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 want 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 jsut a prototype it provides the foundation for further developemnt towards real field equipment for biologists.

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

Table of Contents

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.

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 enable them to track ants/termites with more precision and with better data being produced.

A team at Harvard University previously experimented with using the plotter to track a singel 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 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.

TODO:
Skriv noget om outcome.

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 these 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 by biologists 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 also means that the project was designed and tested in a controlled lab environment where factors such as lighting, wind etc. are constant.

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 and bright green. We did not try any other colors because we found these to have a high degree of contrast which were sufficient.

sorry, bad excuse ;-)
maybe this is how you felt, but you could have gotten a 3d printer platform and two other cameras from my office. Maybe a bit of discussion about alternatives and then why you decided to use the hardware form

As the HP 7046a plotter was the only hardware available to us we have restricted the project to only include this type of plotter. One could potentially gain better results with a plotter with a finer granularity in coordinates and smoother movement. The cameras supplied with the plotter was also the only cameras available to us and we have therefore also restricted the project to only include these cameras. While one could use higher resolution cameras to obtain a better result, the weight of the camera is also important for the smoothness of the movement of the plotter and must be carefully considered.

While it could be intereseting 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.

TODO:
fortæl om hvordan vi IKKE tester

2.0.2 Requirements

Mandatory requirements

intro til reqs

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 using C/C++.
2. The ability to adjust the tracking parameteres before and during the tracking.
3. Moving the tracking hardware in corresponding 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").

need a conclusion here before you continue to the next chapter. Recap what we have learned so far and what that implies for the rest of your work.

2.1 Something with ants?

In this part we had to paint the ants with the paint we bought and we want to give some friendly advice to anyone who wants to do any projects involving ants. If you put them 2-4 minutes in the freezer they become very docile and much easier to paint without harming the ants. When they are taken out they act completely normal after 2-3 minutes. The ants also had a tendency to crawl up the sides of the petri dish. We were recommended to try putting teflon tape (should be really slippery) on the sides without any luck. We were also recommended to try mineral oil but we were unable to get any. Our solution was to place a small petri dish in a bigger petri dish and fill the gap with water. The ants would quickly discover that there was water surronding them which prevented them from escaping the small petri dish.

Skriv at myrer godt kan svømme! plus de problemer vi havde med at holde myrene inden for petri skålen.

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 of a single ant running around in a petri dish with a white background. The ant 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

This section will provide information about five common image processing techniques; thresholding, dilating and eroding, contrast, image segmentation and background filtering. The description of each technique will be backed up by examples. We will also use the *ternary if* statement notation in this section which looks like the one shown in Equation 1.

Rewrite chapter a bit - make it self contained. Keep writing WHY this is important and interesting to read and how it will help track ants.

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

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

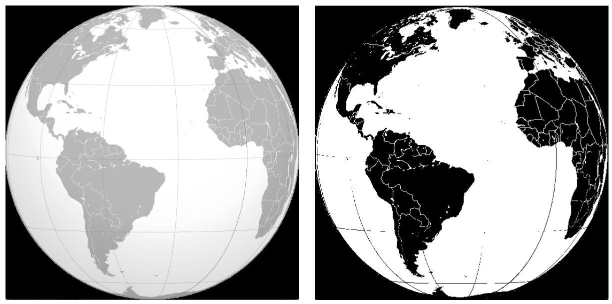
Thresholding

Thresholding is an image processing technique used to make a final decision about each pixel in an image. 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,

$$dst_i = src_i \geq T ? 255 : 0 \quad (2)$$

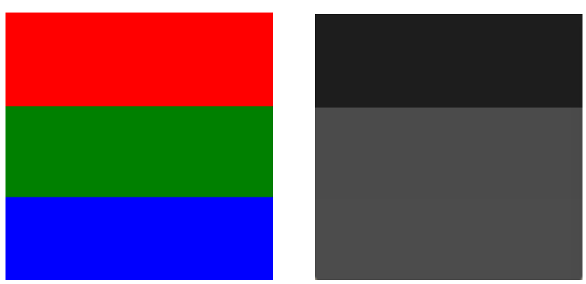
Most threshold operations are applied on grayscale images, where all pixel values range between 0 (black) and 255 (white). Sometimes these values are normalized to range between 0 and 1 instead. However for the rest of this report we will assume that grayscale images use the former convention. Soon we will argue how thresholding can be expanded to also cover thresholding an RGB image. Figure 1 show an example of applying threshold to a grayscale image.

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 2. Unless you want to find the red color, you have no way of differentiating the blue and green color.



(a) Grayscale image (b) Thresholded image

Fig. 1: An example of applying a threshold on a grayscale image. This example use the threshold value $T = 200$



(a) RGB image (b) Grayscale image

Fig. 2: Example of grayscaling a color image

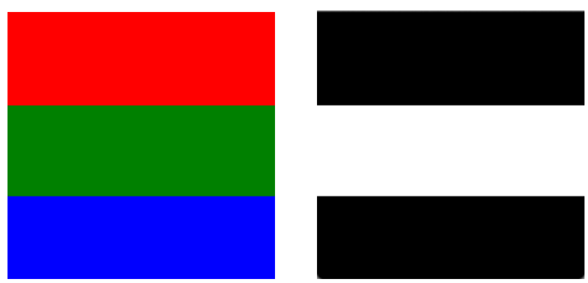
A step in the right direction would be to define the threshold value T as a scalar consisting of three values - one for each color channel. We will denote this threshold scalar as S and define it as shown in Equation 3.

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

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 3 a threshold attemp using this method directly on the RGB image is shown.



(a) RGB image (b) Thresholded image

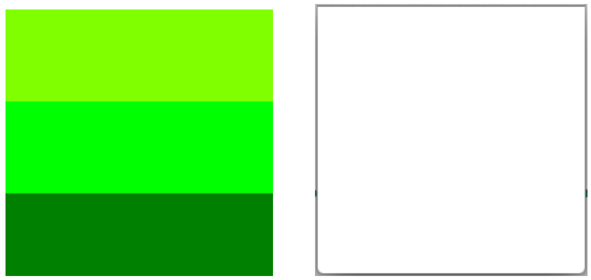
Fig. 3: Example of thresholding a color image with Equation 4 using the scalar S=(0,100,0)

However one issue remains - what if you want to find a color among similar colors? The problem is illustrated in Figure 4 using the same scalar as in Figure 3

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. 4: 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 5.

(a) RGB image (b) Thresholded image

Fig. 5: 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*. These concepts are mostly used to remove noice, isolating individual elements or joining disparate elements in an image. These transformations 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. 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 6 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. 6: 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 7 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 8. It is clear from this example how differently erosion and dilation affects an image.

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

100	100	20
40	50	150
20	10	70

Fig. 7: 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. 8: Eroding and dilating an image. It is clear that eroding expands dark areas and dilate expands bright areas. This example uses a 7x7 kernel.

cannot for color images.

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. To increase the absolute difference 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 9 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.

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

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

Fig. 9: Example of increasing the contrast and brigtness of an image. For this example $\alpha = 5$ and $\beta = 100$

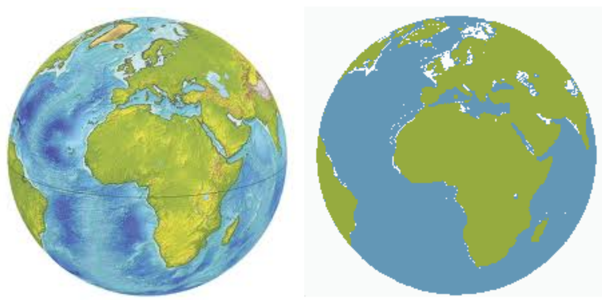
Image Segmentation

Image segmentation is the processing of partitioning all the pixels in an image into S segments (or superpixels). The goal is to simplify an 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.

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 10.



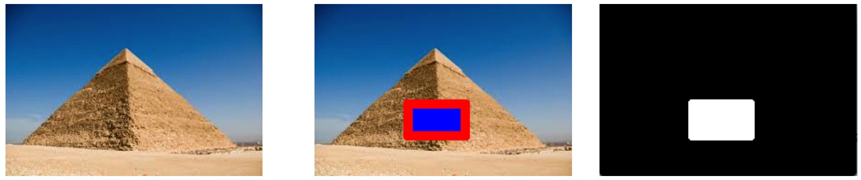
(a) Source image (b) Segmented image

Fig. 10: 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. Computing the difference 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 11

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



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

Fig. 11: 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 12 will be used as a reference point when showing how a certain computer vision technique performs.

Fig. 12

3 Camera and Plotter Integration

The second part of the project interacting with our tracking device and conform the tracking code to work with the device. We received a Hewlett-Packard 7046a XY-plotter by mail from our councillor at Harvard. Additionally we received a small ant farm, containing one queen and seven worker ants (no soldier ants), via mail and we bought some brushes and acryllic paint for the ants. The ants did not proliferate during the project. While we discussed using infrared paint or fluorescend paint, both these ideas were discarded because both types of paint contains compounds that the termites and ants really like which makes them eat the painted ant. Along with the plotter we also received a small circuit board that plugged into the serial port of the plotter and had a USB cabel for us to communicate with it. Lastly the package contained a collection of petri dishes for us to use.

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 an ordinary camera into a tracking device.

3.1 Setup

The system as a whole contains 4 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). 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 a diode that will light up when signals arrive at the USB side of the board. The diode will give a constant blue light in case of an error, but will otherwise flash green. Together with a PC and an RS232 terminal like e.g. Termite from CompuPhase, this diode is 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 are 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 (les 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, a high level C++ API was developed, offering instructions like "Go to coordinate (x, y)", "move 10 units to the left", where left is defined in terms of 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 (blah, blah) 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.

3.3 Moving the camera

Now that we know how to move the plotter, how do we move the camera when the ant moves? what about the volume of the movement sound? what about the speed of the movement?

3.4 Graphical user interface

What was the reqs for the GUI? How did we want it to look? What tasks do we expect the users to do (normal work flow)? Are we satisfied with the GUI (eval)? JNI skulle være cross platform men fungerede ikke med OpenCV. (nævn at det ville have været nice at have det cross platform) Vi har ikke noget der finder myren til at starte med. Hvordan kunne dette gøres?

3.4.1 Statistics

What statistics do we extract? How do the users do this? How are they produced? Can they be improved (future work)?

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

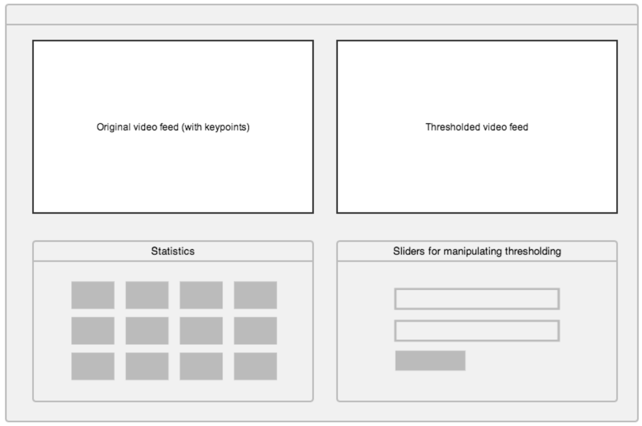


Fig. 13: Initial GUI draft

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

TODO

4.3 Evaluation

Overvej at kalde dette afsnit "Experimentation" eller "Testing"

vis hvornår det virker og hvornår det ikke virker. Se at det virker når vi regnede med at det virkede.

Hvad tester vi? Hvordan tester vi det? Virkede det efter hensigten? Hvorfor/hvorfor ikke? Er der nok testing? Hvordan kan man lave mere testing? Er der andre måder vi kunne have testet på? (fordele og ulemper ved det)

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 Different intensities of light may affect where the ants/termites wants to move towards. Additionally 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 SOMETHING WITH 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. ER DER NOGEN VANSKELIGEHEDER VED DET ELLER ER DET LIGE UD AF LANDEVEJEN?

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

TODO

Remember here to focus on how this has help the work on termite tracking moving forwards as opposed to discussion technical problems of you project.

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`

A First Appendix