



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 Behavior

Vittorio Megaro & Emanuele Rudel

Zurich
May 2008

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.

Vittorio Megaro

Emanuele Rudel

Please wait...

If this message is not eventually replaced by the proper contents of the document, your PDF viewer may not be able to display this type of document.

You can upgrade to the latest version of Adobe Reader for Windows®, Mac, or Linux® by visiting http://www.adobe.com/go/reader_download.

For more assistance with Adobe Reader visit <http://www.adobe.com/go/acrreader>.

Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries. Mac is a trademark of Apple Inc., registered in the United States and other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Abstract

In this article we present three different simulations of strategies adopted by ants to find a food source and to consequently return to their nest. The former strategy is based on pheromone trails and is the most common among ants, while the second and the third strategy — called path and landmark integration, respectively — are typically adopted by desert ants for a number of reasons to become throughout this paper. As a conclusion we also show a comparison between our simulations and the results obtained performing tests on actual ants.

Contents

1	Individual contributions	7
2	Introduction and Motivations	7
3	Description of the Model	8
3.1	Pheromone-based orientation	8
3.1.1	Introduction	8
3.1.2	Pheromone modeling	8
3.1.3	Strategy	9
3.2	Path integration	9
3.3	Landmark orientation	10
4	Implementation	12
4.1	General implementation	12
4.1.1	Ants' random walk	13
4.2	Pheromone-based orientation	14
4.3	Path integration	14
4.4	Landmark orientation	14
5	Simulation Results and Discussion	16
5.1	Pheromone-based orientation	16
5.2	Path integration	17
5.3	Landmark orientation	17
6	Summary and Outlook	17
7	References	18
A	Matlab code	19
A.1	Ant.m	19
A.2	Ground.m	26
A.3	intensity2color.m	28
A.4	Hashtable.m	28
A.5	pheromoneAnimation.m	29
A.6	PheromoneParticle.m	30
A.7	projectPointOnLine.m	31
A.8	randomWalkTest.m	32
A.9	twoArmGlobalVectorAnimation.m	32
A.10	twoArmGlobalVectoTest.m	34

A.11	updateGround.m	35
A.12	vector2angle.m	36

1 Individual contributions

All the simulations and scripts for comparing the results have been implemented in pair. This choice allowed us to shape the design of the system together and to catch each other's mistakes very quickly. Moreover, the benefits of pair programming are that we both acquired knowledge of the complete program and one peer could help the other when stuck on a specific problem.

We split the task of writing the report so that one person would write a section and have the other to correct it afterwards — and vice versa.

For the hash table class we implemented a simplified version of the one freely available at the MATLAB central website¹

2 Introduction and Motivations

Where are we now? It may seem a ridiculously easy question to ask ourselves, especially nowadays with the advent of GPS devices that can tell us our exact position no matter where we are. If we think about it, however, men have always used several reference systems in order to determine their location such as maps, streets, numbers and even stars and constellations. These information are conventionally defined and allow us, aside from being aware of where we are, to be share our position with others whatever the reason might be. Ok, so we know where we are.

How do ants know where they are? They sure don't have the technology that humans have developed and they don't have maps or streets or numbers either. Ants are actually pretty simple animals in terms of intelligence: a colony of 40'000 ants has approximately the same number of brain cells as a single human brain. It is thus ruled out the possibility that ants have a complex orientation system.

Despite being fairly simple insects, ants are one of the most fascinating creatures alive; they appeared on Earth about 130 million years ago and they're present pretty much anywhere in the world. The ant's population is estimated to be one quadrillion (a one followed by fifteen zeros) and they account for roughly 15%-25% of the total animal biomass. An ant can lift 20 times its own body weight and can cover a distance from its nest up to 200 meters. This last fact is really interesting in light of our previous question if we consider that it would correspond, keeping the same size-distance proportion, to a walk of about 35 kilometers for a human being. It is therefore of extreme interest to understand how ants can cover such a long distance without getting lost, and that is what we will try to do in the next section.

¹MATLAB central, Simple hashtable: <http://www.mathworks.com/matlabcentral/fileexchange/6514>

3 Description of the Model

3.1 Pheromone-based orientation

3.1.1 Introduction

When ants set out to leave their nest and start searching for a food source, they constantly emit a chemical called pheromone during their walk. The trail of chemical is useful as it allows ants to find their way back to the nest and it also tells the other ants — since they can smell pheromones — where a peer has been traveling. Whenever an ant finds a food source, it releases a larger amount of pheromones as it runs back to the nest as fast as it can. It is important to point out that pheromone is a chemical that evaporates quickly and it is thus essential that an ant marks a reliable path (i.e. one path that takes to a food source) with a stronger signal. As soon as other members of the colony notice the strong pheromone trail, they start following the path and release the same pheromone to keep the path marked. The advantage of the pheromone evaporating quickly is that when a food source is exhausted ants stop leaving pheromones on that exact path — which becomes thus abandoned — and start looking for the next provisions supplier without having to worry of ending up on an old trail.



Figure 1: Ants follow an artificial pheromone trail (<http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/P/Pheromones.html>).

3.1.2 Pheromone modeling

As stated above, pheromone is a chemical that evaporates fairly quickly. More precisely, its intensity decays exponentially in time as described in the formula

$$I(t) = I(t - \Delta t) * \exp((\log(\frac{1}{2})/t_c)\Delta t)$$

(see reference [1]) where t_c represents the characteristic evaporation time of the pheromone in a particular environment. Using the available experiments (see reference [2]) we have determined the evaporation time to be around 2 minutes in a very hot environment such as the Sahara desert. We chose this evaporation time because we want to find an answer to the following question:

*Which method, out of the three described in this article, is more suitable for desert ants (*Cataglyphis fortis*)?*

The pheromone decay is one variable to take into account for this first simulation, although the most important issue is the strategy adopted by ants to find a food source.

3.1.3 Strategy

As they leave the nest, ants do not have an effective strategy for finding food: they just start walking randomly with the hope of running into a food supply. However, they always make sure not to walk too far away from the nest: if the search is unsuccessful for a given distance, they go back to the nest and start walking in a different direction. Since the walk's length cannot be computed relying just on pheromone trails it is deducible that ants must have a complementary mean to keep track of the distance from the nest. This method is called path integration and is the subject of the next subsection.

3.2 Path integration

Similar to techniques used by men in the past centuries to find out their current position, ants can estimate their location advancing of a small step from a previously computed estimation based upon known measures such as the number of steps walked in a fixed amount of time. As observed in *Path integration in desert ants, Cataglyphis fortis* by Müller and Wehner, path integration has a great advantage over pheromone trails:

“Path integration means that the animal is able to continuously compute its present location from its past trajectory and, as a consequence, to return to the starting point by choosing the direct route rather than retracing its outbound trajectory.”

At a time step n , an ant computes

1. the distance l_n that separates the ant from the nest to its current location

2. the direction ϕ_n in which the ant has covered the distance l_n

using the following formulas

$$\phi_{n+1} = \phi_n + k * \frac{(180^\circ - \delta)(180^\circ + \delta)\delta}{l_n}$$

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

These formulas have been derived from the results of several tests on desert ants, which have showed that ants are not able to solve the problem of path integration correctly most likely because of their limited brain functions. ϕ_n and l_n are always pointing to the nest and together they form what is called the **global vector**.

The drawback of path integration is that being an estimated computation it inherently has an approximation error that grows with time. In extreme cases the ant might even walk home in a completely different direction from the actual one because of sharp turns causing large approximation errors of the angle ϕ_n . If that's the case, the ant usually walks the whole distance l_n in the direction that it believes to be correct and then, not finding the nest, will start searching for it in the neighborhood with a random walk.

3.3 Landmark orientation

Landmark orientation is a technique used in conjunction with path integration: desert ants take a snapshot of the surrounding environment and match it with a well known place or object like the nest, the food source or an obstacle encountered on a walk. Since it is not explained, or probably not even known, how exactly ants match the landmarks, and we will make some assumptions of how they solve this problem in the implementation section. In that paper (see reference [3]), the only landmarks used are four black cylinders surrounding the ants' nest. Two interesting (and mutually exclusive) hypothesis have thus been formulated and verified:

1. the global vector is reset as soon as the ant recognizes the landmark array
2. the global vector is not reset when the ant sees the landmark array

A way to find out which hypothesis is correct is to conduct the following four-phases test:

Phase 1 ants starts foraging in an environment where the four cylinders are placed around the nest, as depicted in the figure below.

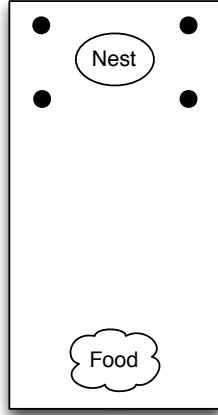


Figure 2: The initial setup in phase 1.

Phase 2 After about 30 foraging runs and when the ants arrive at the food source, they are displaced in a different area with no nest and landmarks. The ants start following their global vector recorded using path integration, but when the global vector is zero and yet there's no trace of the nest, the ants start looking around in a circular fashion.

Phase 3 The four landmarks are now restored in a fictitious position — i.e. where once again, there's no nest — and when the ants see (or perceive) the landmarks they start walking towards them. It is not clear how exactly ants handle the landmark recognition, thus we will make some assumptions when implementing this behavior in the next section.

Phase 4 The landmarks are successively removed. At this point, we can find out whether ants reset or not their global vector after having recognized the landmarks. If they did, they will keep looking for the nest right where they are, while if they didn't, they will follow the global vector until the original nest's position, and only at this point they'll start looking for the nest. The picture below shows the two possible scenarios.

The test carried out by Knaden and Wehner showed that ants actually never reset their global vector, instead they keep it as a backup in case the landmark recognition fails. In our simulation we will make use of this fact and compare our results with the ones obtained testing actual desert ants.

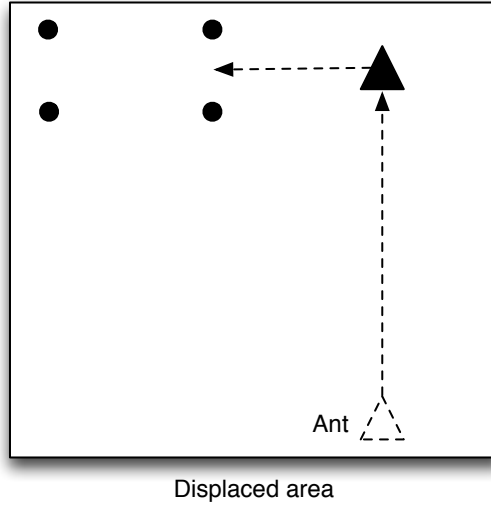


Figure 3: The ant does not find the nest when its global vector is zero. As it notices the new landmarks, it starts walking towards them.

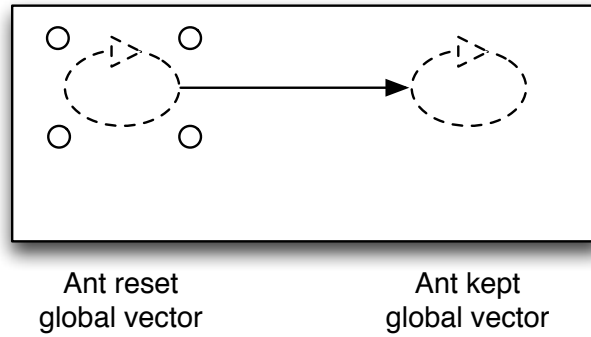


Figure 4: The ant goes back to where it thinks the nest is, according to its global vector.

4 Implementation

4.1 General implementation

The three simulations all take place on the same discrete two dimensional space defined as **ground**, where ants can freely move within the boundaries. The ground keeps track of the nest and food source locations, as well as the landmarks, the ants moving on it and the pheromone particles they have released on the ground.

4.1.1 Ants' random walk

As we previously mentioned that ants walk randomly to find a food source, we also observed that they never venture too far away from their nest. Additionally, ants rarely perform sharp turns and because of these two reasons implementing a simple random walk would have not produced a realistic simulation of how actual ants solve the task of finding a food source. We thus opted for a random walk with the following constraints:

- at each time step, the ant will change its direction of an angle chosen from a normal random distribution. This constraint makes sure that the ant will not abruptly change direction.
- the mean of the normal distribution for the angle is based on a weighted distance from the nest location, which we assume the ant can estimate thanks to path integration. Put it simple, the farther the ant walks, the greater is the probability that it will eventually turn around and start walking in another direction.

This pseudocode represents the random walk implemented for the Ant class:

```
— Ant.random_walk_step(dt)
random_walk_step(ground,dt)
do
  — velocity is the velocity vector of the current Ant
  — ground is the ground the Ant is currently in
  distance_vector := ground.nestLocation-location
  — See the implementation section for an
  — explanation of the constant variable
  distance_norm = norm(distance_vector)/constant
  weighted_vector := velocity/(distance_norm) + distance_vector*distance_norm
  weighted_angle = vector2angle(weighted_vector)
  new_angle = normrnd(weighted_angle, 0.5)
  y_velocity = sin(angle)
  x_velocity = cos(angle)
  velocity := [x_velocity; y_velocity]
  — Knowing the new velocity direction and
  — the time step dt, update the Ant's location
  updateLocation(dt);
end
```

Furthermore, at each time step the ant keeps updating its global vector, releasing pheromone particles — whose intensity changes depending on whether it is walking randomly or following a well known trail — and looking for possible landmarks, although it will only make use of them in the third simulation.

4.2 Pheromone-based orientation

One of the fundamental questions that we asked ourselves at the beginning of this research was

How do ants communicate to each other useful information such as the directions to a food source?

With this first simulation we would like to observe that ants communicate using pheromones of varying intensity to provide useful information to their peers.

We thus let an arbitrary number of ants start looking for a food source. The ants walk randomly trying to find a food source (also known as foraging) and release pheromone particles of a mild intensity in order to be able to find their way back to the nest — even though they will actually make use of path integration for the latter task.

The simulations are saved as video files which will be compared in the next section with still images of different papers that performed experiments on real ants.

4.3 Path integration

The purpose of this simulation is to assess the error accumulated over time using the formulas for path integration described in the previous section. The simulation is based on the same experiments conducted by Müller and Wehner (see reference [4]):

1. An ant walks into a pre-established path that consists of two arms of different lengths.
2. Past the second arm, the ant will head back to the nest through a direct route. The angle between the two arms changes between each run to observe how the path integration error is influenced by the sharpness of one turn.

4.4 Landmark orientation

The third simulation shows ants behavior when an array of landmarks is placed around the nest. We will skip the learning process since the goal is to observe the fact that ants do not reset their global vector in presence of landmarks. The simulation consists of the following steps:

1. the ant is initially in the displaced area (phase 2) and starts walking following its global vector. When it realizes that the nest is not where it should be anymore, the ant starts walking randomly searching for the nest. This phase lasts two minutes.

2. For one minute the four landmarks are placed in a fictitious position. If the ant is close enough to somehow perceive the landmarks, it will start looking for the nest inside the area defined by those four landmarks.
3. The landmarks are removed and the ant is free to move in the area for another five minutes. We will observe that it will first return to where the ant thinks the nest should be, and only after making sure that the nest is most certainly in that position, it will (randomly) start searching for the nest once again.

Regarding how ants recognize landmarks, we made the following assumptions:

- if ants can only see one landmark, they start walking towards to it;
- if ants can see two landmarks, they approximately walk between the two, and
- if ants recognize three or more landmarks, they will estimate the middle point between them and walk in that direction.

One problem we had to deal with in the implementation of this simulation is that our artificial ant must be able to decide in which direction it should go after two landmarks have been detected. Of course a real ant doesn't need to tackle such a problem because it knows the direction it came from and thus the direction it has to follow is trivial. Artificial ants, on the other hand, do not keep track of their previous locations and the figure below depicts the problem that could arise:

The solution is to project the coordinates (y-coordinates in this case) on the dashed line between the two landmarks and depending on whether the projected coordinates are smaller or bigger than the original values, we set the ant's direction to point downwards or upwards, respectively.

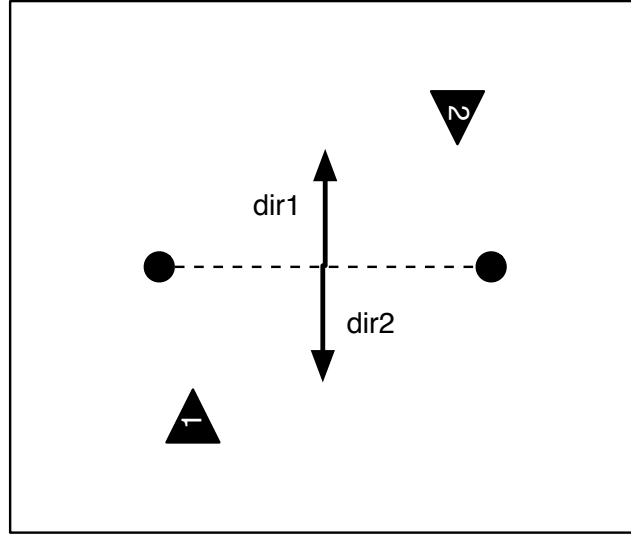


Figure 5: The artificial ant needs to compute the direction in which it should walk. For both artificial and real ants, however, the point of view from 1 and 2 is indistinguishable.

5 Simulation Results and Discussion

5.1 Pheromone-based orientation

The obtained videos of different runs of the simulation clearly showed that there's a strong cooperation between ants belonging to the same nest. As soon as one of them discovers a reliable food source, it increases the pheromone's intensity to alert its peers that the path on which it is moving leads to a food source. This fact is in accordance with the papers that describe the pheromone-based orientation (see references), but unfortunately they did not provide pictures or videos of an actual ants' cooperation. The number of foraging ants was limited to a maximum of 7 for computational limit reasons; in reality a much bigger number of ants is employed to find a food source.

We also tested the pheromone's decay for different environments and materials (based on [1]) and here we present our results:

It is important to point out that in hostile environments such as the Sahara desert, the pheromone evaporates so quickly that ants would only have a very limited amount of time (a couple of minutes) to venture out of the nest and make return before they lost track of their own pheromone and get lost. In the next section we will explain why this does not happen in reality.

5.2 Path integration

The simulation ran for the path integration model was the same as the test conducted in [4]. Our test yielded a graph which has the same trend as the reference paper, but whose values are in strong disagreement. We notice in the comparison below that the error angle (in degrees) measured in our simulation is at least twice as big as the original one: We could not find out the reason of this strange behavior; even changing the fitting constant in the formula for computing ϕ_n did not help the simulation to improve. In the other two simulations we therefore haven't used this particular implementation of path integration.

5.3 Landmark orientation

The test described in the third section has been simulated once using the same locations as in the original paper (see [3]) and once using a different location for the fictitious nest and rotating the landmarks. This procedure ensured that our implementation worked in different situations. Both the results of the first and second simulation match the ones in the original paper (except for the coordinates of the second test, of course) as shown below: We found really interesting the fact that a random walk (slightly constrained as explained above) closely represents how ants actually perform a search for their nest. The relatively simple structure of the ant's brain could justify the lack of a more elaborated strategy. On a different paper² is observed that desert ants perform a search for the nest in a spiral course around the point where the ant believed the nest was at.

6 Summary and Outlook

In this section we would like to answer the questions we had asked ourselves at the beginning of this research and those who later arose when developing the models and implementations.

Which method, out of the three described in this article, is more suitable for desert ants (Cataglyphis fortis)?

It should be clear by now that it is impossible for desert ants to only use a pheromone-based orientation: if they couldn't find any food within a few minutes they would get lost looking for their way back to the nest. Nevertheless, the desert ants' nest emits

² *The hidden spiral: systematic search and path integration in desert ants, Cataglyphis fortis*, M. Müller and R. Wehner, <http://www.springerlink.com/content/121217t6pw686314/>

a strong signal of pheromones and makes it easy for ants in the close neighborhood to find the nest entrance.

Since the desert lacks of landmarks and is continuously changing because of storms, desert ants cannot use landmark orientation either as their primary navigation system.

Desert ants thus mainly rely on path integration to navigate and they also make use of skylight information (not handled in this paper).

Are the three methods described independent of each other?

The pheromone-based orientation is independent of the other two techniques, yet it is never used alone. Ants take a safe approach and always use a combination of the different orientation systems.

Do ants communicate their position and other information to help each other?

Ants do not communicate directly with each other. They indirectly transmit information regarding the position of a food source using pheromones, but they do not share any other information.

How can ants remember exactly a path that is thousands of times their own body length?

They don't. An old conjecture assumed that ants could retrace their way back to the nest by reversing the order and direction in which they previously took their steps. This mechanism is simply not applicable to ants as it would require the insect to store an enormous amount of information. Since ants have a very small brain, this conjecture can be easily rejected.

7 References

- [1] *Alice in Pheromone Land: An experimental Setup for the Study of Ant-like Robots*, S. Garnier, F. Tâche, M. Combe, A. Grimal and G. Theraulaz, 2007
- [2] *Pheromone trail decay rates on different substrates in the Pharaoh's ant, *Monomorium pharaonis**, R. Jeanson, F. Ratnieks and J. Deneubourg, 2003
- [3] *Nest mark orientation in desert ants *Cataglyphis*: what does it do to the path integrator?*, M. Knaden and R. Wehner, 2005
- [4] *Path integration in desert ants, *Cataglyphis fortis**, M. Müller and R. Wehner, 1998

A Matlab code

A.1 Ant.m

```
classdef Ant
    properties
        prevLocation % Needed to link pheromone particles
        location % Position in absolute coordinates.
        velocityVector % Vector composed from Vr Vc Vk.
        carryingFood % Bool
        followingPheromonePath % Bool
        goingToNestDirectly % Bool
        landmarkRecognized % Bool
        viewRange % How far an ants can "see"
        pheromoneIntensityToFollow % From which intensity value the ant
            % starts to follow a pheromone path
        limitSearchDistance % After what distance from a landmark pattern
            % stop to search.
        pheromoneIntensity % How intense is the pheromone particle released
        problemEncountered % String containing the type of problem
            % encountered. Empty string means
            % no problem encountered.
        pointNearbyToSearch % Encountered a problem, try to solve it near
            % the theoretical right position.
        timeToSpendInSearch % Encountered a problem, how much waste time
            % focus the search near the theoretical right
            % position.
        confidenceRegion % Encountered a problem, how much go far to search.
        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.
        pathDirection % Third part. This is the direction in which
            % the ants was walking, in absolute coordinates.
        storedLandmarksMap % A map from a landmark pattern to a
            % local angle to follow.
        lookingFor % String witch says what the ant is looking for.
    end

    %— NOTE: the non static methods requires always an argument.
    %— This is because matlab passes secretly the instance on which
    %— the method is called as an argument.
    %— Thus the method looks like this: function my_method(this)
    %— and the call to the method is: obj.my_method()
    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
    end
end
```

```

% Main behaviour, simply call this at each step.
% It does all the things an ant should do automatically.
function [this ground] = performCompleteStep(this,ground,dt)
    % normalize the velocity vector
    v = this.velocityVector(1:2);
    v = v./norm(v);
    this.velocityVector(1:2) = v;
    % end
    this.pathDirection = this.location-this.prevLocation;
    ground = this.releasePheromone(ground);
    if ~isempty(this.problemEncountered)
        this = this.problemHandler(ground,dt);
    elseif ~isempty(this.lookingFor)
        if norm(this.location-ground.nestLocation) == 0 && ...
            strcmp(this.lookingFor,'nest')
            this.lookingFor = '';
            return;
        elseif norm(this.location-ground.foodSourceLocation) == 0 && ...
            strcmp(this.lookingFor,'food')
            this.lookingFor = '';
            return;
        end
        this = this.lookForSomething(ground,dt);
    else
        if ground.isLocationAtNest(this.location)
            this.carryingFood = 0;
            this.followingPheromonePath = 1;
            this = this.stepBack;
        elseif ground.isLocationAtFoodSource(this.location)
            this.carryingFood = 1;
            this.goingToNestDirectly = true;
            this.lookingFor = 'nest';
            this = this.backToNestDirectly(ground,dt);
        elseif this.followingPheromonePath
            this = this.followPheromonePath(ground);
        end
    end
    this = this.updateGlobalVector(dt);
end

% This method update 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 performs a single step in the random walk of an ant.
function this = randomWalkStep(this,ground,dt)
    v = this.velocityVector(1:2);
    nd = ground.nestLocation-this.location;
    d = norm(nd)/50;
    weightedVector = v./(d+0.00001)+nd.*d;

```

```

        weightedVector = weightedVector./norm(weightedVector);
        weightedAngle = vector2angle(weightedVector);
        angle = normrnd(weightedAngle,0.1*dt); % choose an angle, with normal
            distr.
        yPart = sin(angle);
        xPart = cos(angle);
        this.velocityVector(1:2) = [xPart;yPart];
        this = this.updateLocation(dt);
    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

function this = followPheromonePath(this,ground,dt)
    [bool particle] = ground.hasPheromoneInLocation(this.location);
    if bool
        if this.carryingFood
            this.prevLocation = this.location;
            this.location = particle.next.location;
        else
            this.prevLocation = this.location;
            this.location = particle.prev.location;
        end
    else
        this = this.randomWalkStep(ground,dt);
    end
end

% This method makes the ant go back to its previous position
function this = stepBack(this)
    aux = this.location;
    this.location = this.prevLocation;
    this.prevLocation = aux;
end

% This method release pheromone on the ground, in the current and
% position.
function ground = releasePheromone(this,ground)
    pheromoneParticle = PheromoneParticle;
    pheromoneParticle.location = this.location;
    if this.carryingFood || this.followingPheromonePath
        pheromoneParticle.intensity = this.pheromoneIntensity+100;
    else
        pheromoneParticle.intensity = this.pheromoneIntensity;
    end
    arr = ground.pheromoneParticles; % arr just to abbreviate next line
    [bool prevParticle positionInArray] = ground.hasPheromoneInLocation(
        pheromoneParticle.location);

```

```

        if bool
            newPheromoneParticle = ...
            arr(positionInArray).mergeWhithParticle(pheromoneParticle);
            clear pheromoneParticle;
            arr(positionInArray) = newPheromoneParticle;
            ground.pheromoneParticles = arr;
        else
            [bool prevParticle positionInArray] = ground.hasPheromoneInLocation(
                this.prevLocation);
            prevParticle = prevParticle.setNext(pheromoneParticle);
            pheromoneParticle = pheromoneParticle.setPrev(prevParticle);
            arr(positionInArray) = prevParticle;
            ground.pheromoneParticles = [arr;pheromoneParticle];
        end
    end

% This method updates the global vector after the ant moved.
function this = updateGlobalVector(this,dt)
    v = this.location-this.prevLocation;
    currentL = norm(v);
    % Implementation using the global vector variable
    %this.globalVector = this.globalVector-v;

    % Implementation using a more realist model
    oldDir = this.pathDirection;
    % Needed by the first step
    if isnan(vector2angle(oldDir))
        delta = 0;
    else
        delta = vector2angle(v)-vector2angle(oldDir);
    end
    %this.phi = (this.l*this.phi+delta+this.phi*currentL)/(this.l+currentL);
    if this.l ~= 0
        this.phi = this.phi+4e-2*(pi+delta)*(pi-delta)*delta/this.l;
    end
    this.l = this.l + currentL - delta/pi*2*currentL;
    if ~this.goingToNestDirectly
        this.globalVector = -[cos(this.phi) -sin(this.phi);
            sin(this.phi) cos(this.phi)]*v;
    end
end

% This method tries to recognize a landmark, checking in the
% lanmark array.
function this = recognizeLandmark(this,landmarks,ground,dt)
    this.landmarkRecognized = true;
    if size(landmarks,2) < 2
        this = this.stepStraightTo(landmarks(:,1)+[0.2;0.2],dt);
    elseif size(landmarks,2) < 4
        if size(landmarks,2) == 3
            candidate = 1;
            if norm(this.location-landmarks(:,candidate)) < norm(this.
                location-landmarks(:,2))
                candidate = 2;
            end
            if norm(this.location-landmarks(:,candidate)) < norm(this.
                location-landmarks(:,3))
                candidate = 3;
            end
        end
    end
end

```

```

        end
        landmarks(:, candidate) = [];
    end
    midPoint = sum(landmarks, 2) ./ 2;
    if norm(this.location - midPoint) < 0.3
        [landmarkAngle length confidence] = this.storedLandmarksMap.get(
            '2');
        absoluteAngle = mod(vector2angle(landmarks(:, 1) - landmarks(:, 2)),
            pi) + pi/2 + landmarkAngle;
        vec = [cos(absoluteAngle); sin(absoluteAngle)] .* length;
        % Choose the direction to take passing through a landmark
        pattern
        p = projectPointOnLine(this.location, landmarks(:, 1), landmarks
            (:, 2));
        if p(2) < this.location(2)
            if norm(this.location - (midPoint - vec)) <= this.viewRange &&
                ...
                norm((midPoint) - vec - ground.nestLocation) ~= 0
                this.problemEncountered = 'nestNotFound';
                this.pointNearbyToSearch = (midPoint - vec);
                this.timeToSpendInSearch = confidence;
                this.confidenceRegion = sqrt(length);
                this = this.problemHandler(ground, dt);
                return;
            end
            this = this.stepStraightTo(midPoint - vec, dt);
        else
            if norm(this.location - (midPoint + vec)) <= this.viewRange &&
                ...
                norm((midPoint) - vec - ground.nestLocation) ~= 0
                this.problemEncountered = 'nestNotFound';
                this.pointNearbyToSearch = (midPoint + vec);
                this.timeToSpendInSearch = confidence;
                this.confidenceRegion = sqrt(length);
                this = this.problemHandler(ground, dt);
                return;
            end
            this = this.stepStraightTo(midPoint + vec, dt);
        end
    else
        this = this.stepStraightTo(midPoint, dt);
    end
elseif size(landmarks, 2) == 4
    midPoint = sum(landmarks(:, 1:2), 2) ./ 2;
    midPoint = midPoint + sum(landmarks(:, 3:4), 2) ./ 2;
    midPoint = midPoint ./ 2;
    if norm(this.location - midPoint) < 0.5
        [landmarkAngle length confidence] = this.storedLandmarksMap.get(
            '4');
        absoluteAngle = mod(vector2angle(landmarks(:, 1) - landmarks(:, 2)),
            pi) + pi/2 + landmarkAngle;
        vec = [cos(absoluteAngle); sin(absoluteAngle)] .* length;
        % Choose the direction to take passing through a landmark
        pattern
        p = projectPointOnLine(this.location, landmarks(:, 1), landmarks
            (:, 2));
        if p(2) < this.location(2)

```

```

        if norm(this.location-(midPoint-vec)) <= this.viewRange &&
            ...
            norm((midPoint)-vec-ground.nestLocation) ~=0
            this.problemEncountered = 'nestNotFound';
            this.pointNearbyToSearch = (midPoint-vec);
            this.timeToSpendInSearch = confidence;
            this.confidenceRegion = sqrt(length);
            this = this.problemHandler(ground,dt);
            return;
        end
        this = this.stepStraightTo(midPoint-vec,dt);
    else
        if norm(this.location-(midPoint-vec)) <= this.viewRange &&
            ...
            norm((midPoint)-vec-ground.nestLocation) ~=0
            this.problemEncountered = 'nestNotFound';
            this.pointNearbyToSearch = (midPoint+vec);
            this.timeToSpendInSearch = confidence;
            this.confidenceRegion = sqrt(length);
            this = this.problemHandler(ground,dt);
            return;
        end
        this = this.stepStraightTo(midPoint+vec,dt);
    end
    else
        this = this.stepStraightTo(midPoint,dt);
    end
end
end

% This method makes the ant go back to the nest directly using the
% global vector.
function this = backToNestDirectly(this,ground,dt)
    if norm(this.globalVector) < 1e-6 && ...
        norm(this.location-ground.nestLocation) > 1e-6
        this.goingToNestDirectly = false;
        this.problemEncountered = 'nestNotFound';
        this.pointNearbyToSearch = this.location;
        this.timeToSpendInSearch = inf;
        this.confidenceRegion = 5/dt;
        return;
    end
    if norm(this.globalVector) < this.velocityVector(3)*dt
        target = this.location+this.globalVector;
        this = this.stepStraightTo(target,dt);
    else
        this.velocityVector(1:2) = this.globalVector;
        this = this.updateLocation(dt);
    end
end

% This method look for something. What is looked for is stored in
% the "lookingFor" variable. Basically it performs a randomWalkStep
% if the ant can't see what is looking for, else it changes
% behaviour of the ant. The three options are 'food' or 'nest'.
function this = lookForSomething(this,ground,dt)
    if strcmp(this.lookingFor,'food')

```



```

        if norm(ground.foodSourceLocation-this.location) < this.viewRange
            this = this.stepStraightTo(ground.foodSourceLocation,dt);
        else
            % also if the ant can't see the food source, maybe
            % the ant can see a strong pheromone path
            auxParts = ground.getParticlesInRange(this);
            for i = 1 : length(auxParts)
                if norm(auxParts(i).location-ground.nestLocation)~=0 && ...
                    auxParts(i).intensity >= this.pheromoneIntensityToFollow
                    this = this.stepStraightTo(auxParts(i).location,dt);
                    return;
                end
            end
            this = this.randomWalkStep(ground,dt);
        end
    elseif strcmp(this.lookingFor, 'nest')
        if ground.isLocationAtNest(this.location)
            this.lookingFor = '';
            return;
        elseif norm(ground.nestLocation-this.location) < this.viewRange
            this = this.stepStraightTo(ground.nestLocation,dt);
        elseif this.goingToNestDirectly
            this = this.backToNestDirectly(ground,dt);
        else
            % also if the ant can't see the nest, maybe
            % the ant can see a familiar landmark pattern
            auxLandmarks = ground.getLandmarksInRange(this);
            if size(auxLandmarks,2) > 0
                this.landmarkRecognized = true;
                this = this.recognizeLandmark(auxLandmarks,ground,dt);
                return;
            else
                this.landmarkRecognized = false;
            end
            if ~isempty(this.problemEncountered)
                this = this.problemHandler(ground,dt);
            else
                this = this.randomWalkStep(ground,dt);
            end
        end
    end
end

% This method is a problem handler. If one of the other methods
% encounters a problem, it would pass the problem to this problem
% handler.
function this = problemHandler(this,ground,dt)
    if strcmp(this.problemEncountered, 'nestNotFound')
        if norm(ground.nestLocation-this.location) <= this.viewRange || ...
            this.timeToSpendInSearch <= 0
            this.problemEncountered = '';
            this.goingToNestDirectly = true;
        else
            % Random walk with constraints
            v = this.velocityVector(1:2);
            nd = this.pointNearbyToSearch-this.location;
            d = norm(nd)/(this.confidenceRegion+0.3);
            weightedVector = v./(d+0.00001)+nd.*d;

```

```

        weightedVector = weightedVector./norm(weightedVector);
        weightedAngle = vector2angle(weightedVector);
        angle = normrnd(weightedAngle,0.5); % choose an angle, with
            normal distr.
        yPart = sin(angle);
        xPart = cos(angle);
        this.velocityVector(1:2) = [xPart;yPart];
        this = this.updateLocation(dt);
        % end random walk
        this.timeToSpendInSearch = this.timeToSpendInSearch-dt;
    end
end

% Build an ant
function this = setUp(this,ground)
    v = ([rand;rand]).*2-1;
    v = v./norm(v);
    v = [v;0.125];
    this.velocityVector = v;
    this.carryingFood = 0;
    this.followingPheromonePath = 0;
    this.landmarkRecognized = false;
    this.viewRange = 2;
    this.pheromoneIntensity = 50;
    this.problemEncountered = '';
    this.globalVector = [0;0];
    this.storedLandmarksMap = [];
    this.lookingFor = 'food';
    this.prevLocation = nan;
    this.location = ground.nestLocation;
    this.pheromoneIntensityToFollow = 300;
    this.phi = 0;
    this.l = 0;
    this.pathDirection = [0;0];
    this.goingToNestDirectly = false;
    this.storedLandmarksMap = Hashtable;
end

end
end

```

A.2 Ground.m

```

classdef Ground
    properties
        nestLocation
        foodSourceLocation
        pheromoneParticles
        ants
        landmarks
    end

    methods

```

```

function inRangeParticles = getParticlesInRange(this,ant)
% Allocate enough space for the particles that could be
% in range. Then remove the space not used. This approach
% allocates the array just 2 times instead of *the number of
% particles in range*.
inRangeParticles = PheromoneParticle([1 length(this.pheromoneParticles)
]);
j = 1;
for i = 1 : length(this.pheromoneParticles)
    ph = this.pheromoneParticles(i);
    if norm(ph.location - ant.location) <= ...
        ant.viewRange
        inRangeParticles(j) = ph;
        j = j+1;
    end
end
inRangeParticles = inRangeParticles(1:j-1);
end

function inRangeLandmarks = getLandmarksInRange(this,ant)
% Allocate enough space for the landmarks that could be
% in range. Then remove the space not used. This approach
% allocates the array just 2 times instead of *the number of
% landmarks in range*.
inRangeLandmarks = zeros(size(this.landmarks));
j = 1;
for i = 1 : size(this.landmarks,2)
    lm = this.landmarks(:,i);
    if norm(lm - ant.location) <= ...
        ant.viewRange
        inRangeLandmarks(:,j) = lm;
        j = j+1;
    end
end
inRangeLandmarks = inRangeLandmarks(:,1:j-1);
end

function [bool particle i] = hasPheromoneInLocation(this,locationPart)
bool = false;
particle = PheromoneParticle();
particle.intensity = 0;
for i = 1 : length(this.pheromoneParticles)
    ph = this.pheromoneParticles(i);
    if norm(ph.location - locationPart) == 0
        bool = true;
        particle = ph;
        return
    end
end
i = 0;
end

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

```

```

        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
end
end

```

A.3 intensity2color.m

```

function [r g b] = intensity2color(intensity)
intensity = 255*5 - min(intensity, 255*5);
if intensity <= 255
    r = 255;
    g = 0;
    b = 0;
elseif intensity <= 255*2
    r = 255;
    g = intensity - 255;
    b = 0;
elseif intensity <= 255*3
    r = 255 - (intensity - 255*2);
    g = 255;
    b = 0;
elseif intensity <= 255*4
    r = 0;
    g = 255;
    b = intensity - 255*3;
else
    r = 0;
    g = 255 - (intensity - 255*4);
    b = 255;
end
r = r/255;
g = g/255;
b = b/255;
end

```

A.4 Hashtable.m

```

classdef Hashtable

```

```

properties
    keys;
    data;
end

methods
function hashTable = Hashtable
    hashTable.keys = {};
    hashTable.data = {};
end

function this = put(this,key,data)
    index = find(strcmp(this.keys,key));
    if isempty(index)
        if isempty(this.keys)
            this.keys{1} = key;
            this.data{1} = data;
        else
            this.keys{end+1} = key;
            this.data{end+1} = data;
        end
    else
        this.data{index} = data;
    end
end

function [angle length confidence] = get(this,key)
    index = find(strcmp(this.keys,key));
    if isempty(index)
        angle = [];
        length = [];
        confidence = [];
    else
        indexData = this.data{index};
        angle = indexData(1);
        length = indexData(2);
        confidence = indexData(3);
    end
end
end
end
end

```

A.5 pheromoneAnimation.m

```

function pheromoneAnimation(nAnts,nestLocation,foodSourceLocation,dt,steps,printFlag)
)
if nargin == 0
    nAnts = 4;
    nestLocation = zeros(2,1);
    foodSourceLocation = [7;7];
    dt = 5;
    steps = 500;
    printFlag = true;
end

```

```

ground = Ground;
ground.nestLocation = nestLocation;
nestPh = PheromoneParticle();
nestPh.location = nestLocation;
nestPh.intensity = 0;
nestPh = nestPh.setPrev(nestPh);
ground.pheromoneParticles = nestPh;
ground.foodSourceLocation = foodSourceLocation;
ants = Ant(zeros(nAnts,1));
for i = 1 : length(ants)
    ant = Ant;
    ant = ant.setUp(ground);
    ants(i) = ant;
end
ground.ants = ants;
for i = 1 : steps
    for j = 1 : length(ground.ants)
        [a g] = ground.ants(j).performCompleteStep(ground,dt);
        ground = g;
        ground.ants(j) = a;
    end
    cla;
    hold on;
    axis([-15 15 -15 15]);
    title('Pheromone-based orientation');
    xlabel('length [m]');
    ylabel('length [m]');
    ground = updateGround(ground,i,dt,printFlag);
    drawnow;
end

```

A.6 PheromoneParticle.m

```

classdef PheromoneParticle
    properties
        intensity
        location
        decayTime = 3;
        next
        prev
    end

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

```

```

function this = setPrev(this,prevPart)
    this.prev = prevPart;
end

function this = setNext(this,nextPart)
    this.next = nextPart;
end

function bool = isParticleInRange(this,ant)
    if isempty(this.location)
        bool = 0;
    elseif norm(this.location-ant.location) <= ant.viewRange && ...
        this.intensity >= ant.pheromoneIntensityToFollow
        bool = 1;
    else
        bool = 0;
    end
    bool = logical(bool);
end

% Tells if a particle is at the same location as the passed arg.
function bool = isParticleAtSameLocation(this,particle)
    if isempty(this.location) || norm(this.location - particle.location) ==
        0
        bool = 0;
    else
        bool = 1;
    end
    bool = logical(bool);
end

% Merges 2 particles at same location.
function [this particle] = mergeWhithParticle(this,particle)
    this.intensity = this.intensity + particle.intensity;
end

% The pheromone intensity has to decay. This method handle this
% behaviour. tc is the decay constant: smaller it is, faster the
% particles decay.
function this = decay(this,dt)
    tc = this.decayTime*30;
    this.intensity = this.intensity*exp(log(1/2)*dt/tc);
end
end
end

```

A.7 projectPointOnLine.m

```

%— Projects a point on a line passing in p0 and p1 —%
%—————%
function p = projectPointOnLine(q,p0,p1)
A = double([p1(1)-p0(1) p1(2)-p0(2);
            p0(2)-p1(2) p1(1)-p0(1)]);
b = double([q(1)*(p1(1)-p0(1))+q(2)*(p1(2)-p0(2));

```

```

    p0(2)*(p1(1)-p0(1))-p0(1)*(p1(2)-p0(2))]);
p = A\b;
end

```

A.8 randomWalkTest.m

```

function randomWalkTest
start = [0;0];
xs = zeros(100,1);
ys = xs;
for i = 1 : 100
    v = rand(2,1);
    location = [0;0];
    maxNorm = 0;
    for j = 1 : 10+i*20
        nd = start-location;
        if norm(nd)>maxNorm
            maxNorm = norm(nd);
        end
        d = norm(nd)/i;
        weightedVector = v./(d+0.00001)+nd.*d;
        weightedVector = weightedVector./norm(weightedVector);
        weightedAngle = vector2angle(weightedVector);
        angle = normrnd(weightedAngle,0.5); % choose an angle, with normal distr.
        yPart = sin(angle);
        xPart = cos(angle);
        v = [xPart;yPart];
        location = location + [xPart;yPart];
    end
    xs(i) = i;
    ys(i) = maxNorm;
end
plot(xs,ys);
title('Constant against distance from nest');
xlabel('constant');
ylabel('distance [m]');
end

```

A.9 twoArmGlobalVectorAnimation.m

```

function twoArmGlobalVectorAnimation(alpha,lengthArm1,lengthArm2,dt,printFlag)
if nargin == 0
    alpha = deg2rad(60);
    lengthArm1 = 10;
    lengthArm2 = 5;
    dt = 1;
    printFlag = false;
end

```



```

nestLocation = [0;0];
nodeLocation = [0;lengthArm1];
foodSourceLocation = nodeLocation + [lengthArm2*sin(pi-alpha);-lengthArm2*cos(pi-alpha)];

ground = Ground;
ground.nestLocation = nestLocation;
nestPh = PheromoneParticle();
nestPh.location = nestLocation;
nestPh.intensity = 0;
nestPh = nestPh.setPrev(nestPh);
ground.pheromoneParticles = nestPh;
ground.foodSourceLocation = foodSourceLocation;
ant = Ant;
ant = ant.setUp(ground);
ant.velocityVector(1:2) = [0;1];
ground.ants = ant;

target = nodeLocation;
currentPrint = 1;
while(currentPrint ==1 || ant.location(1) >= 0)
    ant.velocityVector(1:2) = ant.velocityVector(1:2)./norm(ant.velocityVector(1:2))
    ;
    ant.pathDirection = ant.velocityVector(1:2);
    ground = ant.releasePheromone(ground);
    if ant.carryingFood
        target = (ant.globalVector./norm(ant.globalVector)).*ant.l;
        ant.carryingFood = false;
    elseif norm(ant.location-nodeLocation) == 0
        target = foodSourceLocation;
    end
    ant = ant.stepStraightTo(target,dt);
    if ground.isLocationAtFoodSource(ant.location)
        ant.carryingFood = 1;
    end
    ant = ant.updateGlobalVector(dt);
    ground.ants(1) = ant;
    cla;
    hold on;
    minv = min([nestLocation foodSourceLocation nodeLocation]');
    maxv = max([nestLocation foodSourceLocation nodeLocation]');
    axis([minv(1)-2 maxv(1)+2 minv(2)-2 maxv(2)+2]);
    title('Path integration');
    xlabel('length [m]');
    ylabel('length [m]');
    plot(nodeLocation(1),nodeLocation(2),'bo');
    ground = updateGround(ground,currentPrint,dt,printFlag);
    drawnow;
    currentPrint = currentPrint+1;
    if currentPrint > 500
        break
    end
end
end
end

```

A.10 twoArmGlobalVectoTest.m

```

function twoArmGlobalVectorTest(maxAngle, minAngle, nMeasurements)
if nargin == 0
    maxAngle = deg2rad(170);
    minAngle = deg2rad(60);
    nMeasurements = 10;
end
assert(maxAngle > minAngle && nMeasurements > 0);
lengthArm1 = 10;
lengthArm2 = 5;
nestLocation = [0;0];
nodeLocation = [0;lengthArm1];
displacement = (maxAngle-minAngle)/nMeasurements;
xs = zeros(nMeasurements,1);
ys = zeros(nMeasurements,1);
ass = 1;
dt = 1;
for alpha = minAngle : displacement : maxAngle
    foodSourceLocation = nodeLocation + [lengthArm2*sin(pi-alpha);-lengthArm2*cos(pi-alpha)];
    ground = Ground;
    ground.nestLocation = nestLocation;
    nestPh = PheromoneParticle();
    nestPh.location = nestLocation;
    nestPh.intensity = 0;
    nestPh = nestPh.setPrev(nestPh);
    ground.pheromoneParticles = nestPh;
    ground.foodSourceLocation = foodSourceLocation;
    ant = Ant;
    ant = ant.setUp(ground);
    ant.velocityVector(1:2) = [0;1];
    ground.ants = ant;

    target = nodeLocation;
    step = 1;
    while(true)
        ant.pathDirection = ant.location-ant.prevLocation;
        ground = ant.releasePheromone(ground);
        if ant.carryingFood
            break;
        elseif norm(ant.location-nodeLocation) == 0
            target = foodSourceLocation;
        end
        ant = ant.stepStraightTo(target,dt);
        if ground.isLocationAtFoodSource(ant.location)
            ant.carryingFood = 1;
        end
        ant = ant.updateGlobalVector(dt);
        ground.ants(1) = ant;
        step = step+1;
    end
    xs(ass) = rad2deg(alpha);
    ys(ass) = rad2deg(abs(vector2angle(ant.globalVector)-vector2angle(nestLocation - foodSourceLocation)));
    ass = ass + 1;
end

```

```

end
plot(xs,ys,'b-o');
title('Error Against Angle');
xlabel('Angle In Degrees');
ylabel('Absolute Error in Degrees');
end

```

A.11 updateGround.m

```

function ground = updateGround(ground,currentStep,dt,printFlag)

removeList = zeros(length(ground.pheromoneParticles),1);

% Update the pheromone pixels
for i = 1 : length(ground.pheromoneParticles)
    phPart = ground.pheromoneParticles(i);
    [r g b] = intensity2color(max(phPart.intensity,...
                                phPart.prev.intensity));
    if ~isempty(ground.pheromoneParticles(i).prev.location)
        plot([phPart.prev.location(1) phPart.location(1)],...
              [phPart.prev.location(2) phPart.location(2)],...
              'color',[r g b]);
    end
    ground.pheromoneParticles(i) = phPart.decay(dt);
    if ground.pheromoneParticles(i).intensity < 20
        removeList(i) = i;
    end
end

removeList(removeList == 0) = [];
ground.pheromoneParticles(removeList) = [];

% 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');

    %— Used For Debug —%
    %—————%

    % Plot the walk direction
    % plot([ground.ants(i).prevLocation(1) ground.ants(i).prevLocation(1)+ground.
    % ants(i).pathDirection(1)], ...
    %       [ground.ants(i).prevLocation(2) ground.ants(i).prevLocation(2)+ground.
    % ants(i).pathDirection(2)],...
    %       'r');
    % Plot the next position
    % plot(ground.ants(i).location(1),ground.ants(i).location(2),'r*');
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
plot(ground.foodSourceLocation(1),ground.foodSourceLocation(2),'go');

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/currentResult/snap_',...
        zeroStr,int2str(currentStep),'.png'),...
        '-dpng');
end
end

```

A.12 vector2angle.m

```

function angle = vector2angle(v)
angle = atan(v(2)/v(1));
if v(1) < 0
    angle = angle+pi;
end
end

```