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 Behaviour
Simulation of the navigation and movement of desert ants

Georg Wiedebach & Wolf Vollprecht

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.

Georg Wiedebach Wolf Vollprecht

Abstract

This paper is the final result of the course Modeling Social Systems with MATLAB which aimed to offer an insight into the MATLAB programming language and to use said language to model social systems with various different aprroaches. The timeframe of the course is one semester.
In this paper we will try to show how to replicate the behaviourr of desert ants in a MATLAB simulation. Furthermore we will discuss our results and compare them to experimental results obtained by biologists.

Contents

1 Individual contributions

2 Introduction and Motivations

3 Description of the Model

4 Implementation

5 Simulation Results and Discussion

6 Summary and Outlook

A Research Plan

B MATLAB Code

B.1 main.m

```
%% Mainfile
% for common configurations of the simulation (mostly testing
% purposes and initiating)

clc;
clear all;
clf;
close all;
addpath('Maps');


%% Variables

runduration = 5;     % Duration of simulation
render = true;
path_render = 1;



%% Options for different map-loading methods
% only one option should be enabled

% 1. Map from m-file
% ------------------------------------------------
%map1
```

```
% 2. Random map from generator
% Some values need to be set by the user:
% -------------------------------------------------------------
% mapsize = 100;
% s = simulation(mapsize, render, path_render);
% s.l.generateLandscape(mapsize, 30, 55, 0.8);
% s.l.nest = [5 5];
% s.a.position = s.l.nest;
% s.l.feeder = [95 95];


% 3. Map from image.png
% -------------------------------------------------------------
s = simulation(100, render, path_render);
s.l.load_image('map2', 'png')
s.a.position = s.l.nest;
s.l.landmarks = [s.l.landmarks; s.l.nest];



%% Run the simulation

s.a.createGlobalVector(s.l);
s.a.createLocalVectors(s.l.landmarks);

for i = 1:runduration
    s.run();
    i
end

%aviobj = close(s.aviobj);
% enable to create a movie (3/3)


%% Plotting the results on steps

figure(2)
plot(s.a.results_food_finding,'r')
hold on
plot(s.a.results_nest_finding,'g')
legend('food-searching','nest-searching')
xlabel('number of runs')
ylabel('steps needed / time needed')
```

B.2 simulation.m

```
%% Simulation Class
% Handles everything simulationwise e.g. run the simulation, define ...
    simulation wide parameters


classdef simulation < handle
    properties (SetAccess = private)
        l               % landscape
        a               % ant

        r               % true or false
        r_path          % true or false
        r_init          % true or false
        r_ant           % rendering
        r_ant_view      % rendering

%         aviobj = avifile('antmovie.avi','compression','None');
% enable to create a movie (1/3)
    end

    methods (Access = public)
        %% Initialization
        % Initalizes a simulation with landscape size N
        function S = simulation(N,r,r_path)
            S.l = landscape(N);
            S.a = ant();
            S.r = r;
            S.r_path = r_path;
            S.r_init = true;
        end

        %% Initiates the rendering
        function init_render(S)
            S.r_init = false;

            figure(1)
            imagesc(S.l.plant)
            axis off, axis equal
            colormap ([0 1 0; 1 0 0; 1 0 0])
            hold on
            plot(S.l.nest(1), S.l.nest(2),'o','Color','k')
            plot(S.l.feeder(1), S.l.feeder(2), 'x', 'Color', 'k');

            % If landmarks exits they are plotted
            if ¬isempty(S.l.landmarks)
                plot(S.l.landmarks(:,1), S.l.landmarks(:,2), 'o', ...
                    'Color', 'b');
            end

```

```
            % Initiates the Animation in "render"
            S.r_ant = plot(S.a.position(1), ...
                S.a.position(2),'.','Color','b');
            S.r_ant_view = plot(S.a.position(1) + ...
                S.a.detection_radius*cos(2*pi/20*(0:20)), ...
                S.a.position(2) + ...
                    S.a.detection_radius*sin(2*pi/20*(0:20)), 'Color', 'k');
        end

        %% Reset after complete run
        function reset(S)
            S.a.has_food = 0;
            S.a.nest = 0;
            S.a.obstacle_vector = zeros(100, 100, 2);

            % If render is true local vectors are plotted
            if S.r
                S.render_local_vectors;
            end
        end

        %% The simulation
        % if render is true the ant will be plottet on the landscape
        function run(S)
            % On the first run and if render is true rendering is initiated
            if S.r_init && S.r
                S.init_render();
            end

            % Some variables are reset bevore a new run
            S.reset();

            % Ant searces for food until a.has_food is true
            while S.a.has_food == 0
                S.a.findFood(S.l);

                % If render is true
                if S.r
                    S.render()
                end
            end

            % Ant returns to nest similar until a.nest is true
            while S.a.nest == 0
                S.a.returnToNest(S.l)

                % If render is true
                if S.r
                    S.render()
                end
```

```
            end

        end

        %% Render the simulation
        function render(S)
            figure(1)

            % Animation of ant and view-radius
            set(S.r_ant,'XData',S.a.position(1));
            set(S.r_ant,'YData',S.a.position(2));
            set(S.r_ant_view, 'XData', S.a.position(1) + ...
                S.a.detection_radius*cos(2*pi/20*(0:20)));
            set(S.r_ant_view, 'YData', S.a.position(2) + ...
                S.a.detection_radius*sin(2*pi/20*(0:20)));
            drawnow

            % If path plotting is true
            if S.r_path
                plot(S.a.position(1), S.a.position(2),'.','Color','w')
            end

%             F = getframe(1);
%             S.aviobj = addframe(S.aviobj,F);
% enable to create a movie (2/3)
        end

        %% Render local vectors
        function render_local_vectors(S)
            S.init_render();
            for i=1:length(S.l.landmarks)
                quiver(S.l.landmarks(i,1), S.l.landmarks(i,2), ...
                    S.a.local_vectors(i,1), S.a.local_vectors(i,2),'y')
            end
        end

    end % methods
end % classdef
```

B.3 landscape.m

```
%% Landscape class
% A class for handling the landscape of a simulation


classdef landscape < handle
    properties (SetAccess = public)
```

```
        size            % Sitze of quadratic Landcape

        plant           % Matrix storing free and taken points
        landmarks       % Position of landmarks
        feeder          % Position of Feeder
        nest            % Position of Nest
    end

    methods (Access = public)
        %% Initialize Landscape
        function L = landscape(N)
            L.size = N;
        end

        %% Gererate random Landcape
        function generateLandscape(L, n, num, size, prob)
            L.plant = zeros(n,n);
            L.plant(1,:) = ones(1,n);
            L.plant(n,:) = ones(1,n);
            L.plant(:,1) = ones(1,n);
            L.plant(:,n) = ones(1,n);

            % 1. Zuf llige Hindernisse Plazieren Anzahl der Hindernisse ...
                soll fest sein:
            posspeicher = zeros(num,1);

            for i = 1:num
                pos = n+1;
                % Finden eines geeigneten Ortes:
                while L.plant(pos) || L.plant(pos-1) || L.plant(pos+1) ...
                    || L.plant(pos-n) || L.plant(pos+n)
                    pos = randi([n+1,n*n-(n+1)]);
                end

                % Plazieren und speichern des Ortes  f r Schritt 2:
                posspeicher(i) = pos;
                L.plant(pos) = 1;
            end

            % 2. Vergr ssern dieser Hindernisseauf eine bestimmte ...
                Gr sse (Hindernisse
            % wachsen  ber R nder hinaus und auf der Anderen ...
                Spielfeldseite wieder
            % hinein.
            neigh = [-1 1 -n n];

            for i = 1:num
                dir = inf;
                for j = 1:size
                    % Manchmal wird eine Richtungs nderung zugelassen:
```

```
                    if rand < prob
                        dir = inf;
                    end
                    % W hlen einer zuf lligen Richtung zum Vergr ssern:
                    while posspeicher(i) + dir < 1 || posspeicher(i) + ...
                        dir > n*n
                        dir = neigh(randi(4));
                    end
                    L.plant(posspeicher(i) + dir) = 1;
                    posspeicher(i) = posspeicher(i) + dir;
                end
            end
        end

        %% Load a map (invoked from m-files)
        function load_map(L, P)
            L.plant = P;
            L.size = length(P);
        end % load_map

        %% Load a image-map
        function load_image(L, image, type)
            img = imread(image, type);
            L.size = length(img(:,:,1));
            L.plant = ¬img(:,:,1);                          % use hex #ffffff
            [y, x] = find(img(:,:,2) == 153);
            L.landmarks = [x, y];
            [y, x] = find(img(:,:,2) == 238, 1, 'first'); % use hex #1100ee
            L.nest = [x, y];
            [y, x] = find(img(:,:,3) == 238, 1, 'first'); % use hex #11ee00
            L.feeder = [x, y];
            L.plant(1,:) = ones(1,L.size);
            L.plant(L.size,:) = ones(1,L.size);
            L.plant(:,1) = ones(1,L.size);
            L.plant(:,L.size) = ones(1,L.size);
        end

    end % methods
end % classdef
```

B.4 ant.m

```
%% Ant class
% This class defines the behaviour/movement of an ant in a given landscape


classdef ant < handle
```

```
    properties (SetAccess = public)
        % Variables that may be set for testing:
        detection_radius = 20;          % View radius of the ant
        error_prob = 0.3;               % Error probability
        turn_prob = 0.3;                % Random turns

        % general variables
        position                        % Position of Ant
        global_vector                   % Global vector
        has_food                        % true or false
        nest                            % true or false

        % move-related Veriables
        move_direction                  % last move direction
        obstacle_vector                 % Matrix stores found obstacles
        rotation                        % Defines clockwise or ...
            counterclockwise turns
        move_radius                     % Moore neighbourhood (1st) of ...
            the ant
        local_vectors                   % stores local vectors
        updated_local_vectors           % boolean array
        last_local_vector               % stores the last landmark seen

        % result-storing
        step_counter                    % for counting the steps to nest ...
            or feeder
        results_food_finding            % results in steps
        results_nest_finding            % results in steps
    end

    methods (Access = private)
        %% Function to update local vectors (only when returning)
        function update_lv(A, landmarks)
            for i = 1:length(landmarks)
                if norm(landmarks(i,:) - A.position) < ...
                    A.detection_radius && ...
                        ¬A.updated_local_vectors(i) && ...
                        ¬isequal(A.last_local_vector, landmarks(end,:))

                    % "growth-factor" is calculated
                    gfac = 0.5 * exp(-norm(A.local_vectors(i,:))/10);


                    % Local vector is adjusted
                    A.local_vectors(i,:) = round(A.local_vectors(i,:) + ...
                        gfac * (- landmarks(i,:) + A.last_local_vector));

                    % Storing information about update
                    A.last_local_vector = landmarks(i,:);
                    A.updated_local_vectors(i) = true;
```

```
                end
            end
        end

        %% Function to calculate a second direction from given local vectors
        function temp = calc_lv_direction(A, landmarks)
            temp = [0 0];
            for i=1:length(landmarks)
                if norm(landmarks(i,:) - A.position) < ...
                    A.detection_radius && ...
                        ¬isequal(A.local_vectors(i,:), [0 0]) && ...
                        A.updated_local_vectors(i) == 0

                    if isequal(A.local_vectors(i,:) + landmarks(i,:) - ...
                        A.position, [0 0])
                        A.updated_local_vectors(i) = true;
                    end

                    % all local vectors in the detection radius are ...
                        summed up
                    temp = temp + A.local_vectors(i,:) + landmarks(i,:) ...
                        - A.position;

                    if isequal(temp, [0 0])
                        A.updated_local_vectors(i) = true;
                    end

                end
            end
        end
    end % private methods

    methods (Access = public)
        %% Initalization of ant
        function A = ant()
            A.rotation = -1;
            A.move_direction = [0 1];
            A.obstacle_vector = zeros(100,100,2);
            A.move_radius = [1 1; 1 0; 0 1; 1 -1; -1 1; -1 0; 0 -1; -1 -1];
            A.step_counter = 0;
            A.nest = 0;
            A.has_food = 0;
        end

        %% Create the GlobalVector from Landscape
        function createGlobalVector(A, L)
            A.global_vector =  L.nest - A.position;
        end

        %% Initiate the local vectors
```

```
        function createLocalVectors(A, landmarks)
            A.local_vectors = zeros(length(landmarks), 2);
            A.updated_local_vectors = zeros(length(landmarks), 1);
        end

        %% FindFood
        % Moves ant randomly in landscape to find the feeder
        % calculates movevector from localvectors
        function findFood(A, L)

            % if the feeder is found:
            if isequal(A.position, L.feeder)
                A.has_food = true;
                A.last_local_vector = L.feeder;

                % results are stored and the stepcounter is reset
                A.results_food_finding = [A.results_food_finding, ...
                    A.step_counter];
                A.step_counter = 0;

                % some variables are reset or adjusted
                A.update_lv(L.landmarks)
                A.move_direction = -A.move_direction;
                A.updated_local_vectors(A.updated_local_vectors ~= 0) = 0;
                return
            end

            % The Step-Counter is incremented
            A.step_counter = A.step_counter + 1;

            % All local vectors in detection radius are considered
            dir = A.calc_lv_direction(L.landmarks);

            % If there is no local_vector in sight the ant moves based on
            % its previous direction with a probability to trun 45
            % degree
            if isequal(dir, [0 0])
                dir = A.move_direction;
                if rand < A.turn_prob
                    phi = pi/4;
                    n = sign(rand-0.5);
                    err_rotation = [cos(phi), n*sin(phi); -n*sin(phi), ...
                        cos(phi)];
                    dir = round(dir * err_rotation);
                end
            end

            % If the ant can "see" the feeder all previous calcualations are
            % overwriten and the move direction points directly towards
            % the feeder.
```

```
            if norm(A.position - L.feeder) < A.detection_radius
                dir = L.feeder - A.position;
            end

            % move is invoked
            A.move(L, dir);
        end


        %% ReturnToNest
        % Ant returns to nest after it found food
        % The global vector is used
        function returnToNest(A, L)

             % if the nest is reached:
            if A.global_vector == 0
                A.nest = true;
                A.has_food = false;

                % results are stored and the stepcounter is reset
                A.results_nest_finding = [A.results_nest_finding, ...
                    A.step_counter];
                A.step_counter = 0;

                % some variables are reset or adjusted
                A.updated_local_vectors(A.updated_local_vectors ≠ 0) = 0;
                return
            end

            % The Step-Counter is incremented
            A.step_counter = A.step_counter + 1;

            % Local vectors are updated during the way home.
            A.update_lv(L.landmarks);

            % move is invoked
            A.move(L, A.global_vector);
        end

        %% move(A,L)
        % Moves ant in landmark, according to typical ant behaviour.
        % A: Ant
        % L: Landscape
        function move(A, L, move_vector)

            % All known obstacles are considered
            for i = 1:8
                move_vector(1) = move_vector(1)...
                    + A.obstacle_vector(A.position(1) + ...
                        A.move_radius(i,1), A.position(2) + ...
```

```
                    A.move_radius(i,2), 1);
            move_vector(2) = move_vector(2)...
                + A.obstacle_vector(A.position(1) + ...
                    A.move_radius(i,1), A.position(2) + ...
                    A.move_radius(i,2), 2);
        end

        % if the given move_vector is zero a random move is chosen
        if isequal(move_vector, [0 0])
            move_vector = A.move_radius(randi([1,8]));
        end

        % The direction of the ant is given a certain random-error:
        if rand < A.error_prob
            move_vector(1) = move_vector(1) + (rand-0.5) * ...
                move_vector(1);
            move_vector(2) = move_vector(2) + (rand-0.5) * ...
                move_vector(2);
        end


        % Maindirection and seconddirection are calculated from the
        % direction given by the input vecor. The seconddirection ...
            gets a
        % Probability smaller than 0.5 based on the angle between
        % maindirection and global vector.
        maindir = round(...
            move_vector/max(abs(move_vector))...
        );
        secdir = sign(...
            move_vector - maindir * min(abs(move_vector))...
        );
        secprob = min(abs(move_vector)/max(abs(move_vector)));

        % the following tests make sure no error is produced because of
        % limit cases.
        if secdir(1) == 0 && secdir(2) == 0
            secdir = maindir;
        end
        if secprob == 0
            secdir = maindir;
        end
        if secprob <= 0.5
            tempdir = maindir;
            maindir = secdir;
            secdir = tempdir;
            secprob = 1-secprob;
        end

        temp = maindir;
```

```
            if rand < secprob
                temp = secdir;
            end

            % If there is no obstacle near the ant the rotation-direction
            % can change.
            count = 0;
            for i = 1:8
                count = count + L.plant(A.position(2) + ...
                    A.move_radius(i,2), A.position(1) + A.move_radius(i,1));
            end
            if count == 0
                A.rotation = sign(rand-0.5);
            end

            phi = pi/4;
            rot = [cos(phi), A.rotation * sin(phi); -A.rotation * ...
                sin(phi), cos(phi)];

            % Obstacle-Avoiding: New maindirection until possible move ...
                is found!
            % 180deg-Turn-Avoiding: New maindirection if ant tries to ...
                turn around
            while L.plant(A.position(2) + temp(2), A.position(1) + ...
                temp(1)) ≠ 0 ...
                    || isequal(temp, -A.move_direction)

                % A obstacle_vector is created and helps the ant to ...
                    avoid the wall
                % and endless iterations.
                if abs(A.obstacle_vector(A.position(1) + temp(1), ...
                    A.position(2) + temp(2), 1)) < 40
                    A.obstacle_vector(A.position(1) + temp(1), ...
                        A.position(2) + temp(2), 1) = ...
                        A.obstacle_vector(A.position(1) + temp(1), ...
                            A.position(2) + temp(2), 1) ...
                        + 8*temp(1);
                end
                if abs(A.obstacle_vector(A.position(1) + temp(1), ...
                    A.position(2) + temp(2), 2)) < 40
                    A.obstacle_vector(A.position(1) + temp(1), ...
                        A.position(2) + temp(2), 2) = ...
                        A.obstacle_vector(A.position(1) + temp(1), ...
                            A.position(2) + temp(2), 2) ...
                        + 8*temp(2);
                end

                % The ant "turns" around 45deg.
                % rot is rotation matrix defined above
                temp = round(temp * rot);
```

```
            end

            % move direction is stored, position and global vector are
            % adjusted.
            A.move_direction = temp;
            A.position = A.position + temp;
            A.global_vector = A.global_vector - temp;
        end % move

    end % public methods
end
```

C References