TERMES: Termite tracking in collaboration with Harvard University

Nikolaj Aaes, Niklas Schalck Johansson, and Hildur Uffe Flemberg
{niaa, nsjo, hufl}@itu.dk

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S

Abstract. The Self-Organizing System Research Lab at Harvard University wants to track termites as well as being able to extract relevant statistics about them. At the moment there is no existing software to fulfill this function which is why this project was developed. Tracking termites could enable biologists to investigate how they work together and what behavioral patterns they express. This can in turn help swarm robotics develop better collaboration between robots.

This project enables the user to control a XY-plotter and track ants and termites in a controlled enviroment and collect the relevant data. While the solution is just a prototype it provides the foundation for further development towards real field equipment for biologists.

Keywords: Computer Vision, Image Processing, Termite Tracking, Biology, Computer Science.

Table of Contents

1 Introduction 3
 1.1 Project Proposal 4
2 Vision based insect tracking 4
  2.0.1 Scope 4
  2.0.2 Requirements 5
 2.1 Experimentation with ants 6
 2.2 Tracking 8
  2.2.1 Theory 9
  2.2.2 Framework 18
  2.2.3 Realization 18
  2.2.4 Optimising the realized solution on Windows 22
3 Camera and Plotter Integration 23
 3.1 Setup 23
  3.1.1 Plotter 23
  3.1.2 Cameras 24
 3.2 Communication with Plotter 25
 3.3 Moving the camera 26
 3.4 Graphical user interface 27
  3.4.1 Statistics 29
4 Testing, evaluation and conclusion 29
 4.1 Process 29
 4.2 Reflection 31
 4.3 Testing the tracking software with real ants 31
 4.4 Threats to validity 36
  4.4.1 Threats to internal validity 36
  4.4.2 Threats to external validity 36
 4.5 Related Work 38
 4.6 Future Work 38
 4.7 Conclusion 39

1 Introduction

This report is written at the IT University of Copenhagen (ITU) in the fall term of 2013 as a project supervised by Kasper Støy from ITU and with additional guidance from Kirstin Petersen from Harvard University. The project was developed from 1. september to 16. december 2013 and was worth 15 ETCS points. The report is addressed to people interested in tracking ants and/or termites using image analysis.

The Self-Organizing System Research Lab at Harvard University has created a autonomous robot for tracking African termites in a lab environment but lacks the necessary software support. Working with and tracking these termites is cumbersome as they only thrive in environments that resemble their native environment and only when they are together with other termites. Therefore automatic tracking of specific termites is necessary, as tracking up until now has been done manually. Hardware has been created to handle this, however it lacks proper software support.

sidst linje her, har vi ikke lige sagt det?

This project aims to develop software that is able to track both ants and termites in a natural environment using a Hewlett-Packard XY-plotter provided by Harvard University. While it is already possible to communicate with the plotter using the program called Termite [1], there is a need for software that integrates tracking, plotter communication, statistics and a graphical user interface. The end product is to be used by biologists in the field and this software will lay the foundation to enable them to track ants/termites with more precision and with better data being collected.

A team at Harvard University previously experimented with using the plotter to track a single ant/termite on a white background with some success. While the details of this effort were unknown to us we decided to start this project from scracth. Since this team had some success with their approach we use their hardware setup as a starting point for this project. The focus of the project is mainly on the development of the software and the selected hardware has a secondary role.

kan vi få en ref til deres projekt?

Since tracking of ants/termites is very hard to test systematically we have chosen to evaluate it by describing in what instances we expect the tracking to work as intended and in what instances we expect it not to. The results show that the tracking of the ants/termites are both possible and satisfactory but still have situations where the it will fail. Both the results and possible solutions improvements can be found in section 4.3 Testing the tracking software with real ants

This project was undertaken as a substitute for the "Global Software Development" course at ITU and therefore has additional emphasis on the international collaboration and developement process. Section 4.1 Process will describe the tools used as well as evaluate the overall process.

This report assumes that the reader has a basic knowledge of mathematics but any prior knowledge about computer vision is not required. We will first describe the requirements and scope of the project and the tracking theory we either considered or used in the project. Then we will describe how we interacted with the XY-plotter and how we built the tracking and interaction together. Lastly will we describe the tests, analyze the results and conclude the project.

1.1 Project Proposal

The purpose of this project is to develop software, such that the provided hardware can be used to track ants/termites and analyze the output, as well as providing basic user input to the robot tracker.

There are three parts in this project:

1. Tracking termites using a low resolution camera. In order to track a termite, it is necessary to be able to determine its position and move the camera to its new position using image analysis on the camera output.
2. Interact with the tracking device and update the camera's position with the result from the tracking software.
3. Design and develop a user interface that can be used to retrieve statistical data from the tracking.

2 Vision based insect tracking

2.0.1 Scope

While the project lays the foundation for a fully useable field solution we have chosen to narrow our scope to the essential functionality. This means that the project was designed and tested in a controlled lab environment where factors such as lighting, wind etc. are kept as constant as possible.

Because actual termites were out of our reach and that our councillor at Harvard recommended it, we did all of our testing with normal ants instead of termites. We are confident that only minor adjustments, mostly to the blob detection code, are necesarry to make the solution work with termites since the main difference is the size.

Painting the ants in different colors can have some impact in the quality of the tracking. As a rule of thumb it is easier to track colors that contrast the background the most. In this project we chose to paint the ants white, red and bright green. We did not try any other colors because we found these to have a high degree of contrast which were sufficient.

indsæt billede eller ref til billede

The project will not focus on changing the hardware provided by Harvard. The hardware had already been succesfully used in a controlled and very simple environment, and this project focus on implementing the software to handle a more complex environment. However quality suggestions to how the hardware can be improved will be presented if the current hardware is found to be too limited in a new test environment, or if it would improve the results of the developed software.

While it could be interesting to track several ants we have chosen to focus on tracking a single ant in this project due to time restrictions. Tracking several ants can prove more difficult since one has to account for ants crossing each others' paths and would require more time to implement.

Our councillor at Harvard wanted the tracking and the interfacing with the tracking device to be implemented in C++ to make it easier for further developement. The graphical user interface was not restricted when choosing programming langauge and we ended up using C# and Windows Forms to implement it.

2.0.2 Requirements

To ensure the success of the project we compiled a list of the mandatory and optional requirements of the project. This helped us plan our development process and align the expectations between us, our councillor at ITU and our councillor at Harvard University.

of the projec to gang lige efter hinanden

Mandatory requirements

The mandatory requirements of the project are listed below and must be fulfilled for the project to be considered a success.

1. Tracking of ants and/or termites using a low resolution camera implemented in C/C++.
2. The ability to adjust the tracking parameteres before and during the tracking.
3. Moving the tracking hardware in correspondance to the tracking.
4. A graphical user interface containing two modes:
5. - Calibration mode. Must contain a direct feed from the lower camera, a proccessed feed and sliders to adjust the processing.
6. - Tracking mode. Must contain a direct feed from the lower camera, a direct feed from the overhead camera and statistics.
7. The ability to collect the following statistics:
8. - The route of the ant/termite over time.
9. - Heatmap of where in the petri dish the ant stay.
10. The ability to save the collected statistics.

Optional requirements

The optional requirements are the so called "nice-to-have requirements". They are not mandatory for the project but features that could be desirable to implement if the time and resources permits it.

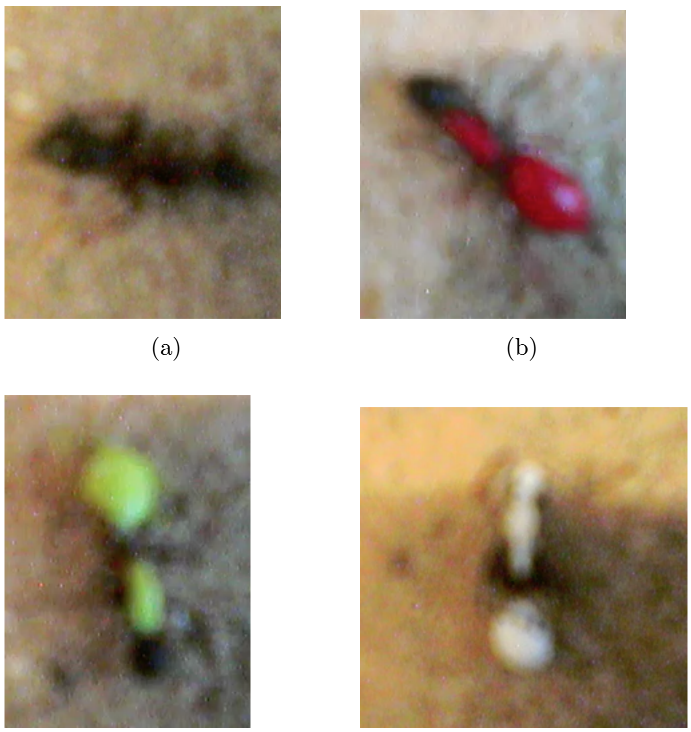
1. An additional third mode in the graphical user interface Bias mode: adds the ability to select a certain point which the plotter will try to lure the termites towards using food or pheromones.
2. The ability to choose certain areas which should be avoided during the tracking.
3. Collection of these additional statistics:
4. - Average speed of the ant/termite.
5. - The amount of ants/termites meet during the tracking. A way to adjust what defines a meeting.
6. - The amount of time between each meeting. A way to adjust when a meeting starts and ends.
7. - The duration of each meeting.
8. - The area of the petri dish covered by the ant/termite. A way to adjust how much area is covered by an ant/termite when stationary (also known as "headsize").
9. - How much area is covered over time.
10. - Mean free path (the amount of time between each "stay").

Now that we have established the background, scope and goals for the project we continue with describing the central parts of the project. First we describe how to find the ant/termite in an image, then how we interacted with the plotter and then how we created the graphical user interface.

2.1 Experimentation with ants

During this project we had to decide whether to paint the ants or not, and if so what color. Figure 1 shows four examples of ants, with both their natural color, shown in image *a*, and painted versions shown in image *b*(red), *c*(green) and *d*(white). The images are caught using the webcam from the plotter. We can see from this image that coloring an ant is preferred to not painting an ant, and furthermore painting it white gives the best contrast compared to the background. Furthermore we suggest putting the ant to be painted in a freezer for 2-4 minutes as this will put it in a sedated state, making it much easier to paint. This will not kill the ant and after a couple of minutes it will act normal again.

A second problem were to contain the ants within our petri dish. The ants had no problem climbing the sides of of our petri dish and escape. We therefore tried several different things to contain them. We were adviced to try teflon tape on the sides of the petri dish, as they should not be able to climb the slippery surface of that tape. This did not prove successful however so we tried other types of slippery tape versions like silicone and vax tapes. See the different types



(a) (b) (c) (d)

Fig. 1: Coloring ants in different colors

of tape in Figure 2. Unfortunately none of them were able to solve our problem. We were also suggested to try and use mineral oil, however we were unable to get a hold of any such oil. We tried with oil for cooking instead such as sunflower oil and olive oil, however they did not have the desired effect.

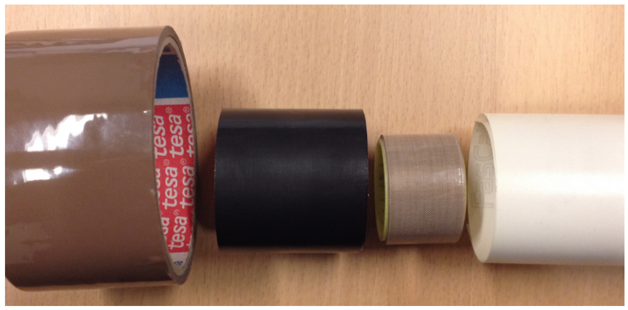


Fig. 2: Different types of slippery tapes used

We followed up by trying to use natural ant repellents such as kitchen salt, however our tests did not prove fruitfull. The ants traversed the repellent like it was not there.

Our solution was to eleminate all sides altogether and create an "island" instead. We filled the petri dish with ground until it was completely full, and placed it within a much larger petri dish full of water. The hope was that the ants would not dare move into water, and thereby stay in the petri dish during our tests. The setup can be seen in Figure 3. This solution proved much more successful, as the ant did not move into the water once it was detected. However after some time it realised that it was completely surrounded by water, and would try and swim away from our artificial island. But it would take a long time for the ant to try this, giving us enough time to test our software before we had to safe the ant from drowning.

2.2 Tracking

As mentioned in the project proposal the project was divided into three parts. The first part of the project was to be able to track termites. Since african termites are hard to come by and transport we settled for giant ants (Camponotus Ligniperdus), which are the largest ants found in Denmark [3], instead. While these are not as big as the termites, they behave in a similar way and the solution should be able to easily adapt to the termites. In the first part we started by implementing the tracking on a video. We received a video from Harvard University of a single ant running around in a petri dish with a white background. The ant

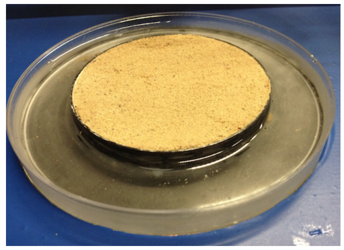


Fig. 3

itself was painted red and green to make tracking easier. This was the simplest setup we could think of and acted as a good starting point. By recommendation we decided to use the OpenCV [2] framework to implement our tracking. We will return to this framework in Section 2.2.2 Framework.

This chapter is divided into three subparts. First we will introduce the reader to different image processesing techniques, and their uses. Following this we will introduce the reader to a framework that supports these techniques, and finish off describing the implemented solution and design choices.

2.2.1 Theory

In this section we will use the *ternary if* statement notation in this section which looks like the one shown in Equation 1.

$$value = Boolean\text{-}Condition ? True\text{-}Evaluation : False\text{-}Evaluation \tag{1}$$

It works just like you would expect from many programming languages or mathematical expression. It is also known as an inline *if-statement*.

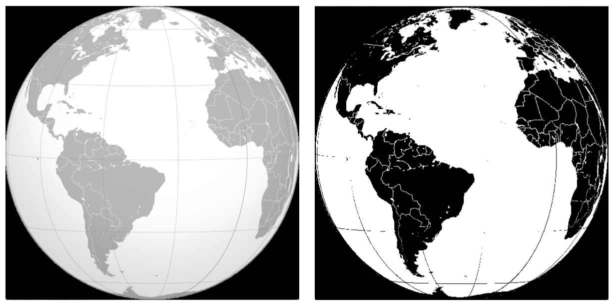
Thresholding

Thresholding is an image processing technique used to make a final decision about each pixel in an image. Thresholding is most commonly used to find objects in images, e.g. finding an ant in an image. During thresholding a decision is made based on the following; either a pixel value is one that we are interested in or it is not. This is usually done by assigning a specific pixel value to the pixel we want, and another to those that we do not want. In general we compare the *ith* pixel of the source image, *src*, to the threshold value, T, and saves the result in a destination image, *dst*. For instance, to create a binary image where we are interested in all pixels above the threshold T, the equation would look like the one shown in Equation 2,

$$dst_i = src_i \geq T ? 255 : 0 \tag{2}$$

Most threshold operations are applied on grayscale images, where all pixel values range between 0 (black) and 255 (white). Sometimes these values are slet denne sætning?

normalized to range between 0 and 1 instead. However for the rest of this report we will assume that grayscale images use the former convention. Soon we will argue how thresholding can be expanded to also cover thresholding an RGB image. Figure 4 show an example of applying threshold to a grayscale image.

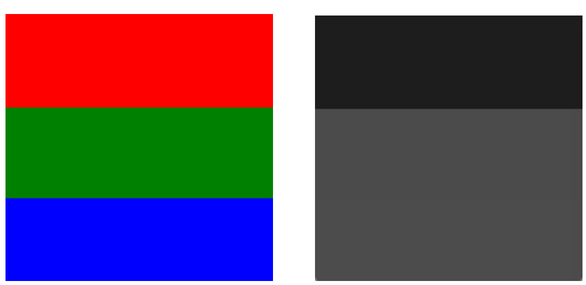


(a) Grayscale image (b) Thresholded image

Fig. 4: An example of applying a threshold on a grayscale image. This example use the threshold value $T = 200$

the source img? src?

Now what happens if what you are interested in is a color? To use standard thresholding we first need to convert it to a grayscale image. However doing so might result in an image that have lost important information as can be seen in figure 5. Unless you want to find the red color, you have no way of differentiating the blue and green color.



(a) RGB image (b) Grayscale image

Fig. 5: Example of grayscaling a color image

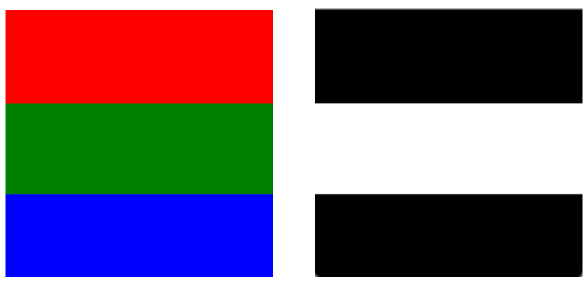
A step in the right direction would be to define the threshold value T as a scalar consisting of three values - one for each color channel. We will denote this threshold scalar as S and define it as shown in Equation 3.

$$S = \begin{pmatrix} S_R \\ S_G \\ S_B \end{pmatrix} \tag{3}$$

To apply it to an RGB image we need to change Equation 2 to the one shown in Equation 4.

$$dst_i = src_{iR} \geq S_R \wedge src_{iG} \geq S_G \wedge src_{iB} \geq S_B ? 255 : 0 \tag{4}$$

Trying it out makes it much easier to differentiate between colors. In Figure 6 a threshold attemp using this method directly on the RGB image is shown. this i stedet for trying it out



(a) RGB image (b) Thresholded image

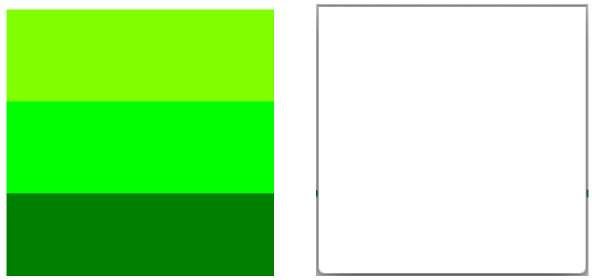
Fig. 6: Example of thresholding a color image with Equation 4 using the scalar S=(0,100,0)

However one issue remains - what if you want to find a color among similar colors? The problem is illustrated in Figure 7 using the same scalar as in Figure 6

The solution is to specify a *range* of acceptable values instead of just a threshold. We will specify two scalars; S Upper, SU, and S Lower, SL.

$$SL = \begin{pmatrix} SL_R \\ SL_G \\ SL_B \end{pmatrix} \tag{5}$$

$$SU = \begin{pmatrix} SU_R \\ SU_G \\ SU_B \end{pmatrix} \tag{6}$$



(a) RGB image (b) Thresholded image

Fig. 7: Example of thresholding a color image with Equation 4 using the scalar S=(0,100,0). Unlike before the result is unsatisfactory.

We will use the definitions in Equations 5 and 6 to update Equation 4. The changes can be seen in Equation 7.

$$dst_i = SU_R \geq src_{iR} \geq SL_R \wedge SU_G \geq src_{iG} \geq SL_G \wedge SU_B \geq src_{iB} \geq SL_B ? 255 : 0 \tag{7}$$

Using a range to specify a color instead will yield a much more satisfactory result as shown in Figure 8. In summary, both grayscale thresholding, and color thresholding can be used to find ants of certain colors in an image.

(a) RGB image (b) Thresholded image

Fig. 8: Example of thresholding a color image with Equation 7 using the range SL=(0,100,0) and SU=(1,150,10). We have sucessfully located the dark green color area.

Dilation and Erosion

The basic concepts of *morphological transformations*are *dilation* and *erosion*. ref?
These concepts are mostly used to remove noise, isolating individual elements or joining disparate elements in an image. Dilation and erosion are applied to either grayscale or binary image, and are often used to clean up an image after thresholding to make analysing the image easier. For example, when locating an ant, there might be many small light dots that make the ant hard to see, which can be much improved by using erosion or dilation depending on the image in question.

The way both erosion and dilation are applied to an image is by defining a kernel, denoted B, that will be applied to an image (or part of an image), denoted A. The kernel can be any shape or size, however it is often a square where the sides are uneven, e.g. 3x3, 5x5, 7x7 and so on. A kernel has an *anchorpoint*, which is the pixel in A that erosion or dilation are applied on, and the result is stored in the corresponding point in the destination image, *dst(i.j)*. Figure 9 shows an example of a kernel and its anchorpoint in the center.

src(i-1,j-1)	src(i,j-1)	src(i+1,j-1)
src(i-1,j)	src(i,j)	src(i+1,j)
src(i-1,j+1)	src(i,j+1)	src(i+1,j+1)

Fig. 9: Given a pixel, src(i.j) the kernel covers the neighbouring pixels. Shown here is a 3x3 kernel.

When transforming an image using dilation, whenever the kernel is applied to a new anchorpoint, the *local optima* is used. When eroding an image the *local minima* is used. Naturally, compared to the original image, bright areas are expanded when dilating, and dark areas are expanded when eroding. However one cannot say that e.g. eroding removes noice and dilating does not. It depends on the image which the transformation is applied on. Figure 10 shows an example of pixels used to determine the outcome of the anchorpoint in the destination image.

An example of how this applies to a real image is shown in Figure 11. It is clear from this example how differently erosion and dilation affects an image.

100	100	20
40	50	150
20	10	70

Fig. 10: An example of a kernel within a source image, src. The value assigned to the destination image, dst(i,j), is 10 if erosion is chosen, else 150 if dilation is chosen.

(a) Grayscale image (b) Eroded image (c) Dilated image

Fig. 11: Eroding and dilating an image. It is clear that eroding expands dark areas and dilate expands bright areas. This example uses a 7x7 kernel.

Unlike thresholding we cannot apply erosion or dilation to RGB images due to the nature of *RGB* values. Imagine the clear colors, red (255,0,0), green (0,255,0) and blue(0,0,255) - how can we argue which color is *greater* or *smaller* than the other? We cannot. We can with grayscale and binary images since we can safely say that the pixel value 150 is smaller than 255, and 0 is smaller than 1. We cannot for color images. In summary, we have presented techniques to make ants much clearer in an image with a lot of noise, that might prevent us from properly finding an ant.

har vi ikke sagt det for to sætninger siden

Contrast and Brightness

Where dilation and erosion sought to expand either dark or bright areas, contrast and brightness seek to increase the overall brightness or darkness of an image or the absolute difference between dark and bright pixels. This can make it easier to find an ant in an image where the image is either too bright, or too dark.

To increase the absolute difference between two pixels, or *increasing the contrast of the image*, each pixel in an image is multiplied with the same scalar, denoted α. Since images are basically matrices this method is equal to *scalar multiplication* shown in Equation 8.

$$\alpha \times \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} \alpha \times a & \alpha \times b & \alpha \times c \\ \alpha \times d & \alpha \times e & \alpha \times f \\ \alpha \times g & \alpha \times h & \alpha \times i \end{bmatrix} \tag{8}$$

To increase the brightness each pixel is added with a scalar, denoted β instead. This way the absolute difference between each pixel is kept, however all pixels get darker or brighter depending on the scalar. Similar to contrast this operation is equal to *scalar addition* as shown in Equation 9. To make the pixels brighter a positive scalar is used, to make it darker a negative scalar is used. Figure 12 shows the result of contrasting and increasing the brightness of an image. Contrast and brightness are used to make dark, bright and diffuse images easier to analyse. In summary, contrast and brightness can help improve the image quality, making it much easier to find an ant, or distinguish it from other parts of the image.

$$\beta + \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} \beta + a & \beta + b & \beta + c \\ \beta + d & \beta + e & \beta + f \\ \beta + g & \beta + h & \beta + i \end{bmatrix} \tag{9}$$

(a) Source image (b) Brightened image (c) Contrasted image

Fig. 12: Example of increasing the contrast and brigtness of an image. For this example $\alpha = 5$ and $\beta = 100$

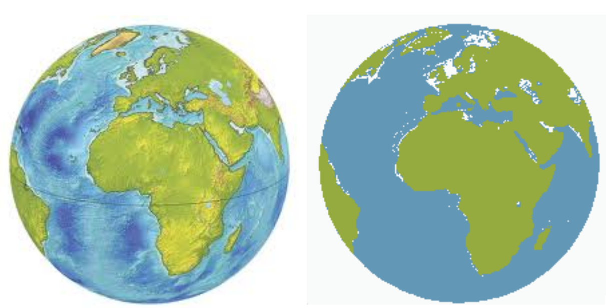
Image Segmentation

Image segmentation is the processing of partitioning all the pixels in an image into S segments (or superpixels). The goal is to simplify or change the representation of an image such that it is easier to analyse. It is mostly used to locate objects that consist of a range of similar colors, that is combined into a single colored object after segmentation, thus making it easier to find. Image segmentation is therefore the process of assigning a label to a pixel, such that such that pixels with the same label share certain visual characteristics. The result of image segmentation is a set of segments that covers the entire image. Thresholding, as described earlier, is the simplest method of segmenting an image. This can be an important technique in finding the ant, should thresholding alone not be good enough as more elaborate algorithms can be used to find the ant in an image.

One of the most common algorithms to segment images is the *k-means* algorithm. K-means is a *clustering algorithm* often used in *datamining* in the *unsupervised learning* category. What k-means basically does it to take a set of unclassified data, and classify it. It does so by assigning all the data to K different clusters, and iteratively assign data to the clusters it is most similar to after new *centroids* have been assigned. The algorithm terminates when data no longers changes clusters or a number of iterations have completed. Even though more robust classification algorithms exists, *k-means* is often used simply because each iteration is relatively fast. The algorithm works as follows:

1. Randomly selects k objects in the dataset D, which initially represents the centroid for each cluster.
2. Each object in D, is then assigned to the centroid it is most similar to.
3. For each cluster, it computes a new centroid from the previously assigned object and repeats (2).
4. If all clusters are unchanged between two iterations, the algorithm terminates.

An example of using k-means to cluster an image is shown in Figure 13. In summary, image segmentation provides a powerful way to easily simplify an image, possibly making ant detection much easier.



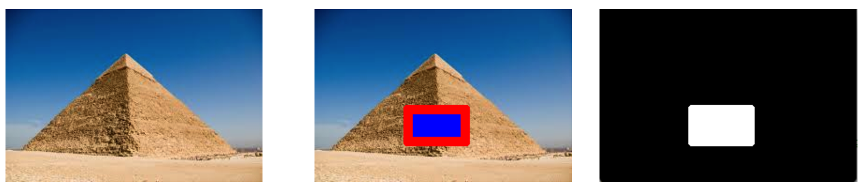
(a) Source image (b) Segmented image

Fig. 13: Example of segmenting an image. In this example the number of clusters specified is 3.

Background subtraction

Background substraction (also known as foreground detection) is a process often used to detect moving objects with a static camera. The idea is that the background remains roughly the same in every image in a video stream, while moving objects enter and leaves the stream shortly after, or become part of the background after some time have passed. The rationale is to have a background image (or reference point) and then substract the image with the new object in it. This technique can be useful for tracking a moving ant, assuming the background remains roughly the same in the test setup. Computing the difference between the foreground and background will result in an image with only the moving object in it, that can be used for analysis. The absolute difference is computed almost like a standard *matrix substraction* as shown in Equation 10. An example of background subtraction is shown in Figure 14.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} abs(a-e) & abs(b-f) \\ abs(c-g) & abs(d-h) \end{bmatrix} \tag{10}$$



(a) Background image (b) Foreground image (c) Object detection

Fig. 14: Example of detecting a new object on a static background. In this example all differences greater than zero is set to 255.

2.2.2 Framework

The theory behind computer vision techniques is solid. However to track an ant in an image solid theory and implementation is not enough. We have to keep in mind that this will have to be done on several consecutive images, without delaying the entire process, allowing the ant to escape the camera. Therefore effeciency is a key aspect as well. Chosing an effecient framework to help us track ants is therefore of outmost importance. OpenCV provides such a framework.

OpenCV is the shorthand for Open Computer Vision, and is an open source project started in 1999 and since maintained by Intel. It is written in C and C++ and runs on Linux, Windows and Mac OS X. OpenCV was designed from to beginning to be computationally efficient with a string focus on real-time applications. Another goal is to provide a simple-to-use computer vision infrastructure. Since its initial creation OpenCV has matured gradually, and along with it computer vision as well. Today OpenCV and computer vision is used in a broad context from web applications to survailance and aerial street maps.

Other computer vision frameworks exists but come short on several parameters. As stated in the requirements, tracking had to be done in `C/C++` which leaves out frameworks for higher level languages such as PyCVF for Python, imageJ for Java and OpenClooVision for C#. Furthermore several frameworks exist that extends OpenCV or creates a simpler abstraction layer such as SimpleCV and other frameworks offers a very limited amount of features such as VisionBlocks. As a result OpenCV was chosen to be the framework of choice, both because of maturity and effeciency, but also because of the amount of computer vision features offered to its users.

2.2.3 Realization

Having presented several computer vision techniques and the framework used, this section will focus on applying these techniques in a real world scenario to find an actual ant within an image from a webcam. To be able to track the ant of interest and make it distinguishable from other ants, we will paint it with a color that is easier to detect than the ant itself. We want to track the ants on real ground material, and not just a static colored uniform background, and the ants available to us are either dark brown or black closely resembling the ground material found throughout Europe. It is therefore important to choose a color that is easy to detect on such material. For this reason we have chosen a white paint for testing purposes.

The image in Figure 15 will be used as a reference point when showing how a certain computer vision technique performs in tracking the ant.

It is evident from the image in Figure 15, that the overall image is very dark. The ant itself is very hard to see, and even the white areas are dark. A way to improve this is to use the theory of *Contrast and Brightness* to improve the lighting condition of the image. Figure 16 shows the result of applying contrast

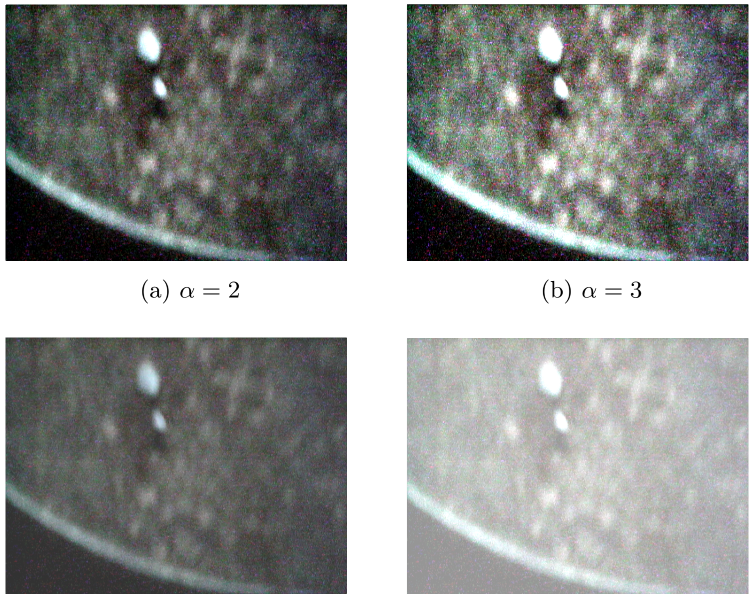
Fig. 15

or brightness to Figure 15.

It is clear from Figure 16 that image c and d does not provide a proper result. Even though the image is indeed brightened, differentiating between the ant and the background has not become much easier. Small changes to the lightning condition would probably also make image d much harder to analyse. Whether image a or b provides the best result comes down to what we want the most - that the object we look for is as close to clear white as possible, or that it is the most distinguishable object in the image. For colors in general, it is better for it to be distinguishable than trying to make it hit a specific color intensity. The result of image a is therefore preferable.

Now that we have improved the overall quality of the image, the next task is to isolate the ant in the improved image. For this task we have three techniques available; *thresholding*, *image segmentation* and *background subtraction*. Background subtraction have a major flaw in our case - the camera in question is in no way static. Furthermore, the area covered by the camera involves different color intensities of ground material, petri-dish edge and water. To create a background filter that would map to all these different materials, and at the same time take into consideration that they will never appear in the same part of the image, makes it almost impossible to do. In most situations there will be so many new objects in the image, that the ant might be the least changable object for every frame. This leaves us with image segmentation and thresholding.

Figure 17 shows examples of running image segmentation with different numbers of segments, K, defined on image a from Figure 16. It is clear from these tests that a certain number of segments are neccessary for a satisfactory result. $K = 8$ and $K = 10$ clearly gives the best results, however it comes at a cost; segmenting these images simply takes too much time. Even with only óne segmentation attemp for each image, it takes around one second or more to produce a result. Considering this needs to be done *multiple* times within a second when using a camera, this solution is simply not feasible or possible for tracking. An



(a) $\alpha = 2$ (b) $\alpha = 3$

(c) $\beta = 50$ (d) $\beta = 150$

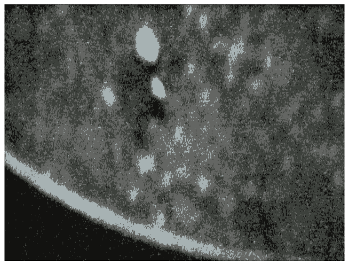
Fig. 16: Applying brightness or contrast to the ant image.

ant can easily move out of a frame without the camera ever knowing, simply because it is too slow or the ant is too fast.

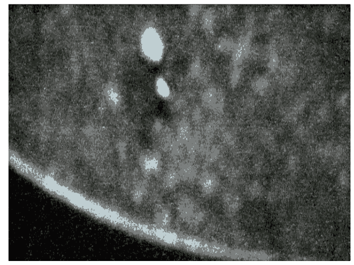
This leave us with only one option; thresholding. Figure 18 shows thresholding attemps on a grayscale version of image a from Figure 16. The results of different thresholds shows us two things; firstly that image d is a very good threshold result and secondly that we need a rather high threshold value to get this result. In contrast to segmentation, applying a threshold to our image is much faster, and usable for a steady camera stream.

Now that we have a satistafactory binary image that clearly shows the position of our ant, the next step is to actually get the position of the ant within the image. For this purpose we will use a *blob detection*. In short, blob detection is a computer vision technique encapsulating mathematical methods that are aimed at detecting regions in a digital image. These regions differ in properties such as brightness and color from the regions surrounding it. So where segmentation seeks to partition an image, blob detection seek to find the partitions. All the pixels in a blob are considered to share similar properties.

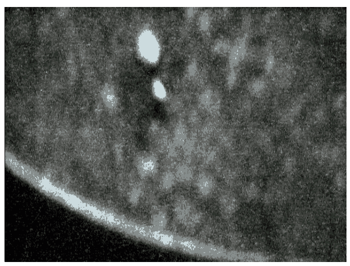
Considering the binary image d from Figure 18 clearly shows that we have to blobs available, that are rather large. This is of course ideal, but we might also have cases where we have smaller blobs caused by reflections on the ground or water. Therefore is is not enough to just use the first blob we find in an image



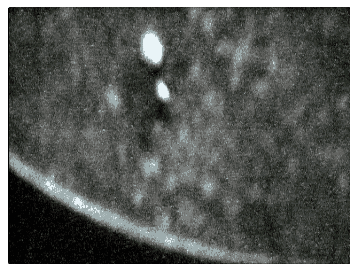
(a) K = 4



(b) K = 6

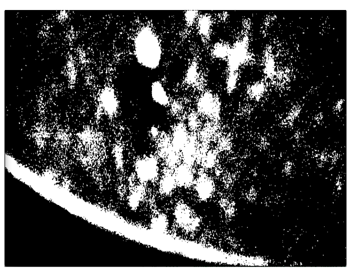


(c) K = 8

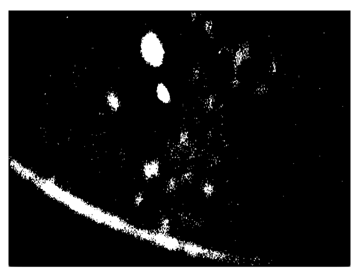


(d) K = 10

Fig. 17: Segmenting the ant image.

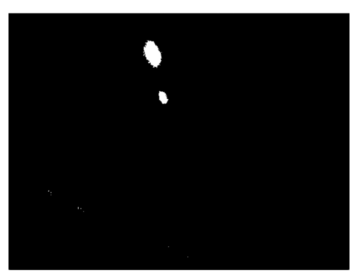


(a) T = 100



(b) T = 150

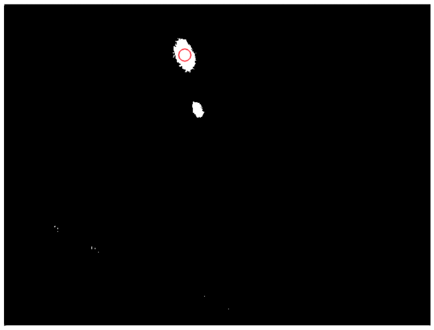
(c) T = 200



(d) T = 240

Fig. 18: Thresholding the ant image.

- we should always use the largest blob available in the image, which hopefully, is our ant. The result of finding the largest blob in our binary image is shown in Figure 19.



(a)
Detected blob drawn on thresh image

(b)
Detected blob drawn on original image

Fig. 19: Final result of finding the ant in our image. The detected blob is drawn on on both the thresholded image and the original ant image.

Concluding this chapter we outline the process of going from a raw image input to a successfully detected ant within an image, based on both the choices and experiments from this chapter. The process is as follows:

1. Increase the contrast of the raw image with an alpha value of at least 2.0.
2. Convert the contrasted image to grayscale.
3. Threshold the contrasted image with a threshold value of at least 240.
4. Run blob detection on the thresholded image.
5. Output the largest blob from the blob detection on the original image.

2.2.4 Optimising the realized solution on Windows

During development we observed a very strange behavior with the OpenCV code that implemented our realization described in the previous section. On OS X the processing of a single image took between 28 and 32 seconds which were satisfactory, however the same code performed much worse on our Windows test machine. This is strange when you also consider that the computer running OS X has much older and much slower hardware than the Windows machine. In comparison, processing the same image on Windows, with the same implementation would take between 200 and 300 ms sometimes taking close to 500ms for a single image. We ran several tests to try and find out why, and even when comparing the OpenCV installations themselves we found no difference in what features were enabled and disabled. We tried to see whether it was the machine itself, however we discovered the same behavior on a much stronger desktop machine running Windows as well where processing took between 160 and 220 ms.

To solve the problem we tried two things:

1. Compile the OpenCV source using CMake, instead of using the precompiled binaries available at the OpenCV download page.
2. Make use of the OpenCV CUDA GPU module that can be used to optimize image processing on computers with NVidia GPU's.

After compiling the OpenCV source code on the machine where it should be used, we found no difference in performance at all. We therefore changed the implementation to use the enabled GPU module and moved some of the processing to the GPU. This greatly improved the performance. Processing time went down to a stable 27-30 ms on our test machine, allowing us to do processing in real time. We cannot say why Windows performs differently than OS X, however this is a satisfactory solution to the problem.

del konklusion på hele afsnittet?

3 Camera and Plotter Integration

In Section 2.2, we introduced the theory behind tracking of objects in images. In this section, we describe how the PC, plotter and cameras cooperate, how we get our hands on a frame to process and how we use the techniques from Section 2.2 to turn a plotter, two standard web cameras and a PC into a tracking device.

3.1 Setup

The system as a whole contains four components

- Hewlett Packard 7046A X-Y Recorder (An XY Plotter)
- Printed Circuit Board (PCB)
- A Gigaware Web Camera 640x480 pixels (overhead camera)
- Web Camera 640x480 pixels (mobile camera)

3.1.1 Plotter The plotter is a 45x45 table with a mechanical control panel on the side (see Fig. XX in section YY). The plotter has a movable arm, moving along the X axis, consisting of a 30x3 cm of metal stretching all the way across the table. On this, a piece of plastic is attached, able to move up and down the arm along the Y axis. Together, these units cover the entire area of the plotter table.

The control panel is divided into labeled regions with sets of controls to manipulate specific parts of the plotter. Specifically, there is a region with controls for adjusting the zero point of the X-axis, one for the Y-axis, one for power supply etc. Most important of these is the two buttons that is used to adjust the zero point (0,0), as the software produced in this project will send only positive X and Y coordinates. Thus, the zero point must be located outside the area of the arena in order for the plotter arm to reach any possible location of the ant.

In order to connect the plotter to a PC, it provides an old-fashioned LPT parallel port interface. In this project, we use a printed circuit board (PCB) converting between LPT and USB. The board has two diodes that will light up when signals arrive at the USB side of the board. One of them flashes green no matter what data is received, the other one toggles between blue and no light when *valid* data arrives. Together with a PC and an RS232 terminal like e.g. Termite from CompuPhase, the diodes are useful for troubleshooting in case something is not working when running the TERMES software.

3.1.2 Cameras To track the ants present in the petri dish, the plotter is equipped with two low resolution standard web cameras connected directly to the PC through USB. One of the cameras is strapped on a piece of hard plastic holding the lens face-down, thus monitoring most of the plotter table from a height of approximately 30 cm. This camera is referred to as the overhead camera and is supposed to provide the user with the big overview, as well as a canvas to draw statistical overlays on, like heatmaps etc. The second camera is mounted directly on the plotter arm and follows the ant as it moves around in the arena. We will refer to this camera as the mobile camera.

Both cameras have a resolution of 640x480 pixels (less than 1 megapixel) as opposed to modern digital cameras with a resolution of more than 15 megapixels. Choosing relatively low resolution cameras has been a deliberate decision, as processing is done on a per-frame basis, meaning there is good reason to believe that a higher pixel density would increase processing time and thus prevent the camera from keeping up with the ant. On the other hand, the resolution should be high enough for the software to be able to identify the ant and/or the painted marker on its body. Fortunately, tests have shown that the mobile camera used in this project has been close to optimal with regards to image quality.

Another performance indicator for the mobile camera is the amount of frames it is able to record per second (FPS). To be able to maintain a stable tracking process, it is a key property that the ant is present in any two consecutive frames recorded by the camera. If this property is not held, it means that the ant is ahead of the camera, and the plotter will need to initiate a special procedure to relocate the ant. The camera needs at least as many FPS as needed for this to hold. On the other hand, more frames means more processing, which is also not good, but as long as the image quality is modestly low, this should not be a problem. Special techniques have been used to improve processing time, which we will discuss in Section XX. The specific mobile camera that was used in this project has a suitable quality, but could use a higher FPS value.

On a side note, it should be noted that the plotter is not new; the internal mechanics are sensitive and will some times cause the arm to do a fast wiggle. The effect of this is similar to taking a picture of a moving object with a DSLR camera whose shutter time is too large. The frame will be blurred and the pixels will "drag lines" that may introduce blobs similar to the one representing the ant. Thus, the plotter is in danger of being biased in a wrong direction and loose track of the ant. This problem is also likely to be reduced by using a camera with a larger FPS rate - or a newer plotter.

3.2 Communication with Plotter

In order to manipulate the plotter, we made a high level C++ API offering instructions like "Go to coordinate (x, y)", "move 10 units to the left", where left is relative to the camera view, etc. Since the coordinate system axes of the plotter and the mobile camera grows in different directions, such an API is very convenient to have in order to construct the logic that moves the camera when the ant leaves the center of the frame.

In order to implement this API, we needed to know the hardware protocol of the plotter. This was, however only given informally. Basically, the plotter accepts commands in the pattern given in Equation 11.

$$0x01 + xxxx + yyyy \tag{11}$$

In this format, $xxxx$ and $yyyy$ are two 4-digit hexadecimal numbers representing the X and Y part of the target coordinate. 0x01 indicates that we are sending something. Thus the plotter can receive coordinates between (0x0, 0x0) and (0xFFFF, 0xFFFF), however, any coordinate larger than (COORDINATES MISSING) lies outside the table area and will overflow and come back at zero.

To transmit the coordinates through the USB connection, we use a light weight open source C library (INSERT REFERENCE). This library is implemented directly on top of the operating system specific calls for manipulating I/O resources (`WriteFile()` in Windows and `write()` in Unix), thus treating the plotter like it was manipulating a file descriptor. The library contains procedures for opening and closing the connection as well as transferring data. When sending commands in the format of Equation 11, one should note that both the X and the Y part needs to be split in two parts of two hex-numbers each. Thus, the data passed to the library should be a 5-entry array of unsigned chars where the 0x01 part goes into the first entry.

Using this library, we developed a high level plotter interface that takes coordinates in decimal format and converts them to appropriate arrays that are sent over the connection. Furthermore, we added functions for moving the camera relatively to its own position by remembering the last requested coordinate. We even experimented with a special procedure for moving the arm in a soft fashion, as an attempt to keep the camera steady during movements to avoid blurry frames as mentioned above. In order to move the plotter some number of units from A to B, instead of going all the way at once, this procedure would start by first taking a small step of 2 units, then 4, 6, 8 and so on until it reaches a maximum step length (and stop in the same way with decreasing steps). Perhaps unsurprisingly, this did not solve much as the plotter can only move with one speed and will start and stop abruptly. Any step smaller than a few units though, will be completed too fast for the camera to blur the frames.

3.3 Moving the camera

Being able to move the plotter, the question remains of how to respond to changes in video frames. In the current version of the software, the plotter will stay centered above the ant. Conversely, this means that the ant should be found in the center of the frame - if this is not the case then it is either not in the frame at all (in which case manual interaction is needed) or it is somewhere along the inner edges, in which case we need to move the camera. To move the camera, the software locates the center of the frame along with the pixel coordinates of the ant, calculates the pixel difference in each direction, translates the distances to plotter units, adds the result to the current position of the plotter and issues a new "goto" command to move the camera. Unless the distance is smaller than approximately 5 units, the camera will blur when it starts to move, so we skip 2 frames, trusting that the 3rd frame will be steady. The choice of skipping 2 frames is chosen by experimentation.

In order to know how many units we need to adjust the plotter in each direction when the camera has identified the ant, we need a way to convert a pixel distance to a unit distance. When doing this conversion, we need to consider that:

- The conversion depends on the height of the camera and should be adjusted whenever the height is changed.
- The true distance of 1 unit in millimeters is different on the X and Y axis of the plotter. Thus, we need separate conversions for each.
- From the cameras point of view "right" points towards zero on the Y axis and "down" points towards zero on the X axis.

To make the conversion, we turned on the camera while sitting on the plotter. Next we layed out a sheet of graph paper under the camera and measured the true distance of one frame width (640 pixels) and one frame height (480 pixels). Next we measured the true distance of an arbitrary amount of units on the plotter. The rest is simple math.

As the kind of ants tested in this project is highly energetic and moves very fast, the camera sometimes looses track of them. We have observed this to be due to mainly three things. Firstly, since we skip two frames for every frame we process, the camera will always be two frames behind real time. From the view of the camera, this makes the ants run even faster than they actually do and it will be harder for it to follow along. Secondly, it takes some time to process each frame in order to isolate the ant and find out how to react when it moves away from the center - the more time it takes, the longer will the camera stand still and allow the ant to run off. In this project we reduced the processing time from about 200ms per frame to approximately 60ms using techniques we will come back to in Section 4.3. Finally, if the ant leaves the center, heads straight for one of the frame edges and then turns 180 degrees immediately after the camera begins to move, then both of the former problems are combined and will double their effects.

This last event happens, though it is rare, but it motivates a discussion of which technique to choose for moving the camera. We chose to keep the ant locked in the center of the frame, but other options clearly exist. One strategy could be to adjust the camera when the ant gets closer than a certain distance to the frame edges. Another one could be to take a chance and assume that whenever the ant runs in a certain direction, it probably won't make any sharp turns right away (say, bigger than 90 degreees). Thus, one could give the camera a head start by moving it along a straight line, headed for the new location of the ant, but instead of stopping when the ant is back in place at the center of the frame, we could continue moving the camera so that the ant ends up on the opposite side of the center. Unless the assumption did not hold[1], this should keep the ant in scope of the camera for a few extra frames.

We experimented with the strategy of moving the camera when the ant reached the edges of the frame but speed combined with frequently changing directions rendered the technique infeasible to use. It is possible that this situation would change if one modified the plotter and increased the height of the camera so it would cover a bigger area on the table. In that way the ant would stay in scope of the camera for a longer duration. We chose not to consider this option as keeping the ant in the center of the frame seemed to be relatively stable in the current setup.

3.4 Graphical user interface

In order to use the system to track ants, we need a user interface. The requirements for the user interface are the following

[1] To make such an assumption would propbably require a thorough analysis of ant behavior. Since the system we are building is intended to provide exactly that, this is slightly paradoxical but could potentially make the system able to improve itself!

- It should be able to show the camera feed from both the mobile and the overhead camera
- It should be able to calibrate the system before initiating the tracking procedure
- It should provide a way to gather statistics during the process of tracking
- It should provide a set of controls to manipulate the plotter manually

We chose to implement the GUI in Windows Forms using C# as Visual Studio provides a convenient way of dragging and dropping of UI controls. Furthermore, it was a requirement that all software would run at least on Windows but preferably on ther platforms as well. We experimented with a Swing GUI in Java, however we experienced some problems with calling into unmanaged C++ code on OSX using Java Native Interface (JNI) so we chose to switch to C# in order to save time.

Furthermore, it is easy to integrate unmanaged code with .NET as no libraries like JNI are needed. All C++ logic is published as a DLL with an interface represented in the GUI code as a list of static methods qualified with the "extern" keyword and called using P/Invoke. The GUI is shown in Fig. 20.

As seen from the figure, the GUI consists of a horizontal menu bar in the top, a video feed for the mobile camera on the left and a video feed for the overhead camera on the right. In the bottom, a panel is placed to display various statistics about the tracking process and the current performance of the system such as processing time, etc.

The GUI provides two modes - one for calibration and one for tracking. Switching between tracking and calibration mode is done through the "Mode" menu point in the top menu. When in calibration mode, the right camera feed will switch and show a threshold of the mobile view instead. Likewise, the lower panel will reveal a slider for setting the thresholdvalue. Thresholding is one of the key techniques behind the tracking and varies with the light and general surroundings of the camera, so it is important to be able to adjust it between sessions. The result of changing the thresholdvalue is visible in real time in the right camera feed and also during tracking if needed. Thus, it is possible to stay in calibration mode while tracking.

Ideally, the camera should be located above the ant when setting the thresholdvalue, as we are essentially telling the system how to identify the ant. To do this, open the manual steering panel by selecting "manual steering" from the plotter menu in the top bar. This panel provides controls similar to the arrow keys on a keyboard for manipulating the plotter. Using the keys and the camera feeds and provided that the ant is not moving too quickly, it is possible to adjust the plotter so the camera is located above the ant. Note: this process might require leaving the ant in the freezer for a couple of minutes beforehand. By default, the plotter will move 5 units each time a button is chosen. This number can be adjusted for each axis individually by setting the text fields in the right side of the window. Alternatively, use the controls in the top of the window to go directly to a specific coordinate.

When the camera is in place, return to the slider and adjust it until the ant is the largest white blob on screen. Ideally, the ant should be the only white object, but the system will look for the largest object so a reasonable amount of minor objects or "dust" are allowed to be visible as well.

When thresholding is in place, tracking is started by choosing "Start" under "Tracking" in the top menu. The camera will track the ant and show statistics in the panel until the user presses "Stop" or the ant is lost in the picture. In either case, the process can be repeated to start tracking again.

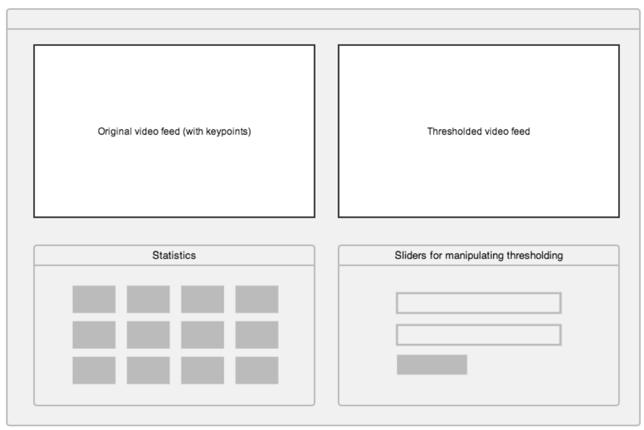


Fig. 20: Initial GUI draft

3.4.1 Statistics

TODO

4 Testing, evaluation and conclusion

4.1 Process

The developement process in this project required more planning than usual because of the long distance collaboration with Harvard University. This meant that we, at an early stage, sat down and agreed on a time plan, a preliminary table of contents for the report and which tools we would use to communicate and track our progress.

For communication we used regular email, skype and the Github [4] wiki and issue tracking system. Email allowed us to easily communicate with our councillor at Harvard even when either part was too busy for a skype meeting. It also helped to express the more formal questions about the project and the email correspondance was a good reference throughout the developement of the solution and writing of this report. We made sure to update our councillors weekly via email to inform them on our progress and problems.

Skype enabled us to have face-to-face meetings several times in the project. A summary of each meeting was created on the wiki both for own sake but also to make it easy for us, our councillor at ITU and our councillor at Harvard to track our agreements and progress. The skype meetings were more sparse than email correspondance because it was harder to agree on a time to hold them because of the time difference, and because they took more time which reduced the time spent coding. Since the Danish culture is a high-context culture (defined by Halls [5]) skype meetings was not greatly advantageous compared to pure email correspondance but it was nice to be able to discuss certain aspects and when a bundle of questions presented themselves at the same time it would save time to ask them in bulk. The fact that both we and our councillor at Harvard shared the same culture (specifically work culture) made the collaboration quite a lot smoother. This made the expectations of work hours, work load and final product align more easily along with avoiding any nasty confrontations as a result of suprising holdidays or customs etc.

The issue tracking system on Github were something we started out using but quickly abandoned. Since most of our work was done in the same location with all group members present the issue tracking systems primary function was to help our councillors track our progress easily. We quickly discovered that we were using a lot of time planning these issues and that our councillors were satisfied with the weekly email update we provided.

In general we were satisfied with the tools we used. The mix of verbal and written communication provided us with good channels to fulfill our needs and we feel that all of our questions were answered in a satisfactory way. Additionally we feel that our tools have helped us to inform our councillors of our progress and problems along the process in a satisfactory way as well. The fact that email is an asynchronus form of communication also helped us deal with the time difference between Copenhagen and Boston. We did experience any problems with this.

The learning outcome of this project regarding the process has been mostly about when to change direction when one encounters a technical roadblock. When we ecountered problems with using JNI and communication with the circuitboard we should have been quicker to search for alternate solutions or abandon the cross platform idea. This could have given us more time to implement more features in the final solution instead of struggling with technical problems.

4.2 Reflection

In this section we will discuss and reflect upon some issues that we believe have had a potential influence on the outcome of the project. These include a discussion of how the situation would have looked if we could have relaxed the requirement of using a mobile camera and how the project could have benefittet from using a different plotter.

Using a mobile camera is a requirement because users would potentially like to be able to stimulate the ant using a pheromon stick attached to the camera. However, excluding this requirement, we have no reason to believe after this project that it shouldn't be possible to reach the same goals using only the overhead camera. This would of course require an overhead camera with a suitable resolution and tests to verify that frames with the chosen resolution can be processed in a satisfactory amount of time. The relation between the resolution of the current cameras and processing times leads us to believe that it would indeed be possible to increase resolution without increasing the processing time to an unacceptable level. However, it of course requires testing to know exactly how much.

On the positive side, using only an overhead camera would get rid of the mechanical "wiggles" that cause the frames to become blurry, remove sounds that might affect the behavior of the ant and remove shadows from the mobile camera on the plotter table. Furthermore, it will increase (but not remove) the upper bound we need to put on processing time per frame as there is no mobile camera to lose track of the ant between frames. With only an overhead camera, it would even be possible to do tracking based on video files instead of a live camera input and thus eliminate all bounds on processing time (because all frames would be recorded beforehand).

Reducing the sounds from the plotter is also something we could have achieved by exchaining the plotter for a different model. Choosing a different plotter would also open possibilities for choosing a better model that has smoother moving arms.

4.3 Testing the tracking software with real ants

Having solved the problem of finding an ant on a single image and integrated our software with the XY-plotter and camera, this section focus on testing how the integrated solution works with a video feed of an ant running arond in a controlled environment.

The test were performed in a ordinary office environment, with the plotter placed on a table. The room were only lit by daylight, and we used the setup explained in Section 2.1, with a petridish filled with ground material surround by water, as can be seen in Figure (INSERT FIGURE WITH IMAGE OF SETUP). According to our experimentation in (INSERT REF TO EXPERIMENTATION) we painted our ant with a white color to have the best possible color for tracking. The goal of this test, is to test ant tracking in an environment that is as close to a real environment as possible, and yet by still be in a controlled environment where we could track the ant.corollary

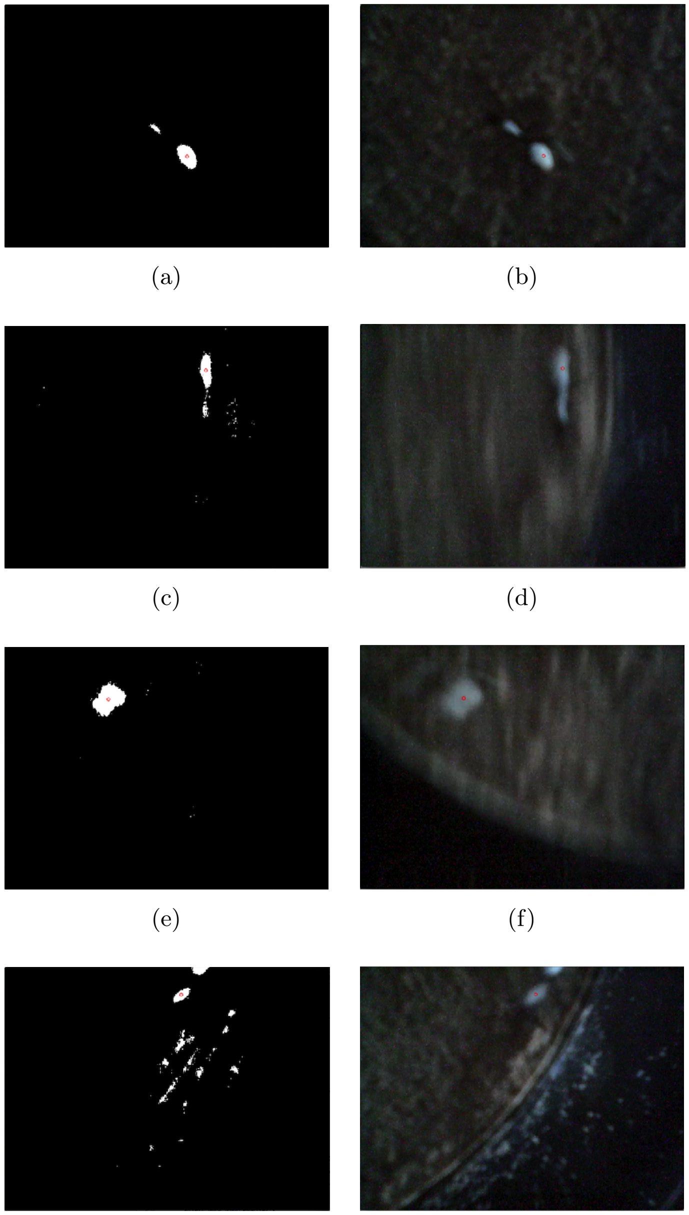
In the following we will show several results of running the software, both where the tracking works as intented, but also situations where the software does not work. We will end this section with a summary of the challenges presented by this real-time testing and suggest possible solutions to the problems and an overall evaluation of how the software together with the plotter an camera performs. We will also comment on the important observations done during testing.

We will begin by showing several images from the test where the software tracking works. Examples can be seen in Figure 21. We are showin both the final thresholded image, as well as the original image. For this test, following values were used $\alpha = 2.0$ and $T = 240$.

We can see from the images in Figure 21 that the software is able to handle different situations where a) the image is very blurry, b) there a noise in the thresholded images and c) where the ant is clearly visible. In general, we can say that it is possible to track the ant in situations where it is distinguishable from anything else in the image, or when it is the largest object present after image processing. However our tests also showed that at times we were unable to track the ant as shown in Figure 22.

In image b and d in Figure 22, the tracking fails because the ant is no longer the largest object in the thresholded images a and c. In image b it is because the gound reflects too much light, and in image d it is because the water reflects too much light. One might wonder why, especially in image c, the ant is considered smaller than the reflection. This is because the blob detector in OpenCV, assume blobs are *circular* and the size of a blob is actually the radius of that circle. Because the blob is lengthy but slim the radius of the circle surrounding the entire blob becomes very large, and as such the radius size reported by the blob detector as well. This leads to the problem that our software interprets the water reflection to be the largest blob in the image.

In image f the problem arises because the images is too dark. Even with then given contrast, most of the white color does no make the threshold boundary,

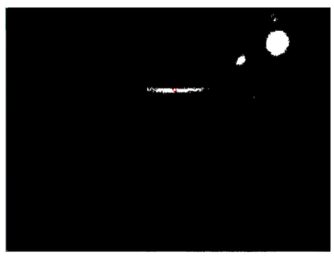


(a) (b) (c) (d) (e) (f) (g) (h)

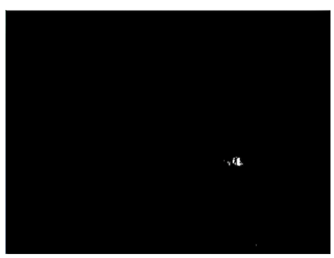
Fig. 21: Examples of real-time ant tracking.


(a)


(b)


(c)


(d)


(e)


(f)


(g)

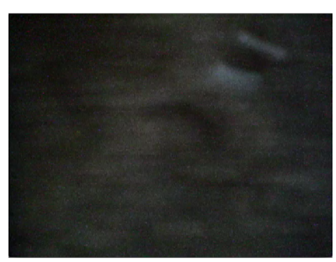

(h)

Fig. 22: Examples of tracking failues.

and the pixels that makes it past the thresold is considered noise by the blob detector. In image *d* the image is simply too blurry (the ant is in the upper left corner), which makes both the ant and the white color "disappear" into the background.

In summary the problems throughout our test can be generalized to cover the following issues:

- Background reflections
- Bad lighting of test area
- Camera blurryness
- Ant speed

To solve the issues of background reflections and light conditions a lab environment could be established were there would be no daylight involved, and have the entire test area be lit by diffuse light. This would ensure both a much better lit ant, and possibly also a solution to track an ant using other colors than white. Furthermore, reflections would not be as apparant due to the lack of a strong single light source, and would further reduce shadows from the arm of the plotter moving the camera around.

During the test it was noticed that the ants we had available in general moved very fast, and every so often they would move out of the cameras view in a second or two. But most of the time the camera could keep up with the ant, but because of the speed many of the images produced by the camera were very blurry, and at some point tracking would fail because the ant could no be located in the blurry images as shown in Figure 22. To solve this issue, one could use ants that moved slower, or move the camera further up from the ground, such that it did not have to move as much between every frame as in our case.

We do not know if termites moves as fast as the ants we have available, and if not, then they would be easier to monitor and follow with this setup. We also noticed during the test that the increased speed of the ants were often triggered when they were frigthened. We noticed this when we caught ants to be painted, and when released ants into the test environment. Over time they would slow down when they became relaxed (or so we assume).

During our test, the old XY-plotter would stutter from time to time, making large noises and shake the petri dish for a few seconds, and this probably caused a frightened reaction with the ant, seing as it started to move very fast again, escaping the camera due to blurry images. With newer and more robust hardware this could be avoided.

Another observation about the test is that the ant itself were almost never visible in the images (at least not to the human eye) and completely disappearing in blurry images, where the white color would still be visible. It is worth noting that this would really complicate tracking of *multiple ants* if the others are not painted as well. This would also complicate the solution further as the software would need to account for multiple colors at the same time, however with a proper lit lab environment this would be doable.

In conclusion, we argue that the software itself works as intended, with only a few situations where it is the cause of the failed tracking. The failed tracking can mostly be credited to a poor test setup, test environment, hardware difficulties and the animals themselves moving much faster than anticipated before the test.

4.4 Threats to validity

4.4.1 Threats to internal validity

While we believe our results to be purely causal there are always some threats to the internal validity that questions which factor could have an impact on the results. We describe the important ones here:

Behavior of the ants This project had a development period spanning over half a year. In this time the season changed and so did the temperature. The ants could have altered behavioral patterns, movement patterns and speed during the project essentially making the ants we tested on at the last part of the project, completely different from the ants we started out with.

Painting We painted our first ant relatively early in the development process and the ants could have had some reaction to the paint. We did use acryllic paint as recommended by our councillor, and we did not observe any noticeable changes but it could still be present.

Hardware Getting the camera in the same position every time is not something that can be done with 100% accuray every time. The small margin of error during positioning might have had a miniscule impact on the each tracking session.

Lights While we did try to do our testing at the same time of day with the same types of light each time it is very hard to keep completely controlled. Since light does have a significant impact on the tracking this threat to validity might the greatest but it is also something that can be hard to control.

4.4.2 Threats to external validity

The outcome of this project is highly dependant on the environment around the hardware and the ants/termites. This makes it very hard to replicate the results even with slight changes in things like light or camera resolution. This if course has the reprocussion that while the software works in theory and in a controlled environment it is unlikely to work in the natural habitat of any ants or termites. The following list contains the environment factors we consider to be important for the results:

Light The tracking relies on thresholded images which can be distorted by different levels of light which makes the tracking more inaccurate and more likely not to find the ant/termite.

Reflections Keeping a lid on the petri dish or filling the outer rim of the petri dish with water, like described in section 2.1 Handling ants, generates a lot of reflections. This is somewhat tied to the light factor and if the lights are positioned in such a way that the camera catches the reflections it can distort the thresholded image used for tracking.

Camera resolution The camera used had a faily low resolution of 640 times 480. While it could be nice to have a larger resolution, the low resolution enables us to process each frame fairly quickly. The threat to validity is mostly a performance one. If a better camera is chosen one might need to skip more frames or upgrade the connected PC since each frame will have a longer processing time.

Camera weight The camera attached to the plotter during our development and testing was very light. This made the plotter stutte less when it moves thereby generate less sound. Both sudden movements and loud noises can have an effect on the movements of the ants and if the camera is exchanged for a heavier camera it could impact the movement of the ants/termites.

Movement of the ants In our testing phase we observed that whenever we placed an ant in our petri dish the first thing it would do would be trying to escape. It would immediatly seek the edges of the dish regardsless of whether we had the rim filled with water or not. If the plotter was bigger or if the species of ants was different this might not have been the case.

Temperature, wind and humidity Ants, like any other animals, react to temperature, wind and humidity. If any of these factors are changed it could impact how the ants move and how they behave which in turn can impact the tracking.

Ants and termites Ants are not termites. This fact is something of a gap in the testing of the project. We did not have the opportuniy to test the software with real african termites and therefore settled for ants. While our councillor assured us that they behave in a very similar way we do not have any data supporting that this project will work with real termites, only a strong indication that it should.

All in all we are quite certain that the outcome of this project can be quite hard to replicate. This makes the project more useful as a theoretic base for further development than an actual tool to take into the field in its current state.

4.5 Related Work

TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction

Paper by Kirstin Petersen, Radhika Nagpal and Justin Werfel [6]. This paper presents a hardware system and high-level control scheme for autonomous construction of 3D structures. The hardware system consist of mobile autonomous robots and a collection of passive marked blocks. The general idea is to use swarm robotics to enable efficient construction even in extraterratial or disaster areas. The behavior of the robots is much like ants and termites in thaat sensing is entirely onbaord and each robot acts indenpendently of each other. Our contribution to the field can help enhance this project in the long run through analysis of the collaboration patterns of natures own swarms: ants and termites. By analysing how termites build their large hives one could potentially find a great way for swarm robots to collaborate when building structures.

Distributed Multi-Robot Algorithms for the TERMES 3D Collective Construction System

Paper by Kirstin Petersen, Radhika Nagpal and Justin Werfel [7]. This paper present the algorithmic part of the TERMES 3D collective construction system described in the paper above [6]. It describes algorithms for autonomous robot to build 3D structures both alone and in collaboration with more identical robots. Our contribution can potentially help enhance these algorithms. The movement of ants and termites could provide the key to better collaboration between the robots. Especially conflict resolution when two ants or termites need to occupy the same space on a path could be interesting to transfer into the robots' behavior.

4.6 Future Work

Because time was a limitid resource for our team there were features wi did not implement and ideas for additional functionality that could have been included. This section describes a short list of the possible extensions to the project we might have worked towrads given more time.

The first task would be to implement the optional requirements listed in section 2.0.2 Requirements. While some of these requirements are relatively easy to implement, given the right amount of time, some of them are more challenging. Tracking multiple ants can be difficult unless all ants are marked and this is something the current solution cannot solve.

When all these requirements were fulfilled it could be beneficial to support multi types of XY-plotters. The plotter we used was a little dated and to be able to switch to any XY plotter could be very practical. Of course this would also imply testing with an array of different plotters to see whether or not the same result could be produced. To support mulitple plotters one could implement plotter control as a service library to make it easy for other applications to control the plotter. This could expose an communication interface so it would be easy to switch the both the hardware and software of the plotter.

One of things we really wanted to do, but did not have the oppertunity to, was to test with real termites. The solution was design to easily be able to switch between different insects of different sizes and to test with real termites would be a strong indicator of how well this switch would work.

To help future development the implementation fo a developer console or log could be helpful. Developers could be able to manipulate the movement of the plotter through the terminal or interact with the tracking parameter. The log could save all the raw tracking coordinates to expose what data is recorded by the tracking and how the statistical data was created.

The graphical user interface could be enhanced by adding a color picker in addition to the RGB sliders that are currently present. Addtionally a third "Bias" mode could be added where an incentive (either food or pheromones) were attached to the camera, a specific point was selected via the GUI and the ant/termite would be led towards that point. This would open the possibility for even more statistics to be collected.

4.7 Conclusion

In this report we have presented software for tracking ants and termites in a controlled environment using the OpenCV computer vision framework and a HP 7046a XY-plotter. Additionally we have developed a GUI to control the tracking as well as extract a heatmap of the termites movement and the speed of the termite. We have tested the software and found that the tracking works in most cases but still have a small amount of cases where it fails, mostly due to the testing environment.

We argue that although the project has room for further development it contriubutes to the TERMES project in general and has potential to help biologsts analyse the behavior of termites in a more precise way than before. This analysis can in turn help the field of swarm robotics to develop better collaboration algorithms for robots in the future.

References

[1] Thiadmer Riemersma. (2013, December 2). *Termite: a simple RS232 terminal* [Online]. Available: `http://www.compuphase.com/software_termite.htm`

[2] Itseez. (2013, December 2). *OpenCV* [Online]. Available: http://opencv.org/

[3] Rune Larsen. (2013, December 2). *Kæmpemyrer (Campotonus ligniperdus)* [Online]. Available: http://www.fugleognatur.dk/artsbeskrivelse.asp?ArtsID=8063

[4] Nikolaj Aaes. (2013, December 2). *Aaes/TermiteTracker* [Online]. Available: https://github.com/Aaes/TermiteTracker

[5] J. & G. Olson. (2013, December 2). *Surprises in Remote Software Development Teams from lectures* [Online]. Available: http://dl.acm.org/citation.cfm?id=966804

[6] Kirstin Petersen, Radhika Nagpal and Justin Werfel. (2013, December 2) *TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction* [Online]. Available: http://www.eecs.harvard.edu/ssr/papers/rss11-petersen.pdf

[7] Kirstin Petersen, Radhika Nagpal and Justin Werfel. (2013, December 2) *Distributed Multi-Robot Algorithms for the TERMES 3D Collective Construction System* [Online]. Available: http://www.eecs.harvard.edu/ssr/papers/iros11wksp-werfel.pdf