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 Adaptive orientation of desert ants (Cataglyphis)

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

The Desert Ant (Cataglyphis) is an interesting creature. It lives in the desert where conditions are very harsh. Being to long outside of its nest the ant will die due to the ambient heat, therefore it is more than vital for the ant's survival that it has a sophisticated way to navigate through the pans of sand that make out most of its habitat. Since the early 20th century biologists have been fascinated by this ant and its navigation through the desert. Recently there has been speculation about the methods that are utilized by these ants to navigate between sources of food and their nest. If we could reproduce the path-pattern observed by real ants with a model we would be able to predict how they would react if something would change in their ecosystem. We will base our work upon an essays written by R. Wehner [1],[2], [3]. In his work he suggests 3 methods of navigation which the ants can use, namely pathintegration and visual landmark recognition. 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 article. In the 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 problems as well as to bind all methods together. Furthermore we want to implement a memory system along with a learning machine which will allow the ant to dynamically adapt to an environment.

2 Individual contributions

2.1 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 pray usally consists of small parts of insects, that can be carried by on ant, and no need for sharing information arises. [Appendix B]

2.1.1 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 mean of all its turning angles executed and is therefore always aware in which direction its nest is located. These calculations are executed with imperfections which accumulate with growing distance. 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, they take into account the rough approximation of the ants pathintegration.

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

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

where k is a normalization constant, δ is the angle with which the ant is turning its current direction and the step width is assumed to be 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 $\varphi = 0$ 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 $\mu = 0$ and $\sigma = \frac{\pi}{16}$. The plot of the expected error of the global vector in this one step can be seen in the figure below.

[1] on page 5288

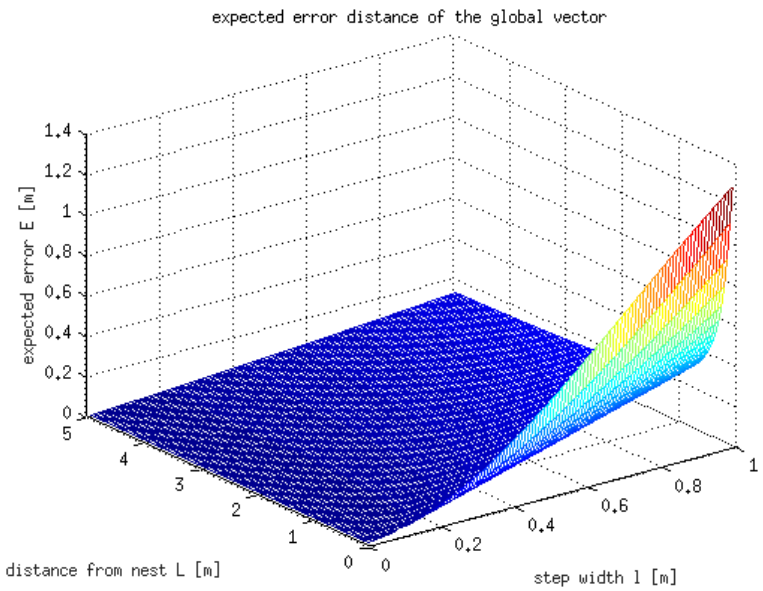

Figure 1: Pathintegrator Error

It can easily be observed that the distance from the nest L has almost no influence on the error expect for very small distances (0m – 2m). The performance of the global vector model is rather influenced by the choice of the step width l. According to the figure, the step width should be chosen sufficiently small.

By varying the step width we end up with a new problem. The path the ant walks along is strongly influenced by the choice of the step width. But what we would like to have 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 theta in relation to dt (step width l of the ant is determined by $l = dt \cdot antSpeed$). We made the following ansatz:
$\sigma(dt) = dt^c \sigma_0$ 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 $\sigma_0 = \frac{\pi}{12}$ and $dt = 0.1$ can be seen in the figure below.

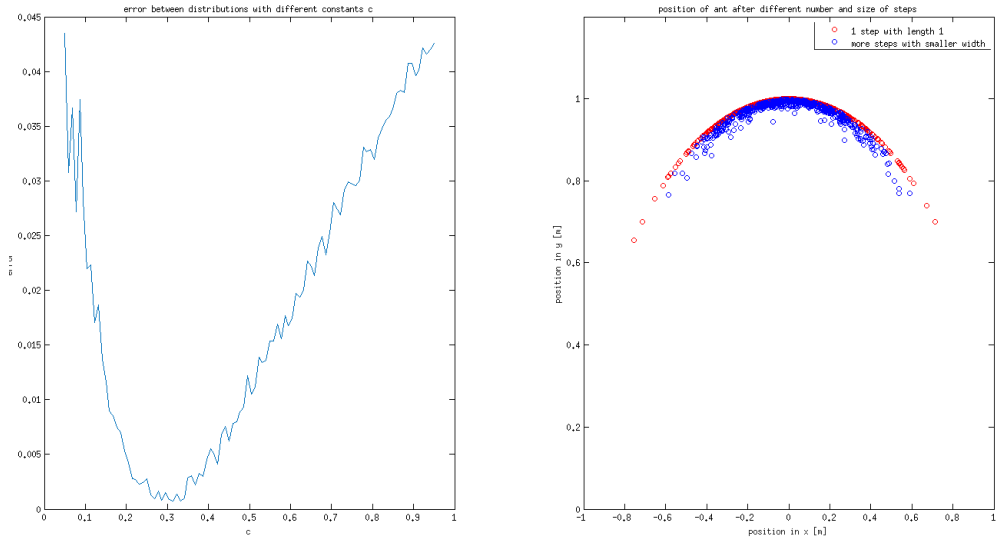

Figure 2: VarianceForStepWidth

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 read off as being approximately 0.3.

2.2 Local landmark Orientation

....

2.2.1 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

3 Introduction and Motivations

4 Description of the Model

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.

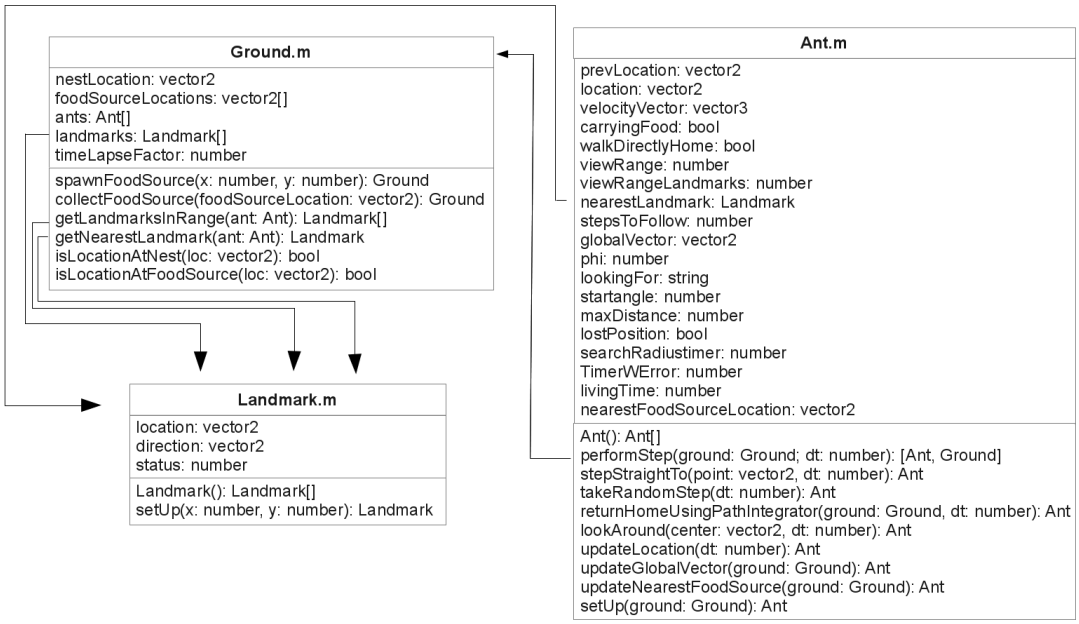

Figure 3: UML

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)

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      1
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:

[2] Mathworks2015[4]

- **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 2.2.1) . 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

7 Summary and Outlook

A Appendix

A.1 run.m

```
function run(dt,printFlag)
    if nargin == 0
        dt = 0.2;
        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,dt);
    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: Update gets written
    end

    % place landmarks
    nLandmarks = 5;
    landmarkDistance = 10;
    xCoord = landmarkDistance*(2*rand(1,nFoodSources)-1);
    yCoord = landmarkDistance*(2*rand(1,nFoodSources)-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),dt,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

```
PASTE HERE
 distance between2Points
```

Listing 3: distanceBetweenTwoPoints.m

A.3 vector2angle.m

```
PASTE vector2angle.m
```

Listing 4: vector2angle.m

A.4 updateGround.m

```
PASTE updateGround here
```

Listing 5: updateGround.m

A.5 Landmark.m

```
PASTE Landmark.m here
```

Listing 6: Landmark.m

A.6 Ground.m

```
PASTE Ground.m here
```

Listing 7: Ground.m

A.7 Ant.m

```
PASTE Ant.m here
```

Listing 8: Ant.m

B Correspondence

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

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.