    end

    nestLocation = [0;0];

    global add % global variable to check if new food sources were added
    global coords % coordinates to spawn source
    global exit % handling clean exit
    add = 0;
    exit = 0;
    coords = [0;0];

    ground = Ground;
    ground.timeLapseFactor = 10000;
    ground.nestLocation = nestLocation;

    % place ants
    nAnts = 10;
    ants = Ant(zeros(nAnts,1));
    for i = 1 : length(ants)
        ants(i) = Ant;
        ants(i) = ants(i).setUp(ground);
    end
    ground.ants = ants;

    % place food sources
    nFoodSources = 5;
    foodSourceDistance = 100;
    xCoord = foodSourceDistance*(2*rand(1,nFoodSources)-1);
    yCoord = foodSourceDistance*(2*rand(1,nFoodSources)-foodSourceDistance);
    for k = 1:nFoodSources
        ground.spawnFoodSource(xCoord(k),yCoord(k));     % WRONG: Has no effect
        ground = ground.spawnFoodSource(xCoord(k),yCoord(k)); % CORRECT:
    end

    % place landmarks
    nLandmarks = 5;
    landmarkDistance = 10;
    xCoord = landmarkDistance*(2*rand(1,nLandmarks)-1);
    yCoord = landmarkDistance*(2*rand(1,nLandmarks)-1);
    stepWidth = ants(1).velocityVector(3)*dt;
    landmarks = Landmark(zeros(1,nLandmarks));
    for k = 1:nLandmarks
        landmarks(k) = landmarks(k).setUp(xCoord(k),yCoord(k),stepWidth);
    end
    ground.landmarks = landmarks;

    hFigure = figure;  %# Create a figure window
    hfAxes = axes;      %# Create an axes in that figure
    %set(hFigure,'WindowButtonMotionFcn',...  %# Set the WindowButtonMotionFcn so
```

```
    %    {@axes_coord_motion_fcn,hAxes});      %#   that the given function is called
    %                                          %#   for every mouse movement
    setGlobalhAxes(hfAxes);

    set(hFigure, 'WindowButtonDownFcn', @axes_coord_click_fcn);
    set(hFigure, 'CloseRequestFcn', @onClosing);

    currentPrint = 1;
    while(currentPrint == 1)
        % set timestamt
        tic;

        if add == 1 % add food source
           disp('new food source at:');
           disp(coords);
           add = 0;
           ground = ground.spawnFoodSource(coords(1),coords(2));
        end
        if exit == 1
           break;
        end

        for j = 1 : length(ground.ants)
            [ants(j), ground] = ants(j).performStep(ground,dt);
            ground.ants(j) = ants(j);
        end
        cla;
        hold on;
        axis([-100 100 -100 100]);
        title('foraging ants');
        xlabel('length [m]');
        ylabel('length [m]');
        ground = updateGround(ground,length(ants),printFlag);
        drawnow;

        % a pause so that according to the timeLapseFactor a step in
        % realtime takes ONE second.
        timeToWait = 1-toc;

        if timeToWait < 0
            timeToWait = 1;
        end
        pause(timeToWait*(ground.timeLapseFactor)^(-1)-0.002);
    end
end

%%********************************************************************
% UI handling
%%********************************************************************
% The following section handles all UI interactions such as closing and
% spawing new foodsources

% set hAxes variables for callback
function setGlobalhAxes(val)
    global hAxes
    hAxes = val;
end
```

```
% get hAxes variables for callback
function r = getGlobalhAxes
    global hAxes
    r = hAxes;
end

% close callback used to terminate application
function onClosing(src, callbackdata)
    disp('end');
    delete(src);
    global exit
    exit = 1;
end

% Callback when plot is clicked
function axes_coord_click_fcn(src,event)
  global coords
  hAxes = getGlobalhAxes;
  coords = get_coords(hAxes);            %# Get the axes coordinates

  global add
  add = 1;
end

function value = get_in_units(hObject,propName,unitType)

  oldUnits = get(hObject,'Units');  %# Get the current units for hObject
  set(hObject,'Units',unitType);    %# Set the units to unitType
  value = get(hObject,propName);    %# Get the propName property of hObject
  set(hObject,'Units',oldUnits);    %# Restore the previous units

end

function coords = get_coords(hAxes)

  %# Get the screen coordinates:
  coords = get_in_units(0,'PointerLocation','pixels');

  %# Get the figure position, axes position, and axes limits:
  hFigure = get(hAxes,'Parent');
  figurePos = get_in_units(hFigure,'Position','pixels');
  axesPos = get_in_units(hAxes,'Position','pixels');
  axesLimits = [get(hAxes,'XLim').' get(hAxes,'YLim').'];

  %# Compute an offset and scaling for coords:
  offset = figurePos(1:2)+axesPos(1:2);
  axesScale = diff(axesLimits)./axesPos(3:4);

  %# Apply the offsets and scaling:
  coords = (coords-offset).*axesScale+axesLimits(1,:);

end
```

Listing 2: run.m

A.2 distanceBetweenTwoPoints.m

```
function distance = distanceBetweenTwoPoints(point1,point2)

    distance = norm(point1 - point2);
end
```

Listing 3: distanceBetweenTwoPoints.m

A.3 vector2angle.m

```
function angle = vector2angle(v)

angle = atan(v(2)/v(1));
if v(1) < 0
    angle = angle+pi;

end
angle=mod(angle,2*pi);
end
```

Listing 4: vector2angle.m

A.4 updateGround.m

```
function ground = updateGround(ground,currentStep,printFlag)
    try % for some reason the ant array can have less entries that get counted
        % Update the ants pixels
        for i = 1 : length(ground.ants)
            text(ground.ants(i).prevLocation(1),ground.ants(i).prevLocation(2)+1,...
                strcat('#',int2str(i)),...
                'BackgroundColor',[.78 .89 1],...
                'FontSize',8,...
                'HorizontalAlignment','center');
            plot(ground.ants(i).prevLocation(1),ground.ants(i).prevLocation(2),'ko');
        end
    catch
        warning('Ant index exceeded.');
    end

    % Update the landmark pixels
    for i = 1 : length(ground.landmarks)
        plot(ground.landmarks(i).location(1),ground.landmarks(i).location(2),'bo');
    end

    % Update the nest pixels
    plot(ground.nestLocation(1),ground.nestLocation(2),'ro');

    % Update the food source pixels
    [~, count] = size(ground.foodSourceLocations);
    for i = 1 : count
        plot(ground.foodSourceLocations(1,i),ground.foodSourceLocations(2,i),'go');
```

```
    end

    if printFlag
        % It assures that the up to 9999 frames
        % the images are saved in the right order
        zeroStr = '000';
        if currentStep > 999
            zeroStr = '';
        elseif currentStep > 99
            zeroStr = '0';
        elseif currentStep > 9
            zeroStr = '00';
        end
        print(strcat('results/pheromoneResults/snap_',...
                zeroStr,int2str(currentStep),'.png'),...
              '-dpng');
    end
end
```

Listing 5: updateGround.m

A.5 Landmark.m

```
classdef Landmark
    properties
        location; % location of the landmark
        direction; % directionant has to go when reaching landmark
        stepsToFollow; % number of steps in a direction when reaching landmark
        status; % bool: is true if landmark is in viewRangeLandmarks of ant
    end

    methods

        function this = setUp(this,x,y,stepWidth)
            if nargin ~= 0
                this.location = [x;y];
                this.direction = -[x;y];
                this.stepsToFollow = floor( sqrt(x^2+y^2)/stepWidth );
                this.status = 1;
            else
                this.location = nan;
                this.direction = nan;
                this.stepsToFollow = nan;
                this.status = 0;
            end
        end

        % Needed to preallocate an array of landmarks
        function landmarks = Landmark(F)
            if nargin ~= 0 % Allow nargin == 0 syntax.
                m = size(F,1);
                n = size(F,2);
                landmarks(m,n) = Landmark; % Preallocate object array.
            end
        end
```

```
    end
end
```

Listing 6: Landmark.m

A.6 Ground.m

```
classdef Ground
    properties
        nestLocation
        foodSourceLocations
        ants
        landmarks
        timeLapseFactor % determines how fast the simulation is running
    end

    methods

        % adds a food source at location (x,y)
        function this = spawnFoodSource(this, x, y)
            this.foodSourceLocations = [this.foodSourceLocations [x; y]];
        end

        % Removes food source if collected by ant
        function this = collectFoodSource(this, foodSourceLocation)
           [~, count] = size(this.foodSourceLocations);

           if count == 0
              return;
           end

           index = 1;
           [~, count] = size(this.foodSourceLocations);
           for i = 1 : count
              if isequal(this.foodSourceLocations(:,i), foodSourceLocation)
                  index = i;
              end
           end
           this.foodSourceLocations(:,index) = [];
        end

        function bool = isLocationAtNest(this,loc)
            if norm(this.nestLocation-loc) == 0
                bool = true;
            else
                bool = false;
            end
        end

        function bool = isLocationAtFoodSource(this,loc)
            bool = false;
            for i = 1 : size(this.foodSourceLocation,2)
                if norm(this.foodSourceLocation(:,i)-loc) == 0
```

```
                    bool = true;
                    return;
                end
            end
        end
    end
end
```

Listing 7: Ground.m

A.7 Ant.m

```
classdef Ant
    properties
        prevLocation % previous position in absolute coordinates
        location % Position in absolute coordinates
        velocityVector % Vector [Vx Vy speed]
        carryingFood % Bool
        walkDirectlyHome % Bool
        stepsToGo % how many steps the ant has go in a specific direction
        viewRange % How far an ants can "see" food sources and the nest
        viewRangeLandmarks % How far an ant can see landmarks
        nearestLandmark % nearest landmark to position of ant
        globalVector % Vector pointing directly to the nest
        phi % Part to implement the "global vector" in a more
            % realistic way. phi represent an angle.
        l   % The second part needed for what is described above.
            % l represent the total length walked till now.
        meanDirection % mean for the turning angle
        lookingFor % String witch says what the ant is looking for.

        startangle % angle with ant leaves the nest

        isLeavingNest  % prohibit returning directly to nest on leaving
        maxDistance % when global vector reaches this value, ant returns to nest
        lostPosition % ant cannot return this is the place where global vector=0
        searchRadius % radius around lost distance in which the ant will be searching

        timer % counts the time that has ellapsed since the ant had left the nest
        timerWError % time the ant thinks has ellapsed since it left the nest
        livingTime % time after which the ant dies of overheating
        nearestFoodSourceLocation % nearest food source to ant
    end

    methods

        % Needed to preallocate an array of ants.
        function antsArr = Ant(F)
            if nargin ~= 0 % Allow nargin == 0 syntax.
                m = size(F,1);
                n = size(F,2);
                antsArr(m,n) = Ant; % Preallocate object array.
            end
        end
```

```
% ant performs a step
function [this, ground] = performStep(this,ground,dt)
    eps = 1e-4;

    % ant dies if it's out for too long
    if(this.timer >= this.livingTime)
        return;
    end

    % update the lookingFor property of ant

    % ant returns to nest if remaining time to get back gets short
    % ant plans with a return distance that is securityFactor times
    % the distance it would need to return directly
    effectReturnTime = this.l/this.velocityVector(3);
    securityFactor = 2;

    if(this.timerWError+securityFactor*effectReturnTime-this.livingTime)>0
       this.lookingFor = 'nest';
    end

    % ants goes back if it's too far away
    if (abs(this.l) > this.maxDistance)
        this.lookingFor = 'nest';
    end

    % ant picks up food and set nest as target if its location is
    % at food source
    if strcmp(this.lookingFor, 'food') && norm(this.location- ...
        ~this.nearestFoodSourceLocation) < eps
        this.carryingFood = true;
        this.lookingFor = 'nest';
        ground = ground.collectFoodSource(this.nearestFoodSourceLocation);
    end

    %  ant puts down food and set food source as target if its
    %  location is at nest
    if strcmp(this.lookingFor, 'nest') && norm(this.location - ...
        ~ground.nestLocation) < eps
        this = this.setUp(ground);
    end

    % ant looks for nearest landmark if in sight and if nest is not
    % nearer
    this.nearestLandmark = this.getNearestLandmark(ground);
    if this.nearestLandmark.status
        if strcmp(this.lookingFor,'nest') && ...
                distanceBetweenTwoPoints(this.nearestLandmark.location, ...
                ~this.location) < this.l && this.walkDirectlyHome
            this.lookingFor = 'landmark';
        end
    end

    % ant walks directly towards the nest if it is at landmark
    if strcmp(this.lookingFor, 'landmark') && norm(this.location - ...
        ~this.nearestLandmark.location) < eps
        this.lookingFor = 'nest';
```

```
            this.walkDirectlyHome = 1;
            this.stepsToGo = this.nearestLandmark.stepsToFollow;
            this.velocityVector(1:2) = this.nearestLandmark.direction;
        end

        % ant moves
        if this.walkDirectlyHome == 1
            this = this.updateLocation(dt);
            this.stepsToGo = this.stepsToGo - 1;
        elseif strcmp(this.lookingFor, 'food')
            if norm(this.nearestFoodSourceLocation-this.location) < this.viewRange
                this = this.stepStraightTo(this.nearestFoodSourceLocation,dt);
            else
                this = this.takeRandomStep(dt);
            end
        elseif strcmp(this.lookingFor, 'nest') && ~this.walkDirectlyHome
            this = this.returnHomeUsingPathIntegrator(ground,dt);
        elseif strcmp(this.lookingFor, 'landmark')
            this = this.stepStraightTo(this.nearestLandmark.location,dt);
        end

        % update rules for ant
        this.timer = this.timer + dt;
        this.timerWError = this.timerWError + dt*(1 + 0.5*randn(1,1));
        this = this.updateGlobalVector(ground);
        this = this.updateNearestFoodSource(ground);

        % stop following path when stepsToFollow == 0
        if this.stepsToGo == 0
            this.walkDirectlyHome = 0;
        end
    end

    % This method makes the ant do a step directly straight to some
    % point. If the target is in range, it stops there.
    function this = stepStraightTo(this,point,dt)
        v = point - this.location;
        if norm(v) < this.velocityVector(3)*dt
            this.prevLocation = this.location;
            this.location = point;
        else
            this.velocityVector(1:2) = v;
            this = this.updateLocation(dt);
        end
    end

    % ant performs random step
    function this = takeRandomStep(this, dt)
       c = 0.3;
       factor = dt^c; % variance gauge
       sigma0 = pi/16;
       varphi = normrnd(0,factor*sigma0);
       this.velocityVector(1:2) = [cos(varphi) -sin(varphi) ; ...
       ~sin(varphi) cos(varphi)]*this.velocityVector(1:2);
       this = this.updateLocation(dt);
    end

    % Navigate home using path integrator
```

```
function this = returnHomeUsingPathIntegrator(this, ground, dt)
    if isnan(this.l)
        if norm(ground.nestLocation-this.location) < this.viewRange
            this = this.stepStraightTo(this.nearestFoodSourceLocation,dt);
            this.lostPosition = nan;
        else
            if isnan(this.lostPosition)
                this.lostPosition = this.location;
            end
            this = this.lookAround(this.lostPosition,dt);
        end
    else
        if norm(ground.nestLocation-this.location) < this.viewRange
            this = this.stepStraightTo(ground.nestLocation,dt);
        else
            this = this.stepStraightTo(this.location-this.globalVector,dt);
        end
    end
end

% ant is searching in an area of radius 'searchRadius'
% and center 'center'
function this = lookAround(this,center,dt)
    if norm(this.location-center) >= this.searchRadius
        this = this.stepStraightTo(center,dt);
    else
        this = this.takeRandomStep(dt);
    end
    % if ant can see a landmark it goes there
    if this.nearestLandmark.status
        this.lookingFor = 'landmark';
    end
end

% This method updates the location of the ant using velocity vector
% information
function this = updateLocation(this,dt)
    v = this.velocityVector(1:2);
    theta = vector2angle(v);
    yPart = sin(theta)*this.velocityVector(3)*dt;
    xPart = cos(theta)*this.velocityVector(3)*dt;
    this.prevLocation = this.location;
    this.location = this.location + [xPart;yPart];
end

% This method updates the global vector after the ant moved.
function this = updateGlobalVector(this, ground)
    % if near nest set global vector to zero
    if norm(ground.nestLocation-this.location) < this.viewRange
        this.l = 0;
        this.phi = vector2angle(this.velocityVector);
        this.isLeavingNest = 1;
    else
        if this.isLeavingNest == 1
            this.isLeavingNest = 0;
            this.l = this.viewRange;
            this.phi = vector2angle(this.location-ground.nestLocation);
        else
```

```
                k = 0.1316; % fitting constant from paper transformed to radians
                eps = 1e-6;

                v = this.location - this.prevLocation;
                delta = vector2angle(v)-this.phi;

                if abs(delta) > pi+eps % if angle obtuse-convert it to acute angle
                    if (delta > pi)
                        delta = -(2*pi-delta);
                    else
                        delta = 2*pi+delta;
                    end
                end

        this.phi = mod(this.phi + norm(v)*k*(pi+delta)*(pi-delta)*delta/this.l,2*pi);
        this.l = this.l + norm(v) - 2*abs(delta)/pi*norm(v);
        this.globalVector = [cos(this.phi) ; sin(this.phi)]*this.l;
            end
        end
    end

    % gets all the landmarks in sight of the ant
    function inRangeLandmarks = getLandmarksInRange(this,ground)
        inRangeLandmarks = Landmark(zeros(size(ground.landmarks)));
        j = 1;
        for i = 1 : length(ground.landmarks)
            landmark = ground.landmarks(i);
            if norm(landmark.location - this.location) <= this.viewRangeLandmarks
                inRangeLandmarks(j) = landmark;
                j = j+1;
            end
        end
        inRangeLandmarks = inRangeLandmarks(1:j-1);
    end

    % gets the nearest landmark from all the landmarks in sight of the
    % ant
    function nearestLandmark = getNearestLandmark(this,ground)
        nearestLandmark = Landmark;
        inRangeLandmarks = getLandmarksInRange(this,ground);
        if ~isempty(inRangeLandmarks)
            nearestLandmark = inRangeLandmarks(1);
  minDistance = distanceBetweenTwoPoints(nearestLandmark.location,this.location);
            for i = 2 : length(inRangeLandmarks)
                landmark = inRangeLandmarks(i);
  distance = distanceBetweenTwoPoints(landmark.location,this.location);
                if distance <= minDistance
                   nearestLandmark = landmark;
                   minDistance = distance;
                end
            end
        end
    end

    % set nearest food source
function this = updateNearestFoodSource(this, ground)
    [~, length] = size(ground.foodSourceLocations);
     if length == 0
```

```
    this.nearestFoodSourceLocation = [100;100]; % set the location somewhere far away
            return;
            end
    this.nearestFoodSourceLocation = ground.foodSourceLocations(:,1);
            distance = norm(this.location-ground.foodSourceLocations(1));
            if length > 1
                for i = 2 : length
                    newDistance = norm(this.location-ground.foodSourceLocations(:,i));
                    if newDistance < distance
                        distance = newDistance;
              this.nearestFoodSourceLocation = ground.foodSourceLocations(:,i);
                    end
                end
            end
        end

        % Build an ant
        function this = setUp(this,ground)
            v = ([rand;rand]).*2-1;
            v = v./norm(v);
            v = [v;1];
            this.maxDistance = 100;
            this.isLeavingNest = 1;
            this.startangle = vector2angle(v);
            thi.phi = vector2angle(v);
            this.velocityVector = v;
            this.carryingFood = 0;
            this.walkDirectlyHome = 0;
            this.stepsToGo = nan;
            this.viewRange = 2;
            this.viewRangeLandmarks = 5;
            this.nearestLandmark = Landmark;
            this.globalVector = [0;0];
            this.lookingFor = 'food';
            this.prevLocation = nan;
            this.location = ground.nestLocation;
            this.phi = vector2angle(this.velocityVector);
            this.searchRadius = 8;
            this.l = 0;
            this.lostPosition = nan;
            this.timer = 0;
            this.timerWError = 0;
            this.livingTime = 150;
            this = this.updateNearestFoodSource(ground);
        end
    end
end
```

Listing 8: Ant.m

B Correspondence

R. Whener ,author of [1],[2], [3] kindly answered us a few questions.

Guten Tag Herr Wehner

Wir sind Elektrotechnik-Studenten der ETH und sind im Rahmen eines Software-Projektes damit beschäftigt, das Verhalten eines Ameisenvolks des Typs "Cataglyphis Fortis" nachzubilden.

Wir haben ein paar Papers von Ihnen gelesen und haben dort herausgefunden, dass sich diese Ameisen-Art mittels "Path-Integrator" und "Landmarks" orientiert. "Pheromone-Based-Orientation" spielt anscheinend keine Rolle, da die Pheromone in der Wüste zu schnell verdunsten.

Nun haben wir folgende Frage:
Wenn eine Ameise dieses Typs eine Nahrungsquelle gefunden hat, wie "informiert" es sein Volk, wo diese Nahrungsquelle ist? Schliesslich gibt es ja keine Pheromon-Spur, der die Artgenossen folgen könnten.

Wir bedanken uns schon jetzt recht herzlich für Ihre Zeit.

Beste Grüsse

Matthias Heinzmann

Von: Rüdiger Wehner [rwehner@zool.uzh.ch]
Gesendet: Mittwoch, 18. November 2015 08:49
An: Heinzmann Matthias
Betreff: AW: Cataglyphis Fortis

Sehr geehrter Herr Heinzmann,

vielen Dank für Ihr Interesse an unseren Cataglyphis-Arbeiten. In der Tat, Arbeiterinnen aller Cataglyphis-Arten sind Einzel-Fourageure, die andere Nestmitglieder nicht über mögliche Futterstellen informieren. Es gibt also keinerlei Kommunikationsmittel zu Futterstellen. Das ist i.a. auch nicht nötig, da es sich bei der Beute um einzelne Insektenleichen handelt, die – einmal weggetragen – nicht mehr vorhanden sind.

Mit besten Grüssen

Rüdiger Wehner

****
Prof. Dr. Rüdiger Wehner

Brain Research Institute
University of Zürich
Winterthurerstrasse 190
CH-8057 Zürich, Switzerland
Phone: +41 44 635-4831
Secretary: Jacqueline Oberholzer
Phone: +41 44 635-4813
E-mail: rwehner@zool.uzh.ch<mailto:rwehner@zool.uzh.ch>

C References

Literatur

[1] R. Wehner. *Karl von Frisch lectures.* Springer, July 2003.

[2] R. Wehner et al. *Neurobioligy Vol. 85.* Proc. Natl. Acad. Sci. USA, July 1988.

[3] R. Wehner et al. *Letters to Nature Vol.394.* Macmillan, July 1998.

[4] http://ch.mathworks.com/help/matlab/numeric-types.html. November 2015.

[5] V.Megaro, E.Rudel. *MSSSM - Thesis GordonTeam.* ETH, 2008.