UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andres Traumann

Semi-Automatic Method for Creating Virtual Reality Environments

Master's Thesis (30 ECTS)

Supervisor: Margus Niitsoo, PhD
Supervisor: Madis Vasser, MSc

Tartu 2017

Semi-Automatic Method for Creating Virtual Reality Environments

Abstract: Traditional virtual reality environments are based on rendering all the surroundings using 3D models of objects. This requires graphically modelling the whole environment, which is usually a long and tedious task. 360-degree cameras that have recently become commercially available might be able to change that situation, as image-based virtual reality requires no graphical modelling and therefore has the promise of being faster with the results resembling the real world more accurately. The downside of this approach is the requirement to create a separate image for each possible location in the scene that is being captured, thus resulting in a large number of images that need to be labelled, so that navigation between images would become possible. This thesis investigates possible methods to create such maps semi-automatically and provides an interface to display the virtual reality environments so created.

Keywords: Virtual Reality, 360-degree Imaging, Computer Vision

CERCS: P170 Computer science, numerical analysis, systems, control

Poolautomaatne meetod virtuaalreaalsuskeskkondade loomiseks

Lühikokkuvõte: Traditsioonilised virtuaalreaalsuskeskkonnad põhinevad kõikide ümbritsevate objektide renderdamisel 3D mudelitest. Selleks tuleb kogu keskkond graafiliselt modelleerida, mis on enamasti pikk ja üksluine tegevus. Hiljuti turule ilmunud 360-kraadi kaamerad võivad olukorda muuta – piltidel põhinev virtuaalreaalsus ei vaja graafilist modelleerimist ja omab seetõttu potentsiaali luua keskkondi palju kiiremini ning pärismaailmale palju sarnasemate tulemustega. Selle lähenemise puuduseks on aga vajadus jäädvustada iga vaadeldava asukoha jaoks eraldi pilt. Seega on lõpptulemuseks suur arv pilte, mis tuleb sildistada viisil, mis võimaldaks piltidevahelist navigeerimist. Lõputöö uurib võimalikke meetodeid selliste virtuaalreaalsuskeskkondade poolautomaatsel teel loomiseks ning kirjeldab kasutajaliidest, mis võimaldab neid kuvada.

Võtmesõnad: Virtuaalreaalsus, 360-kraadi pildistamine, Tehisnägemine

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

Introduction

Virtual reality as a new computing platform and medium is an exciting field. The idea of virtual reality goes back to the 60s [1], but a breakthrough in widespread usage has not happened. For a long time it was due to low computational power and expensive hardware. Nowadays, however, problems with raw computational power are starting to disappear. Computers are capable of rendering complex scenes with close to photorealistic quality at real-time speeds and devices like virtual reality glasses are now available in the market for prices affordable for a much larger audience. Nevertheless, virtual reality is not very common yet. One of the remaining problems hindering the widespread use of virtual reality products is content - creating believable VR environments is difficult and time consuming, as the traditional methods rely on graphical modelling, which makes the development process slow even for a team of talented professionals.

This thesis explores the options of an alternative way for creating photorealistic virtual reality environments by making use of 360-degree imaging. A remotely controlled car is used for moving the camera and capturing a video in the target location that is going to be mimicked in the virtual reality environment, which is going to be created by the algorithm described in this thesis.

The main contribution of this thesis is the method of creating a navigable map from 360-degree images. The remaining parts describe the virtual environment navigation system along with a brief discussion about the 360-degree cameras, unorthodox image formats for holding 360-degree visual information, and the process of stitching images together.

The rest of the thesis is organised as follows: Chapter 1 gives a brief overview of the background of the problem and briefly describes related works. Chapter 2 introduces the 360-degree imaging process and Chapter 3 describes the image stitching process. Chapter 4 describes the map generation algorithms. Chapter 5 talks about the computer graphics methods that are used for displaying the virtual reality scenes, and Chapter 6 talks about the implementation details of the system.

1 Background and Related Work

Traditionally virtual reality has been closely connected to the computer graphics pipeline. To be able to show the virtual world, it would need to be modelled first. Those models could then be rendered with high efficiency. In that kind of approach, how realistic the world looks depends on the goodness of the model. The problem is that graphical modelling requires a lot of time and experience. To be able to make the process fast, an entirely different approach is needed.

One possible alternative is based on photogrammetry methods [2]. Creating a 3-dimensional model from images is a standard computer vision problem that has been extensively researched for decades. As a result there are numerous approaches, but overall it is a very complex problem. In recent years a lot of progress has been made and it can become a serious alternative in the future. Since it uses quite different methods, going any deeper into this is outside the scope of this thesis.

This thesis explores yet another option: It is possible to not create 3-dimensional models at all and to replace 3D-rendering with 360-degree images, that way photorealism can be achieved even without doing any graphical modelling. Obviously it only includes scenarios where the aim is to create a virtual environment mimicking some real world place.

The disadvantage of the image-based approach is the fixed viewpoint. When the environment is modelled it is easy to simply move the view point, turn it around in all kinds of different directions, everything would still work, as the surroundings are modelled and can be looked at from every direction and location. In recent years this disadvantage has been rapidly disappearing, however, as a new type of camera that is able to record 360-degree images has reached the market. This new development has effectively removed half of the previously mentioned disadvantage, as one 360-degree image contains enough information to generate a look-around simulation. Yet, the camera is still fixed to a single location, so part of the problem remains.

The solution to the fixed camera problem in image based virtual reality can be solved by capturing a large number of adjacent images, so that they would densely cover the whole area that should be covered in the virtual reality environment. This seemingly simple task has many unexpected problems that need to be overcome. The camera that captures 360-degree images needs to be operated from a distance, for otherwise a person controlling it would himself/herself appear on the image. This means that someone always needs to place a camera in the required location and go far enough so he/she would not be visible before capturing an image. As

this is very tedious to do for a large number of images, it would make sense to create a system where the camera could be relocated from afar in some way.

The ability to create virtual reality environments of the existing locations in the real world has a large number of applications. The internet is full of images displaying various locations. In many cases the ability to look around and move around in these places would be a great improvement over an image. Google Street View [3] lets people see the world without leaving home. In a same way the described method could be used for creating virtual environments that would let people look around inside large shopping centres, people looking for a place to live compare rental apartments without going there in person, restaurant owners attract customers by showing how nice and comfortable their place is by letting potential customers look around the restaurant before actually going there. Basically everything that is showing or promoting something could make use of such a system.

Our approach has 3 main steps: image stitching, map generation, and rendering. We will now discuss the current state of the art in all of them.

1.1 Image Stitching

In this work image stitching is done using software provided by Samsung [4]. The exact implementation details of the algorithm are unfortunately not known so some effort was put into becoming more familiar with how the process could work in general.

The image stitching problem is studied extensively and numerous methods exist, mainly in the context of stitching together a panorama image. The method is developed well enough so that nowadays it can be done on cellphones in nearly real time. The methods operate on a variety of different conditions and assumptions, but the main workflow is usually quite similar. Using a feature extractor like SIFT, a number of useful features are extracted. Then common features are determined using Ransac or some similar method. The methods often make use of a camera matrix approximation for better results. According to the common features, the final image is constructed and finally the colours near the image borders are blended together for a better visual appearance. Examples of such methods include [5], [6], and [7].

One of the problems, however, is that most of the general techniques can not be used directly, as they are designed to work on conventional images, making them hard to use for spherical fisheye images. Additionally, these methods are in fact operating on unnecessarily wide assumptions. For example, it is not necessary to

estimate the distance between the camera locations, as they are always fixed in 360-degree cameras. Also, the camera parameters are already known beforehand and camera matrix estimation is not necessary. The common areas between the images are also quite well known due to the previous reasons. It means that the problem is in fact simpler than the general case, but needs a somewhat different approach. The problem is usually solved with methods quite similar to the ones operating on conventional images. In the work by Schleimer and Segerman [8] a good overview of spherical to equirectangular conversion is given, and an example of image stitching methods for spherical images is presented in the work of Deng et al. [9]. It should be noted, however, that there does not seem to be much research on spherical image stitching, possibly because it is a somewhat recent problem.

1.2 Map Generation

We are currently not aware of any research on the exact topic discussed in this thesis. However, there is a wide range of research on problems somewhat related to this work.

Perhaps the most relevant is a well-known work by Snavely et al. [10]. It describes a virtual reality system capable of exploring through large image collections using camera location estimation based on SIFT features. It tries to achieve a similar objective to this thesis: based on a large set of arbitrary images, it tries to make a navigable virtual reality space. The approach is far more general compared to the objective in this thesis, and as a tradeoff for achieving it, the system is much more complex.

Image retrieval is the task of exploring very large image data sets, attempting to find images resembling the one given as input. This task is relevant to our work, as it is often achieved by mapping all the images in the dataset into some lower-dimensional space, and in this thesis we are also trying to map images into a 2-dimensional space. Additionally they often propose novel image browsing systems that resemble the virtual reality navigation system described in this thesis. For these reasons, it is possible that this field of research might contain some useful techniques. Examples of such work are done by Rubner et al. [11] and Sivic et al. [12].

Another seemingly quite similar work is done by Endo et al. [13], and of course in the famous Google Street View system [3]. Nevertheless, they are not actually creating any maps, as road maps already exist. Also, such maps would be easier to construct due to the linear structure of roads.

There is also a wide area of dimensionality reduction embedding problems that

share the same goal of trying to visualise high-dimensional objects in a lower-dimensional space, so that it would retain much of the structure. Examples of such methods include Multidimensional scaling [14], Isomap [15], Locally Linear Embedding (LLE) [16], Laplacian eigenmaps [17], and Manifold sculpting [18]. Some of these ideas are explored in Chapter 4.

Another potentially relevant class of algorithms are Graph Embedding algorithms. They attempt to find such low-dimensional embeddings that would be useful for visualisation and have a minimal number of edge crossings. In this work, the image location would form a graph, so the methods surveyed in [19] could be useful.

1.3 Rendering

Rendering 3-dimensional objects is a heavily researched topic, and as a result there is a standard well-working pipeline that can be used, which the computer graphics textbook by Shirley et. al [20] covers in detail. For the purposes of this thesis, the standard method is more than sufficient. Conventional virtual reality environments depend on rendering efficiency much more, as they need to draw the whole scene from scratch using the modelled objects, whereas the image-based method described in this thesis only needs to render a single sphere.

Rendering 360-degree panoramas has been done previously as well. The predominant method also used in this work is texture mapping the full 360-degree image onto a sphere, although mapping it to a cylinder and even a cube has also been tried, as it slightly depends on the exact camera system. For example systems, where the full 360-degree view is not available, might find cylinder mapping an interesting alternative.

Another interesting and quite novel approach is to use a light-field camera. Contrary to a conventional camera, it captures light ray direction in addition to light intensity [21]. It allows to capture images from a single point, yet move around in a small radius of virtual space later. The technology is still in early stages and quite expensive, but it is a possible improvement over the 360-degree image based system.

2 360 degree Imaging

In the last two years, many low-cost 360-degree cameras have appeared on the market. Previously, creating 360-degree videos required complicated setups of multiple cameras or some other very high-cost equipment. Now, however, there are many such cameras affordable even for hobbyists.

360-degree images need to be handled slightly differently compared to images from ordinary cameras. Most conventional cameras output images that more or less correspond to the way we see the world. The view angle is at most 120 degrees in most cases, so an image looks similar to what we would see with our own eyes. 360-degree images, however, show the world differently: all the surroundings are visible on the same image at once. For a human observer that kind of situation is hard to comprehend and it would be much more meaningful to show only a subset of that image corresponding to some narrow view angle. By selecting a different subset the view angle would change and some different part of the scene would become visible. This system can be made interactive by letting the user decide which direction to look at.

There are many slightly different ways of storing the 360-degree image data. Initially a camera captures images in an ordinary fashion by looking at the surrounding through a lens and projecting everything onto an image plane. This representation often needs to undergo a stitching process where it is combined with other images to produce a combined image of slightly different format. There exist a few different formats for storing the final 360-degree image. They are discussed in Subsection 2.2.

2.1 Cameras

While similar systems have certainly been created before, they were not easily available for average hobbyists. In the recent two or three years, however, there has been a sudden emergence of low price 360-degree cameras: Ricoh Theta S, 360fly, Samsung Gear 360 (see Figure 1), LG 360, Kodak PIXPRO SP360, Bublcam, Sphericam, probably many more. This is a brand new field of active industrial competition, promising a rapid advance of virtual reality.

The cameras share many common design principles. They all make use of ultra wide angle lenses ranging from 190 degrees used in Samsung Gear 360 to 240 degrees in 360fly. The number of lenses varies, but most often, there are two lenses that are able to capture the whole 360-degree view. 360fly and Kodak PIXPRO SP360 feature only one lens and do not in fact capture the whole 360-degree

Figure 1: Samsung Gear 360 camera [22] seen from two different view points

view, but a large subset of it, choosing to focus on action camera capabilities. Kodak PIXPRO SP360 is supposed to survive up to 2 metre drops and extreme temperatures, while 360fly can work under water. Some cameras have even more lenses: 4 for Bublcam and 6 for Sphericam.

The resolution of such cameras is still somewhat low. At first such cameras could only capture 720p images, but by now 4K images has already become a standard. Note, however, that while 720p images might not sound like a very low quality, it should be taken into account that only a subset of the image is shown at the same time, so the actual perceived resolution is much lower and might be closer to 480 x 360, which is certainly very low. Nevertheless, given the recent pace of development, it is likely that the resolution will quickly also increase to much more reasonable quality.

2.2 Projections

Images that are captured by an ultrawide fisheye lens camera are shown in Figure 2. Together these two circular images form a full 360-degree image, as both are covering approximately 190 degrees. For one single fisheye lens that kind of representation could be used directly in applications, but when there are multiple such images, then they need to be combined and stitched first before being used. The stitched version of the image is usually converted to a format that would be slightly easier to use in practise.

Figure 2: The world seen through the wide angle lenses of Samsung Gear 360 [22] camera

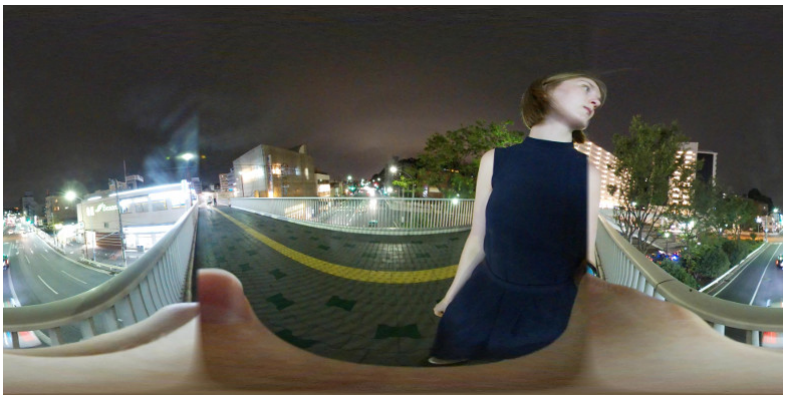


Figure 3: Equirectangular image created from the images of Figure 2

A common choice for storing stitched images is the equirectangular image format. The fisheye lens images on Figure 2 were converted to that format and are shown in Figure 3. Such images can be created using the equirectangular projection. Another well-known use case of this projection is for mapping the surface of the globe to 2-dimensional maps that we are used to seeing in geography. A downside of this format is the large amount of redundancy near the top and bottom edges of the image. We can see a similar effect when looking at the Antarctic on the world map, it looks narrow and stretched out in the bottom side of the map despite having a rather circular shape in reality.

An alternative to the equirectangular format is the cube map format. The previous images converted to that format are shown in Figure 4. The main advantage over the equirectangular format is redundancy removal. Previously, the higher and lower parts of the equirectangular image contained a lot of redundancy due to the

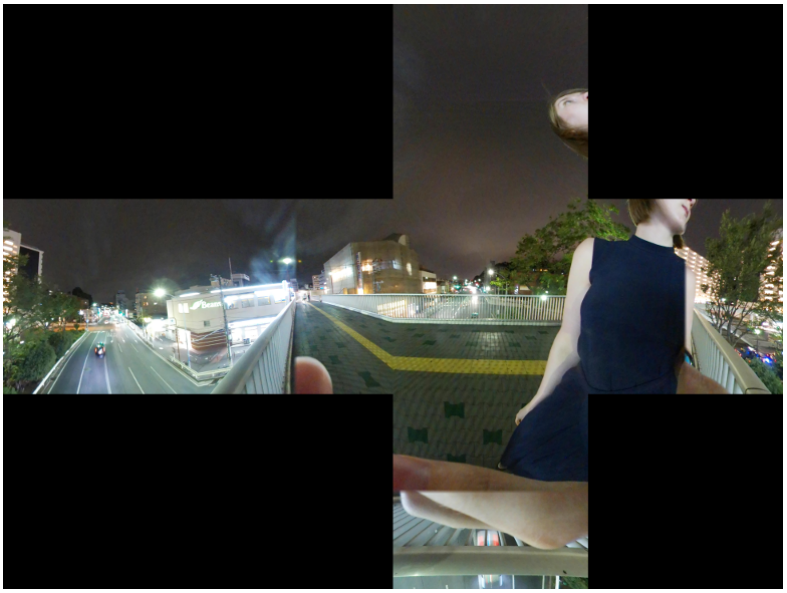


Figure 4: Cube map image created from the images of Figure 2

nature of the mapping. Large redundant areas of the equirectangular image are mapped to much smaller cube faces, which need much less storage space when ordering the cube faces into a three by two grid. Storage is an important aspect when dealing with 360-degree images, as they need large resolutions to show high quality surroundings, so it is necessary to optimise the bandwidth usage as much as possible.

3 Image Stitching

In this section the Image Stitching problem is described. Most of the work described in this Chapter was originally done during a project in Research Seminar in Data Mining course [23]. In most cases 360-degree cameras consist of several lenses and produce several images that need to be stitched together in order to produce the full image. In this work, the Samsung Gear 360 camera was used, it consists of two lenses and natively outputs images consisting of two circular fisheye images. The camera comes with a program that handles stitching. That software has numerous drawbacks:

1. Some parts are stitched together wrong and discontinuities in straight lines can be seen
2. Image borders are visible due to inefficient smoothing
3. The software works only on Windows
4. Slow and inconvenient to use.

For these reasons, an effort was made by the author to come up with a similar algorithm that would produce comparable results. It was expected that the result would not be as good, but it would give a valuable insight into the problems that are faced during that process so it would be possible to develop a better algorithm if it becomes necessary. The smoothing of image borders is not discussed in this thesis, although it is expected to be a much simpler problem and even a simple smoothing technique is expected to give a noticeable improvement. The choice of focusing on the stitching part is motivated by the larger complexity of this problem along with its relative uniqueness that prohibits the use of traditional image stitching algorithms. The problem of image stitching is visualised on Figure 5.

3.1 Spherical to Equirectangular

The surrounding world captured by the camera can be thought of as a sphere that is surrounding the view point. Everything that the camera sees is some point on the inside of that sphere. For that reason, it makes sense to use the spherical coordinate system where each point on the unit sphere can be specified with (θ, ϕ) where θ is the polar angle, and ϕ is the azimuthal angle. The location of each

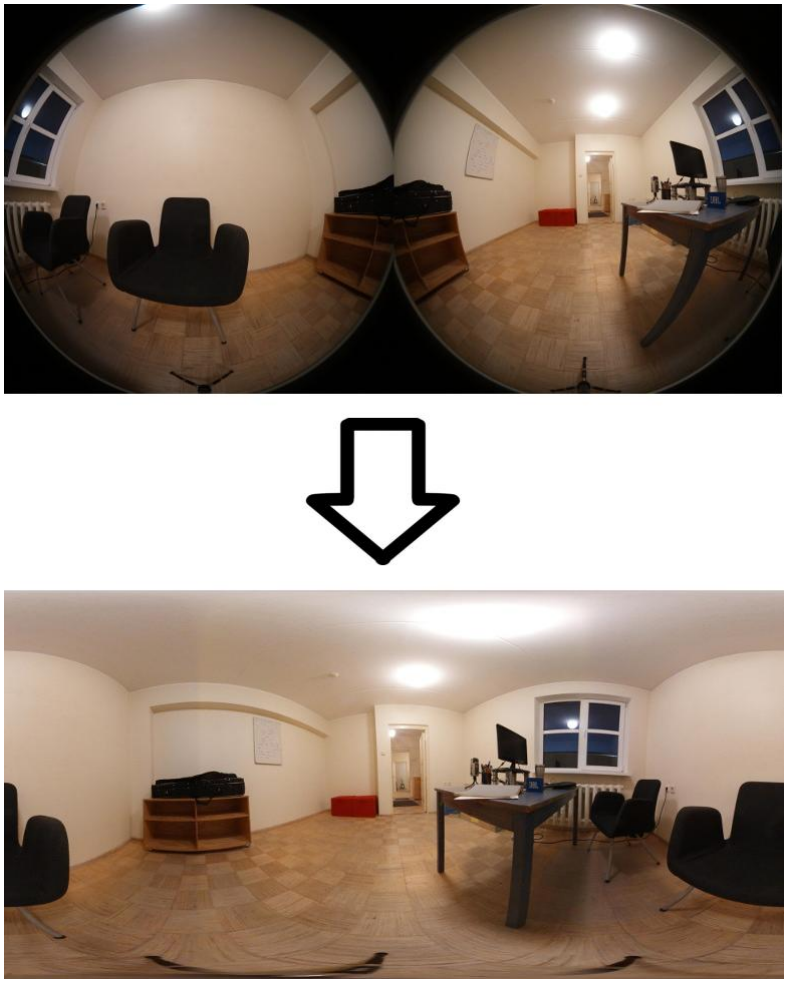


Figure 5: Image stitching task visualised

point on the equirectangular image can be expressed as

$$\begin{aligned} x &= \frac{\phi w}{2\pi} \quad 0 \leq \phi \leq 2\pi \\ y &= \frac{\theta h}{\pi} \quad 0 \leq \theta \leq \pi, \end{aligned} \tag{1}$$

where w and h denote the width and height of the image respectively.

In case of Samsung Gear 360 we have two lenses with view angle of approximately 190 degrees, facing in opposite directions of each other. Both of them can be thought of as half spheres. The half spheres need to be converted to the same spherical coordinate system to map them to an equirectangular image. Each point (x, y) on a spherical image can be converted to a spherical coordinate system by

subtracting the sphere centre coordinates, finding the angle α and length of the radius r.

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$$
$$\alpha = \text{atan2}(y - y_c, x - x_c),$$

where x_c and y_c denote the centre coordinates of the spherical image, and atan2 denotes an arctangent function that returns the result in the correct quadrant based on the signs of its arguments. A spherical coordinate system can be formed using α as the azimuthal angle, polar angle β can be found by rescaling the radius r:

$$\beta = \frac{r}{x_c} \cdot \frac{95\pi}{180}$$

β denotes the polar angle of the spherical coordinate system (95 is half of the view angle of the Samsung Gear 360 lens).

One such spherical coordinate system can be created for both spherical images. The problem, however, is that they are facing in opposite directions, so in one case the spherical polar 0 degree direction shows left, in another case to the right. We need to rotate the points, so that the coordinate system would be the same. We choose a system where the polar 0 points upwards.

First we find the points on the sphere in both coordinate systems as follows:

$$x = \sin(\beta)\cos(\alpha)$$
$$y = \sin(\beta)\sin(\alpha)$$
$$z = \cos(\beta)$$

Now each (x, y, z) point is rotated. For points on the first half sphere:

$$Zr_{-90}\, Yr_{90} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ -z \\ x \end{pmatrix},$$

and for the points on the second half sphere:

$$Zr_{270}\, Yr_{-270} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -y \\ -z \\ -x \end{pmatrix},$$

where Zr_{-90}, Yr_{90}, Zr_{270}, Yr_{-270} denote rotation matrices rotating about Z-axis by -90 degrees, about Y-axis by 90 degrees, about Z-axis by 270 degrees, and

about Y-axis by -270 degrees respectively. Now all the points of two half spheres are in the same coordinate system and can be converted back to the spherical coordinate system:

$$\phi = \arctan\left(\frac{y}{x}\right)$$
$$\theta = \arccos(z)$$

Now that both half spheres have been converted to the same spherical coordinate system, they can be mapped to the equirectangular image using equation 1.

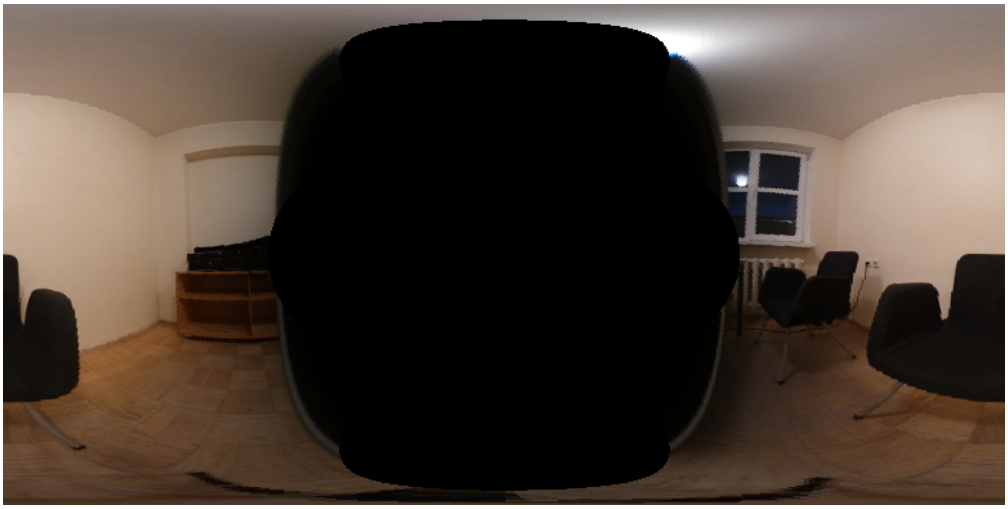


Figure 6: First spherical image mapped to equirectangular image

Figure 7: Second spherical image mapped to equirectangular image

Figure 8: Initial equirectangular image without doing any stitching

As a result of the previous computation, both images can be mapped to the same equirectangular image. This is seen in Figures 6 and 7, corresponding to the mapped two spherical images.

When looking at the two images it is clear that they have a large part in common and it is not clear which image should be used for drawing some particular part of the image. To illustrate that the task is not trivial, an example where exactly 180 degree part was used from both sphere halves is shown in Figure 8.

The result shows clear artefacts. The problem arises from the fact that the mapping between the two spheres is not static. The two cameras behind the two lenses are in fact a few centimetres apart and therefore the mapping depends on the geometry of the room. The stitching procedure needs to be performed.

3.2 Algorithm Description

The image stitching algorithm attempts to create a correspondence between the common parts of the two spherical images. This can be done as both cameras of Samsung Gear 360 have a view angle of approximately 190 degrees, which means that there is a 10 degree part that should be visible from both cameras. That part is where the correspondence needs to be created. The two image strips are shown in Figure 9. For accuracy, it should be noted that both the shown images have additional 5 degree non-overlapping part, totaling 15 degrees, mainly for purposes of matching robustness.

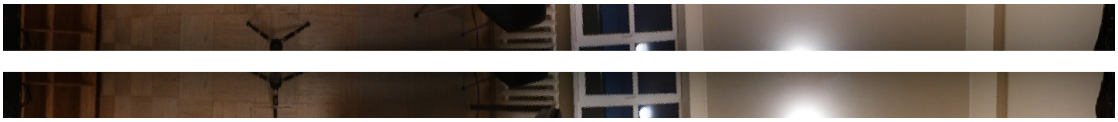


Figure 9: Common parts of the images as seen from two cameras.

The developed algorithm has two parts: 1) vertical alignment; 2) horizontal alignment.

3.2.1 Vertical Alignment

The vertical alignment attempts to detect a correspondence along each column of the image strips. It is done using the y-direction gradients computed using the Sobel operator [24]. A small window of summed up gradient absolute values is maximised along the column on both images, and the position with the maximum value is used as a reference point. The rationale behind it is that the maximum gradient very likely belongs to the same detail on both images, whereas summing up in a window is for robustness purposes.

As there are more uniform areas on images, it is not a good idea to use all the points found in this step, as gradients in uniform areas are probably not very accurate and can be affected by noise a lot. Therefore only strong gradients (larger than some fixed threshold) are used. After finding these points, points of similar height are clustered together and set to a single average height value. This is necessary, as the whole process is very noisy and always seems to contain numerous outliers. After finding the locally similar clusters, however, it is possible to create a quite smooth path showing the approximate vertical touching point between the image strips, see Figure 10. The horizontal line on both images is now used as input for Horizontal Alignment that use Dynamic Time Warping.

Figure 10: Vertical correspondences are highlighted by red dots. Green dots show the separation line computed based on the correspondence points. Narrow green line is computed by linearly extending the found correspondences.

3.2.2 Dynamic Time Warping

Dynamic time warping is a standard dynamic programming algorithm that is in this work used for the purposes of image stitching and later for image sequence matching. A brief overview of this algorithm is given here. A more comprehensive overview of this algorithm can be found in [25].

The dynamic time warping algorithm is used to align two sequences $X = (x_1, x_2, ..., x_N)$ and $Y = (y_1, y_2, ..., y_M)$ in some cost-optimal way. A cost function $f : [1, N] \times [1, M] \rightarrow R$ is defined between these sequences, the algorithm aims to find such a correspondence between the two sequences that the matching sum of costs would be minimal. Let $\mathbf{D} \in Mat_{N,M}$ denote the distance matrix between X and Y, so that $\mathbf{D}_{i,j} = f(x_i, y_j)$. The following pseudocode expresses the dynamic time warping algorithm:

Algorithm 1: Dynamic Time Warping

```
DTW ∈ Mat_{N+1,M+1}
for i ∈ [0, N] do
    for j ∈ [0, M] do
        DTW_{i,j} = ∞
DTW_{0,0} = 0
for i ∈ [1, N] do
    for j ∈ [1, M] do
        DTW_{i,j} = D_{i-1,j-1} + min(DTW_{i-1,j}, DTW_{j-1,i}, DTW_{i-1,j-1})
```

As a result of applying this algorithm, $DTW_{N,M}$ now contains the cost of the optimal match between sequences X and Y. Extracting the match can be done by simply backtracking through the DTW matrix and following the minimum path.

3.2.3 Horizontal Alignment

Horizontal alignment is done after an approximate vertical separation line has been found, as explained in the previous step. Horizontal alignment is done by applying the dynamic time warping algorithm on the horizontal Sobel gradients at the vertical separation line, in positions highlighted on Figure 10 by the green lines. Using a single pixel value of the gradient image was accurate enough for the proof of concept solution, although it is straightforward to generalise the approach

to also use the neighbouring pixels, which should increase the matching accuracy. It should be noted, however, that there might be too few neighbouring pixels if the separation line tends towards the edges, so it needs to be considered how the algorithm should work in such cases.

It should be noted that the time warping algorithm was slightly modified by applying a penalty on cost for non-diagonal movement through the time warping matrix i.e. instead of the conventional expression:

$$DTW_{i,j} = D_{i-1,j-1} + \min(DTW_{i-1,j}, DTW_{i,j-1}, DTW_{i-1,j-i})$$

the following expression is used instead:

$$DTW_{i,j} = D_{i-1,j-1} + \min(DTW_{i-1,j} + \phi, DTW_{i,j-1} + \phi, DTW_{i-1,j-i}).$$

In the author's own subjective view, the results looked best when ϕ was around one hundredth of the length of the warped sequence. That change had quite a large effect, as the overall image quality and mapping quality both increased. Basically it means that deviations from the diagonal movement are done only when there is a really good reason, so slight noise would not have much effect on the computed path, whereas larger gradients are mapped properly.

Figure 11: First image stretched horizontally to match the second with Dynamic Time Warping.

As a result of applying dynamic time warping, the horizontal correspondences are acquired and one of the images is modified accordingly by stretching it to accurately match the second image at positions of strong horizontal gradients. This step is especially important, as failure to correctly align horizontally will be very noticeable in the final image. The result of this operation is shown in Figure 11.

3.2.4 Constructing the Image

The horizontally and vertically aligned image needs to be mapped back onto the spherical images, so that an equirectangular image could be created. One of the spheres is used as a reference and the other one is constructed with stretching. It would obviously be more accurate to modify both spherical images to acquire a

perfect alignment, but in practise, the differences are small and only very slight changes need to be made, thus trying to correct both spherical images would probably result in excessive complications in the algorithm.

Let denote (x, y) denote some image coordinate on the modified image. Let's convert them to polar coordinates (ϕ, r). The angle ϕ. The angle is used to determine the coordinate x_0 on the previously mapped edge image:

$$x_0 = \frac{\phi W}{2\pi},$$

where W denotes the width of the image strip. x_0 can be used to find the location on the original image that should be mapped to (x, y): The $\hat{x}$ component is extracted from the time warped map determining the new "corrected" angle, and the $\hat{y}$ component is extracted according to the height of the so called "touching point" between the edge images (height of the green line on Figure 10). The new angle $\hat{\phi}$ can be expressed as

$$\hat{\phi} = \frac{2\pi\hat{x}}{W},$$

and the new radius $\hat{r}$ can be expressed as

$$\hat{r} = \frac{r}{90}\left(95 - \frac{15\hat{y}}{H}\right),$$

where H denotes the height of the image strip. Finally, the new polar angle $(\hat{\phi}, \hat{r})$ is mapped back to Cartesian coordinate system. This is the location where the new corrected image coordinate (x, y) should be sampled from.

To be more precise, the $\hat{x}$ location should in fact be interpolated according to the dynamic time warping path. However, for large enough image resolutions, the effect of that interpolation is barely visible, therefore it is not described in detail here.

3.3 Results

Using the methods described in the previous sections, an equirectangular image (see Figure 12) can be generated. When looking back to Figure 8, it is clear that the result is much better, as the two parts of the images are properly aligned now. Both the Samsung algorithm and our algorithm produce a clearly visible line at the image borders. In our algorithm this problem was simply ignored, as the focus was on the mapping part, nevertheless it seemed that it would be possible to achieve

Figure 12: Room example stitched using our method

comparable results to the Samsung algorithm using some quite basic smoothing or blending method. When looking at the images by the Samsung algorithm it at looks like they are using some blending method. This problem is most likely quite hard to solve perfectly, as the two cameras look in different directions and might be exposed to different light conditions. It means that the whole image should be modified and it can be expected that barely smoothing the edges will not solve the problem very well.

The described method works quite well, but in its current form it contains several constants that work for images used for testing, but would possibly be sub-optimal for some different sets of images with different resolutions. To really use that method it probably needs some more work to make it robust and usable for a wide range of images. Nevertheless, it seems like the described image stitching problem is possible to solve without some specialised black-box software, at least after some more work is put into it.

Some more results are shown in Figures 13 and 15, whereas Figures 14 and 16 show the same images stitched using the Samsung software for comparison. Note that while the Samsung software produces slightly better results, it also has some serious artefacts that was a motivation for starting this project. In some cases it is due to information being lost between the view angles of two cameras, which might happen when an object is very close to the camera. In some other cases nearby things are simply matched together badly even though they are visible by both cameras. A funny problem is visible on Figures 13 and 14 where one of the

buildings has a different shape on the two images, whereas the author is not sure which image of those two is more correct.

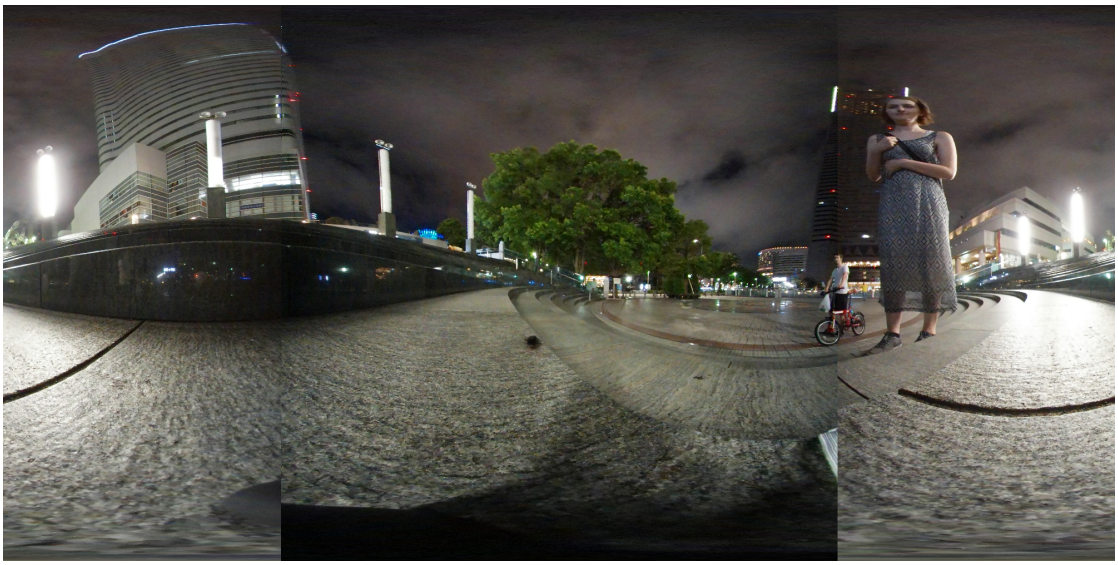


Figure 13: Outdoors example stitched with our method

Figure 14: Outdoors example stitched with Samsung software

Figure 15: Another outdoors example stitched with our method

Figure 16: Another outdoors example stitched with Samsung software

4 Map Generation

In order to create virtual reality environments based on single images, the number of required images becomes large very quickly. To make smooth movement in the virtual environment possible, there should be a sufficiently large number of images captured, otherwise moving from one image to another would create big jumps and look unrealistic and not very similar compared to moving around in the target location in reality. For that reason, it is desirable to have a large image density in the whole area. If one image is taken every 20 centimetres, then even a small room would require several hundred images. Capturing them manually one by one would require a considerable amount of time and effort. In order to automate the problem, it is possible to mount the camera on a remote controlled car. That way, it would become possible to record a video while driving the car around the target area. This approach saves a lot of time, but it becomes unclear, which image corresponds to which location in the room. When taking the images manually, it is always possible to mark down the coordinate of the current location, on a moving vehicle, however, the location needs to be determined in some other way.

4.1 Data Set and Initial Experiments

To start the initial experiments an image data set was manually created by repositioning the camera tripod in different places in the room. The data set consists of 206 images captured in a small room shown in Figure 17, so that there is one image per 20 x 20cm square, each image was labelled with its ground truth location. The ground truth map can be seen in Figure 18. The objective is to generate similar structure purely from the image data. The aim is to learn the same neighbourhood graph, so that each node would have at most 4 neighbours corresponding to left, right, up and down directions.

This chapter starts by conducting some initial experiments using this data set. Afterwards the problem description is modified according to some practical considerations by assuming that the order of images is known. The remaining experiments are done by using video sequences as input instead of single images.

4.2 Similarity Measure

To be able to organise images into a map, there needs to be a metric for comparing two images, as it is reasonable to assume that the most similar images were also captured close to each other in the real world space. For a pair of three-channel

Figure 17: Room that was used for creating the small test dataset

colour images $I_1, I_2 : \{0, 1, ..., W-1\} \times \{0, 1, ..., H-1\} \to [0, 255]^3$ the similarity $s(I_1, I_2)$ of the images can be expressed as

$$s(I_1, I_2) = \sum_{(x,y) \in \{0,1,...,W-1\} \times \{0,1,...,H-1\}} ||I_1(x, y) - I_2(x, y)||,$$

where W and H correspond to the width and the height of the images respectively and $||\mathbf{v}||$ denotes the Euclidean norm of vector $\mathbf{v}$. This similarity measure is used to compute the pairwise differences of all the images to form a distance matrix D that is used by the algorithms described in this section.

In this thesis the similarity measure is used to compare images that have already been stitched together. It is obviously possible to apply that similarity measure on the spherical fisheye images directly. In practise, there is no reason to expect much difference, but it should make the result slightly more accurate when the projection specifics are considered. For example the equirectangular image should be sampled by taking its large redundant areas into account.

To be able to use this similarity measure, all the compared images must obviously have the same orientation, as the metric is obviously not rotation invariant. In this work, the rotation problem has been largely ignored. In our tests the camera was either simply placed using the same angle when capturing the images or the images were manually rotated to have the same orientation.

The image similarity comparison method is one of the possible aspects that can be improved in the future by using some rotation invariant metric. Another idea

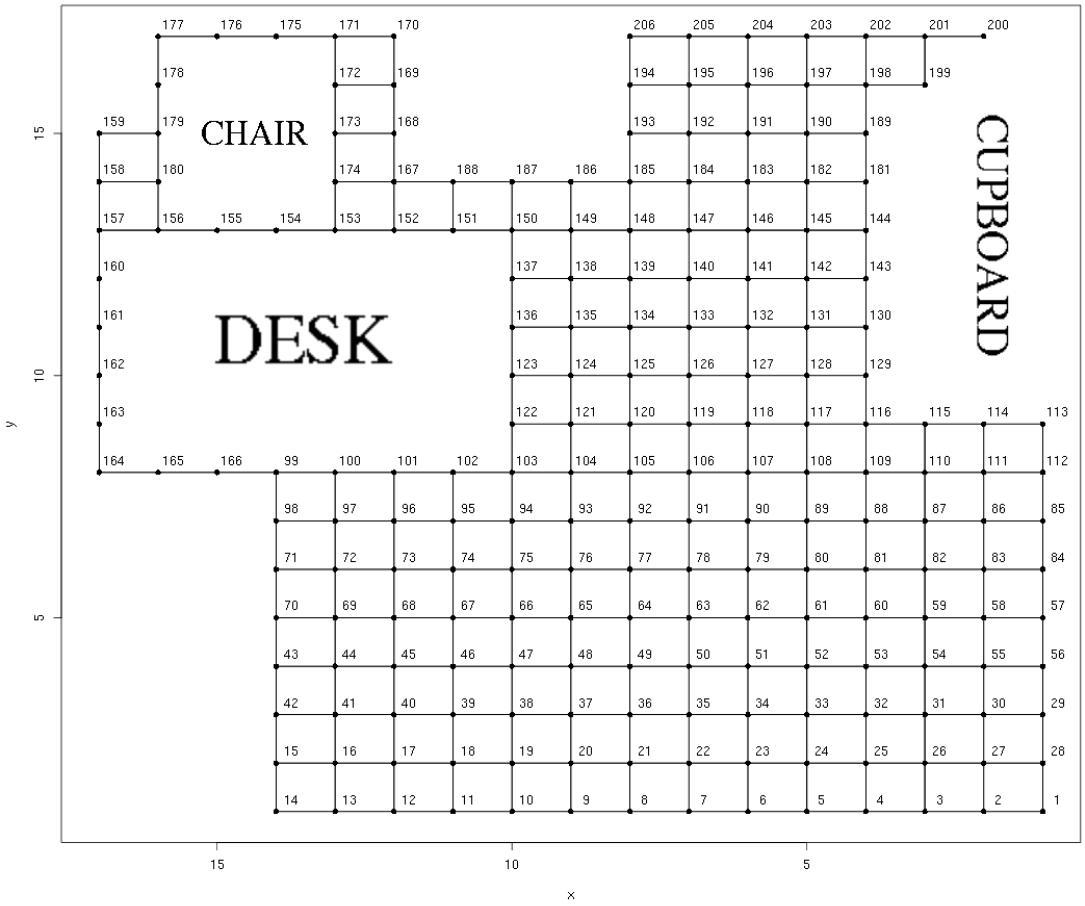


Figure 18: Ground truth map of the dataset

would be to first map the images into some meaningful feature space and later use them for image similarity comparison. As the number of useful features will probably be much smaller compared to the total number of image pixels, it might considerably increase the speed of image comparisons, especially when all the images need to be compared to each other. Obviously the accuracy could also be improved. Choosing a different image comparison method needs additional work and testing, however, and initial results indicate that our simple metric works quite well. Therefore the focus of this thesis is places on other aspects like map generation.

Figure 19 shows a heatmap of the distance matrix created using the sum of error measure. It is easy to notice the grid-like structure in most parts of the image. Figure 20 shows the ground truth distances. Its structural similarity to Figure 19

can easily be seen, indicating that it should be quite possible to form a map based on our distance metric.

4.3 Multidimensional Scaling

Multidimensional Scaling [14] is a well-known dimensionality reduction and data visualisation method that operates on a distance matrix and constructs a lower-dimensional embedding that attempts to keep the distances between the points as close to their original distances as possible. The method is a logical starting point for trying to create a map using the image distance matrix created described in the previous subsection. First, a brief overview of the underlying algorithm is given. After that, some experimental results obtained using that method are shown.

4.3.1 Algorithm Description

The objective of multidimensional scaling is to use an n-dimensional distance matrix $\mathbf{D}$ to find a lower-dimensional representation that would retain roughly the same distance matrix i.e. we aim to find a set of k-dimensional points ($k < n$) with distance matrix $\mathbf{\Delta}$, so that $\mathbf{D} \cong \mathbf{\Delta}$ would hold [14]. It should also be noted that the classical version of the multidimensional scaling used in this work assumes that $\mathbf{D}$ is an Euclidean distance matrix.

Let's denote the matrix of original n-dimensional points as $\mathbf{X}$. Let's also assume that its columns have been centred, so that each variable has a mean equal to zero. This can be done, as translation of a column does not affect the end result. The squared distance matrix $\mathbf{D}^2$ of $\mathbf{X}$ can be expressed as follows:

$$\mathbf{D}^2 = \mathbf{c}\mathbf{1}^T + \mathbf{1}\mathbf{c}^T - 2\mathbf{X}\mathbf{X}^T = \mathbf{c}\mathbf{1}^T + \mathbf{1}\mathbf{c}^T - 2\mathbf{B},$$

where $\mathbf{1}$ is a vector of ones and $\mathbf{c}$ is a vector consisting of diagonal values of $\mathbf{X}\mathbf{X}^T$. After centring the square distance matrix and given that $\mathbf{X}$ was assumed to have columns with zero mean, the following holds:

$$-\frac{1}{2}\mathbf{J}\mathbf{D}^2\mathbf{J} = \mathbf{B} = \mathbf{X}\mathbf{X}^T,$$

where $\mathbf{J} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is the centring matrix. Now eigenvalue decomposition can be used on $\mathbf{B} = \mathbf{X}\mathbf{X}^T$:

$$\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T,$$

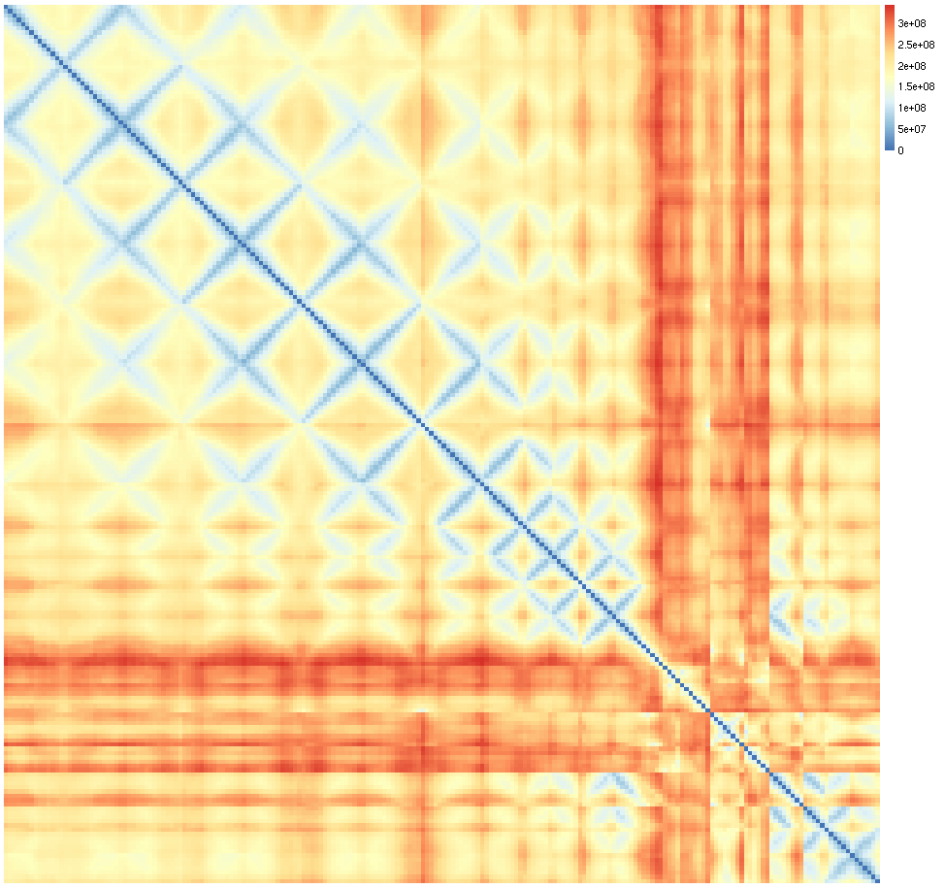


Figure 19: Heatmap of the distance matrix constucted using element-wise sum of differences

where $\mathbf{V}$ is a matrix of eigenvectors and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues. To obtain the final k-dimensional representation, the eigenvectors corresponding to k largest eigenvalues can be used. Let's denote the k-dimensional eigenvalue and eigenvector matrices by $\mathbf{\Lambda}_k$ and $\mathbf{V}_k$ respectively. Then the coordinates can be obtained:

$$\mathbf{V}_k\mathbf{\Lambda}_k\mathbf{V}_k^T = \mathbf{V}_k\mathbf{\Lambda}_k^{\frac{1}{2}}\mathbf{\Lambda}_k^{\frac{1}{2}}\mathbf{V}_k^T = \mathbf{Z}\mathbf{Z}^T,$$

where $\mathbf{Z}$ contains the k-dimensional coordinates.

4.3.2 Results

The distance matrix obtained by computing the distances between all the images in the dataset was used as input to the Multidimensional Scaling method. The

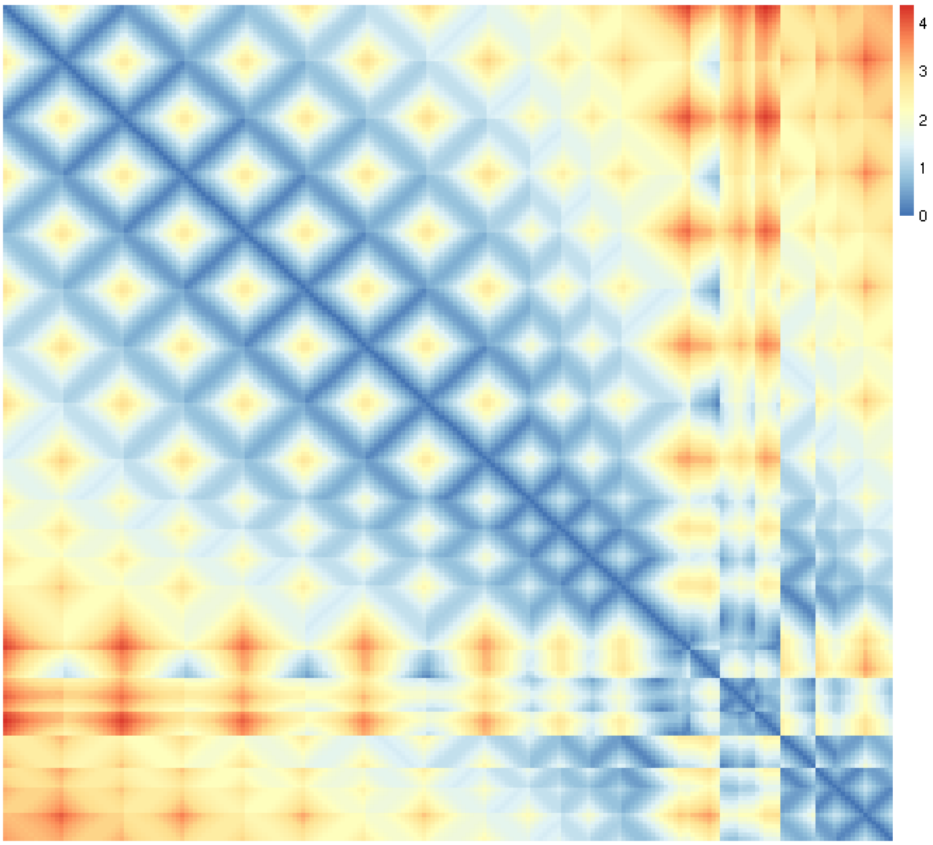


Figure 20: Heatmap of the ground truth distances in metres

resulting 2-dimensional representation can be seen in Figure 21. The results of that method do not look very promising. Although there are some parts that contain grid-like structure, most of the map is simply not recognisable compared to the ground truth map. The most likely reason why this method fails to give useful representation lies in the similarity measure: although it is quite effective at detecting closest neighbours, it does not seem to correspond well to the real life distances. Therefore, it is possible that two objects with a similar pixel-wise distance actually have very different spatial distances. As multidimensional scaling needs the distances to be comparable to produce meaningful results, it means that some improvements need to be made.

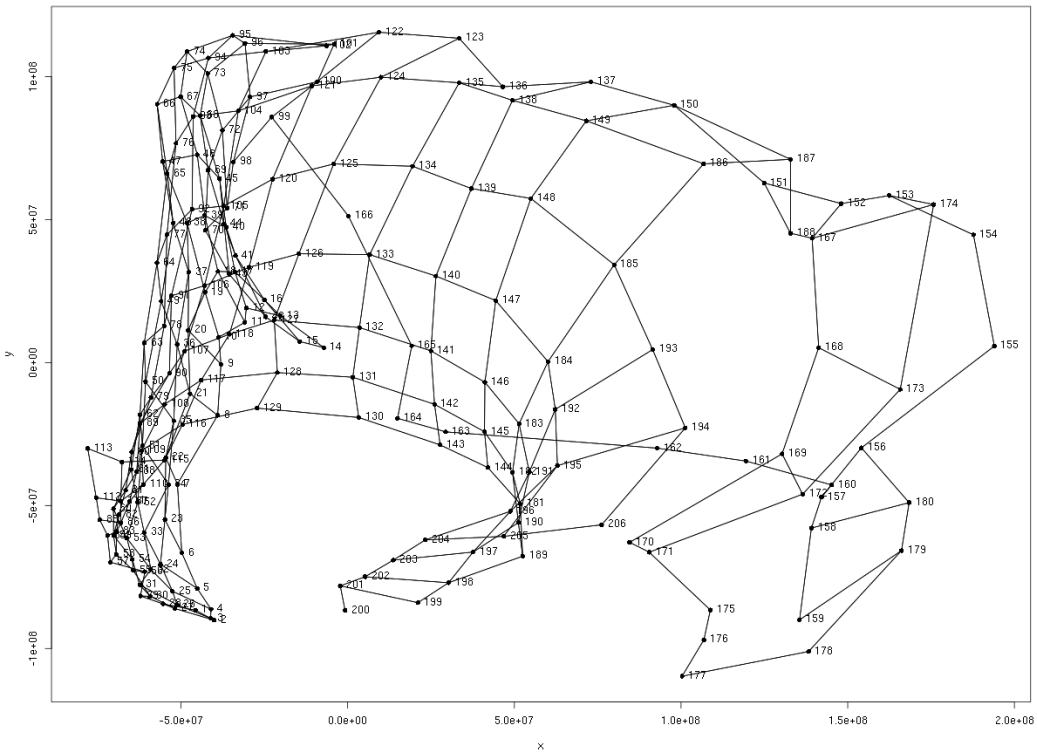


Figure 21: 2-dimensional representation obtained by using Multidimensional Scaling. The lines connecting the points are shown for visualisation purposes and correspond to their ground truth neighbours in the dataset.

4.4 Isomap

As an improvement over the traditional multidimensional scaling method the Isomap algorithm [15] should be tried. The Isomap algorithm works similarly, but operates using approximations of geodesic distances between the points, and uses the multidimensional scaling algorithm to perform the last subtask of the dimensionality reduction. This algorithm is a good choice, as it only requires correct identification of the nearest neighbours, all the remaining distances are determined by computing the shortest distances between the data points. First, a more detailed overview of the Isomap algorithm is given. After that the results are shown and discussed.

4.4.1 Algorithm Description

The Isomap algorithm consists of the following steps:

1. Find nearest neighbours for each point and form a neighbourhood graph;
2. Find the shortest distances between all the nodes in the graph using all-pairs shortest paths algorithm;
3. Find the lower-dimensional embedding by applying the Multidimensional Scaling algorithm on the distance matrix from Step 2.

There are many slightly different ways of implementing the algorithm depending on the use case. For example, there are different ways for finding the nearest neighbours, one option is to use a threshold radius. Another way is to always select the k nearest points. The neighbourhood graph can have weights based on the nearest neighbour distances, but it is also possible to construct a graph with unit edge lengths.

4.4.2 Results

The neighbourhood graph is constructed by finding at most 4 nearest neighbours using the previously computed distance matrix. Additionally, it makes sense to require that all the nearest neighbours are mutual i.e. a pair of points are required to both be one of the nearest neighbours of each other. That way, the neighbourhood graph becomes undirected, which is a natural choice for our task, as distances are symmetric in real life. If some of the points have a non-mutual nearest neighbour then that graph edge is simply removed. The maximum number of neighbours is set to 4 in the hope of obtaining a graph where each point has a neighbour to its left, right, up, and down, as it should encourage the creation of grid-like structures we hope to find in the final map.

After creating the neighbourhood graph, the Floyd-Warshall algorithm [26] is used to find a new distance matrix comprised of the shortest distances between the nodes in the neighbourhood graph. The new matrix can be seen in Figure 22. Almost all of the heatmap consists of grid-like shapes and resembles the heatmap of the ground truth distances shown in Figure 20. Previously the distances were all based on the pixel-wise sum of differences, which was good for finding nearest neighbours, but was not reliable for estimating distances between points more further away from each other. That is also a probable reason why the multidimensional scaling method did not work very well. Now, however, all the distances are based on the

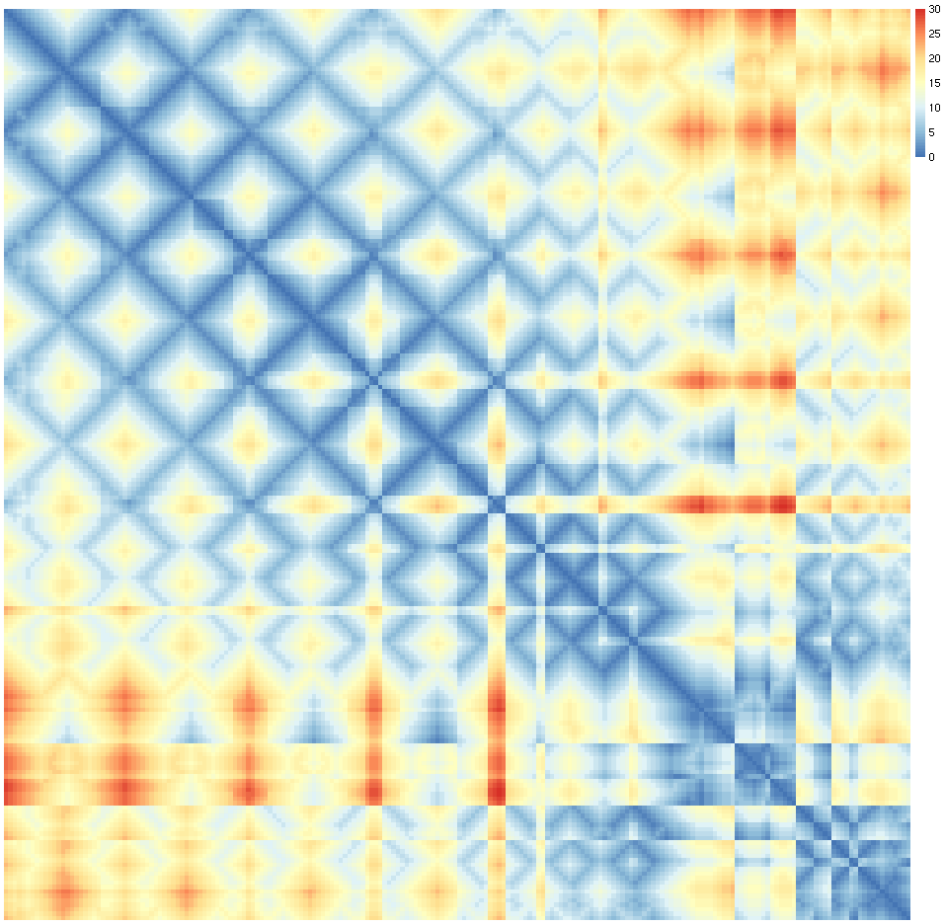


Figure 22: Heatmap of the distance matrix consisting of shortest distances between the nodes of the neighbourhood

detection of the nearest neighbours, so all the results using that matrix should be much more stable and reliable.

The results of applying the Isomap algorithm on the new geodesic distance matrix can be seen in Figure 23. The result is clearly much better compared to the result acquired by using the multidimensional scaling algorithm. All the points are located in a grid with a small gap where the table is located. Most of the points are located in their correct positions and connected to other points in a correct manner. There are some problems due to missing or extra connections caused by wrong nearest neighbour detection, but overall the result is very good and could probably even be used in the final virtual reality creation method. The neighbourhood graph could definitely be used for navigating, as most of the map has quite consistent left, right, up, down directions.

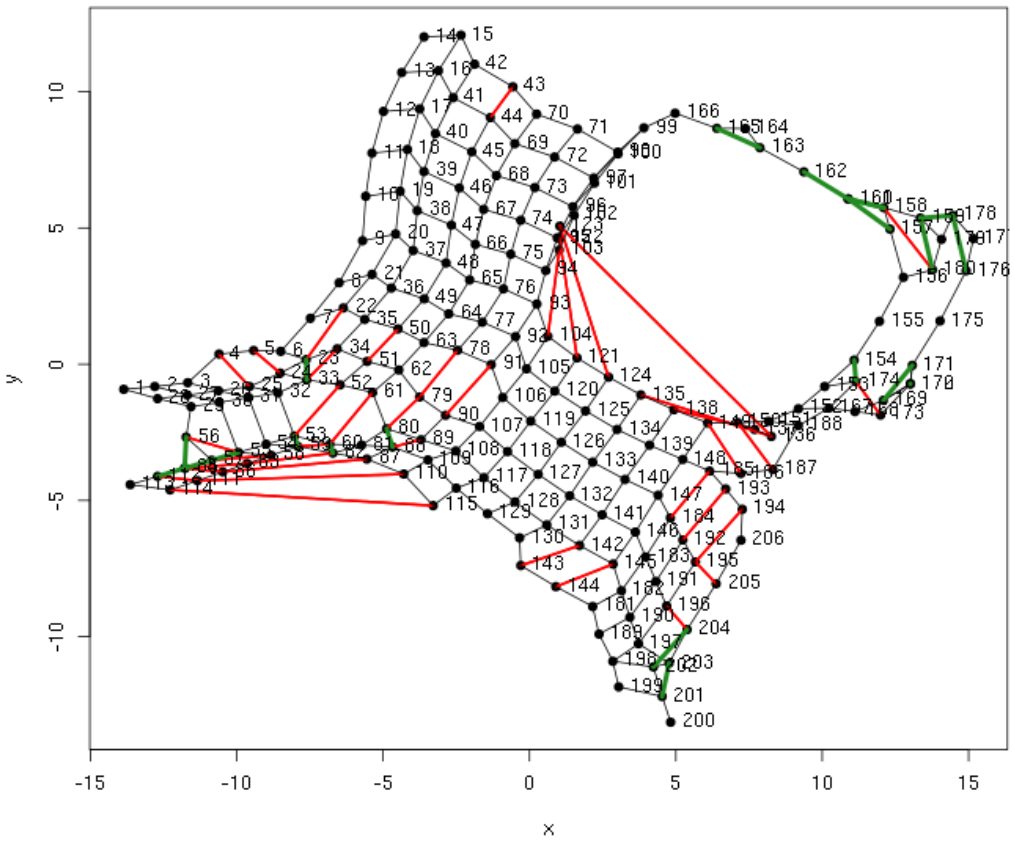


Figure 23: 2-dimensional visualisation created by the Isomap algorithm. Green lines represent connections that do not exist in reality, red lines represent points that should be connected, but are not. Dark lines represent the correct connections.

4.5 Image Sequences

The previously discussed algorithms worked in the general case. They tried to create a 2-dimensional map given only a set of images. Some quite promising initial results were shown and the general case problem seems solvable. In reality, however, there are often some sources of additional information that can be used to make the problem easier. For example, in most real-life cases, it is also known in which order the images were taken. There might also be some information about the general movement pattern of the camera. Such information can be utilised to create a much more robust algorithm.

From now on it is assumed that the input consists of roughly parallel video sequences and the objective is to align them side by side to form the map. To make the method practically useful there can be several groups of such sequences, each group is used to create a part of the map, in the final stage they can be merged together.

The algorithm for creating a map using image sequences, which is described in detail in the next subsections, consists of the following steps:

1. Use dynamic time warping to align the sequences in each group;
2. Create a map from the sequences of each group;
3. Merge the maps together to form the final map.

Before starting to develop this algorithm, the initial idea was to also use the Isomap algorithm by detecting the missing neighbours along one axis and fixing the known ones along the other axis. After some thought, however, it turned out to be a needlessly complicated approach for the new constrained and simplified problem. One of the biggest problems in the Isomap approach is that it is not known how many neighbours each image should have: it might be located on the edge of the map and only have one neighbour along that axis, it might also be part of a long sequence reaching slightly farther than others and would not have any neighbours along the second axis. The problem with the current distance metric is that while it is good for finding closest neighbours, it is not very good for finding how close exactly the neighbour is and it sometimes causes false detections. Of course, there are ways for figuring out how many neighbours an image is expected to have, but the process of figuring that out is almost as complex as the algorithm used for creating the map. For that reason the Isomap algorithm simply does not seem to be very useful for this problem.

4.5.1 Sequence Mapping

The constrained mapping is solved by designing an algorithm that takes a list of image sequences as input, shown in Figure 24 where each blue square denotes an image of the sequence. The aim of the algorithm is to organise them into a map.

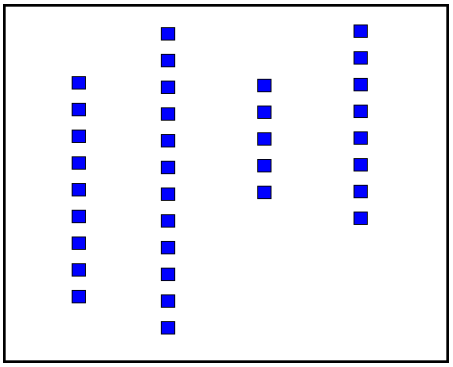


Figure 24: Algorithm input

The algorithm starts by finding all the pair-wise dynamic time warping correspondences between the image sequences using Algorithm 1 shown in Chapter 3. To find a time warping match between the two sequences, their distance matrix is examined and the point with the minimum value in the first row or first column can be used as a starting point for the warping path. The endpoint is chosen in a similar manner by choosing the position with the smallest value in the last row or last column of the distance matrix as the endpoint of the warping path. The reason for these selections is that the first row corresponds to the first image of one sequence, while the first column corresponds to the first image in the second sequence. Similarly, the last row and column correspond to the last images of the first and the second sequence. Choosing the starting and ending points this way enables to find a matching between the overlapping parts of the sequences, as in most cases they do not match exactly from start to end.

It is easy to set custom starting and ending points in dynamic time warping algorithm. The only modification that needs to be done compared to the algorithm described in Chapter 3 is that the dynamic time warping matrix should be initialised at zero in some other coordinate than $(0, 0)$ as shown in Algorithm 1, the warping path can then be constructed based on the chosen starting and ending points after the dynamic time warping matrix has been computed.

After the dynamic time warping step, each pair of images can be easily aligned with respect to each other according to the matching parts determined by the time warping algorithm. In Figure 25, a correspondence between a pair of sequences given as input are visualised. Note that the dynamic time warping can produce many-to-one correspondences, as shown in the previous figure. In practise, it is not a problem, as only a small subset of images is used in creating the final map: there are often too many images in the sequence due to a high video sample rate and, as a side effect, most of the many-to-one connections also disappear.

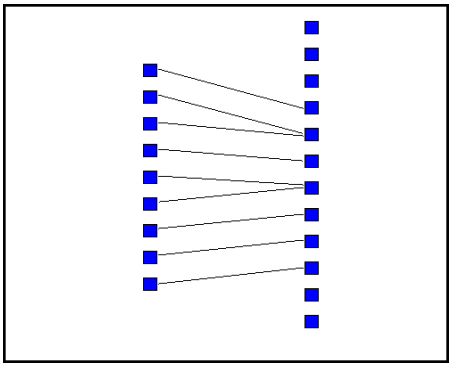
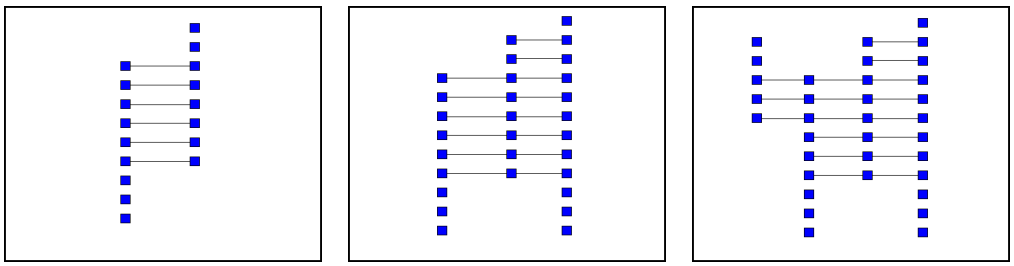


Figure 25: Two sequences matched using dynamic time warping

Using the time warping correspondences, a greedy algorithm can be created for forming the map. The algorithm is initialised by adding one of the sequences on the map as a reference. At each following iteration one of the remaining sequences is chosen and aligned with respect to one of the sequences that has already been placed on the map by using the cost of the time warping match divided by the length of the time warping path as a metric. At each step, the algorithm considers two possible locations: the left border of the map and the right border of the map (see Figure 26). An unassigned sequence that can be added to one of those two places with the smallest cost is chosen and added to the map.



Step 1 Step 2 Step 3

Figure 26: Three steps of the greedy algorithm, at each step one of the sequences is added to the map.

To make this algorithm useful in practise, it needs some improvements. For example, consider the following structure shown in Figure 27. The previous version of the algorithm would not place two sequences on the same X-coordinate, but it is something that is necessary.

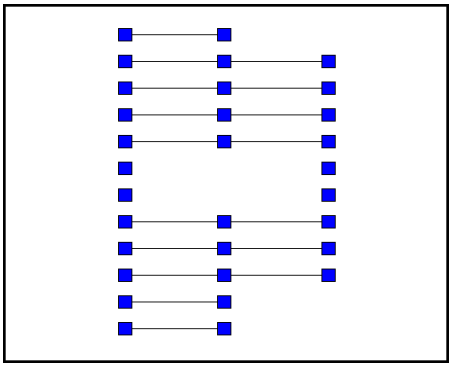


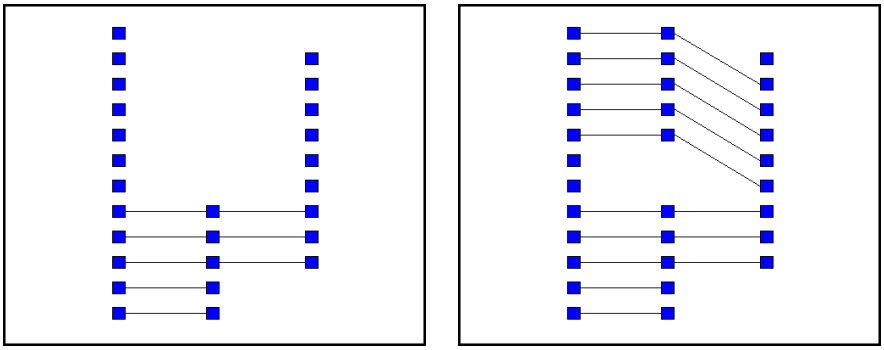
Figure 27: It is important that the algorithm is able to create this kind of structures also.

To make the previously described algorithm able to manage those kinds of situations, it needs slight improvements. Instead of only considering the left and right borders for placing new sequences, it should consider all the free locations next to the already placed sequences. To make the map consistent, it is also necessary to check that the placed sequence would not have any connections in common with the sequences that have been placed on the same X-coordinate.

While this idea seems promising, it turns out that there are some problems. Depending on the order of placement, it may happen that a sequence is placed in a location where it has a neighbouring sequence on either side and it is not so clear which one should be used for alignment purposes. We can hope that both its neighbours would produce the same alignment, but there are certainly cases of suboptimal matches that would cause problems in such scenarios.

Another, even bigger problem is caused by possible movement speed differences in the videos. Dynamic time warping is used to align each sequence, but what happens when parts of the sequences do not have a neighbour? The part that has correspondences can be mapped properly, but the remaining coordinates need to be generated in some other way. This fact will also cause problems in scenarios where a sequence needs to be placed between two already assigned sequences, as the sequences that are not direct neighbours themselves can be expected to not correspond very well. This problem is illustrated on Figure 28.

Perhaps there is some reasonable strategy to deal with such problems, but the algorithm would very likely become much more complicated. An important observation should be made here: the map is used for making navigation possible and whether or not it has 2-dimensional coordinates is less important. Even in the previous example, if we assume that the dynamic time warping correspondences are correct, they can be used for creating an implicit map consisting only of directed



Step $n-1$ Step n

Figure 28: While everything seems correct after step $n-1$, problems occur on step n due to different movement speeds in video sequences.

connections. By using implicit maps the algorithm can stay simple. Theoretically, that kind of implicit map could be used as input for the Isomap algorithm to construct an explicit version of the map, but it is simply not necessary for our application.

Examples of the maps created by the algorithm are shown in Figure 29. These maps are created explicitly for visualisation purposes, and also because they did not contain more complicated structures that were described before. Each of these maps contain around 200 points and can be used for navigating a small room.

4.5.2 Merging Maps Together

It is often more convenient to create smaller parts of the map separately and merge the pieces later. This can be achieved as follows: one of the map parts is used as a basis for creating the full map, all the other parts can then be aligned with respect to that one. Each remaining part of the map to merge into the full map will be determined by finding the closest pair of images between the current full map and the remaining map parts. As we are dealing with implicit maps consisting of directed connections, map alignment is not necessary and adding the required connections is sufficient.

There is also the problem of map rotations: it might happen that different parts of the maps need to be rotated before they can be assigned to the map. Furthermore, the rotation needs to be known for determining the map parts which to merge to each other, as it involves image comparisons. Before that can take place, the images need to be rotated properly, so they could be compared to each other.

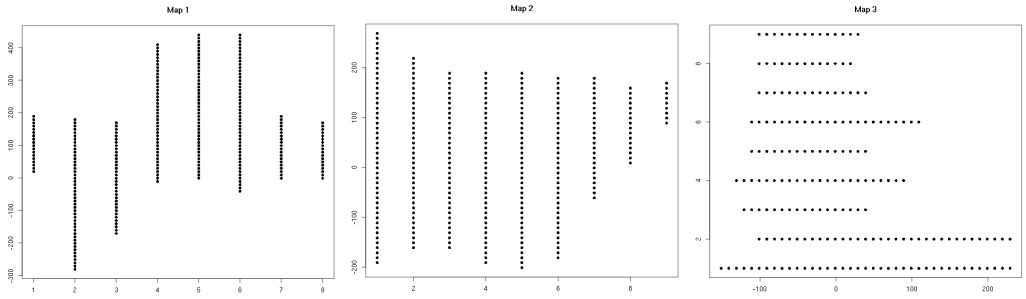


Figure 29: Three different maps created using the described algorithm that should finally be merged together to produce the final full map for the virtual reality environment.

In this thesis, the rotation is determined manually before starting the merging process. It is probably possible to automate this by using some strategy of looking through images and incrementally rotating them, then comparing at each small change, and finding such rotation that minimises the image difference. By using some relevant features, it would be possible to also compare images faster and the problem might even be quite easily solvable. Nevertheless, this needs some more work and experimentation, and is outside the scope of this thesis.

5 Environment Rendering

A 360-degree image can be viewed directly as an ordinary image, but it would lead to a distorted look, as shown on Figure 30. The best way would be to show only a relevant subsection of the image corresponding to some small view angle, as it would be seen from the viewpoint of a person. It would also enable to change the viewing angle and to look in a different direction and create an impression of looking around in that location in reality. These kinds of problems can be solved using the conventional computer graphics pipeline, which is briefly introduced in the next subsection.

The high level idea is to construct a sphere around the viewpoint, and to map the inner side of the sphere with an image acquired from the 360-degree camera, so it would give an impression of the world seen on the image surrounding the viewer from all sides.

The currently visible part of the image would be determined by the viewing angle, which can be controlled by the user. At first, there would be one part of the image visible, after changing the view direction some other part of the image would become visible. In short, it would enable to look around the scene using only a single image by using it as an input to the computer graphics pipeline that would generate many more images exactly suitable for a human observer. Examples of such images are shown in Figure 31.

5.1 Rendering Pipeline

Here a brief overview of the conventional rendering pipeline is given. A more comprehensive overview can be found in [20].

In conventional 3D computer graphics everything is constructed out of triangles consisting of three 3D points i.e. vertices. Given a set of triangles in the 3-dimensional space, the main objective of the rendering process is to project them to 2-dimensional screen coordinates and to colour them in some manner. To determine the projection, there is also a position in the 3-dimensional space that is considered the view point along with a view direction vector.

The projection from the world space to image plane is generally achieved using a series of matrix transformations that are applied to all the 3D points:

1. **Camera Transformation:** Coordinates of all vertices are translated and rotated, so that the view point becomes the origin and the view direction

conveniently becomes parallel to one of the three axis, the new coordinate system is called the camera space.

2. **Projection Transformation:** All vertices visible when looking to the view direction are mapped to a unit cube i.e. the canonical view volume. The matrix that performs this transformation makes use of field of view constants for determining which vertices will be visible.

3. **Viewport Transformation:** The canonical view volume us mapped to the screen space. After applying this transformation all the visible vertices have a corresponding pixel coordinate.

After obtaining the screen coordinates for each visible vertex, the triangles must be rasterised. Rasters are used for computing final values for pixels: each raster corresponds to one pixel, but there might be many rasters corresponding to that one pixel. This is caused by surfaces blocking each other. Logically, the surface nearest to the camera should be visible. To achieve that, the final stage of the rendering pipeline needs to determine which raster should be used for updating the pixel. Nowadays, the most common method is called the z-buffering. It keeps track of fragment depth values and selects the fragments with the closest depth. The pixel colour is updated with the colour of the raster and the image can then be observed.

Of course, the previous discussion did not explain how the raster colours are determined. There are many different methods for that. The simplest case is that each vertex also has a colour value, so that all the raster colours could be interpolated using these vertex colours. In most cases, however, the raster colours are sampled from a texture image. In that case each vertex has a corresponding 2-dimensional texture-domain coordinate that is used for determining its value. All the raster colours can then be sampled from the image. Finally, most rendering systems have some sort of shading scheme, as usually there is a light source that affects the colours of the surfaces according to various factors depending on the method. In fact, our system does not need any shading approach, as it does not need it. For that reason it is not discussed any further in this work.

5.2 Texture Mapping

The 360-degree image must be mapped to a sphere, so it can be rendered using the previously described pipeline. In this work, equirectangular images were used so the descriptions assume that format. However, for other formats, very similar mapping schemes should work.

Figure 30: Image taken using the Samsung Gear 360 camera [22] and converted into the equirectangular format

Texture coordinates, denoted as (u, v), such that $u, v \in [0, 1]$, are defined for each vertex and determine the point on the texture image that is used for colouring that particular vertex. Image coordinates can easily translated into texture coordinates by scaling the image dimensions into $[0, 1] \times [0, 1]$ in such a way that $(0, 0)$ corresponds to lower left corner of the image and $(1, 1)$ corresponds to the upper right corner.

Let us now find the texture coordinates for an equirectangular image given a vertex v. Firstly, we use spherical coordinates (r, θ, ϕ) for expressing the location of v. In our case we have a sphere surrounding the view point, so that the radial distance r does not matter. Therefore, we can simply use a unit sphere with $r = 1$ and the texture coordinates only depend on the polar angle θ and the azimuthal angle ϕ. Assuming $\theta \in [0, 180]$ and $\phi \in [0, 360]$ are expressed in degrees, then the texture coordinates can be expressed as

$$u = \frac{w\phi}{360}; \qquad v = \frac{h\theta}{180},$$

where w and h correspond to the width and the height of the texture image respectively. After assigning the texture coordinates, the environment mapping is ready and can be rendered and observed.

Figure 31: Images rendered from the equirectangular image shown in Figure 30 using different view angles

5.3 Results

After constructing the unit sphere and mapping an equirectangular image texture to it as described in the previous subsection, the scene is rendered with the view point in the centre of the sphere. The described method enables us to use the input of equirectangular images like the one shown in Figure 30, and to produce rendered images like the ones shown in Figure 31. Notice how the curved lines in the equirectangular texture image become straight lines on the rendered images due to perspective projection. The smaller viewing angle also contributes to the more natural and human-friendly look on the generated new images.

This method provides a central component in the virtual reality environment visualisation. As each image in the virtual reality map can be rendered in the same way, it becomes possible to simulate movement in the environment by switching the texture images according to the user input and rendering using the corresponding new texture images, which should create an impression similar to moving around in the environment in reality.

6 Implementation and Experiments

The described map generation system is implemented in several steps: The computations for finding image similarity matrix and creating the neighbourhood graph are implemented in C++ using the OpenCV library [27]. After that some computationally less heavy tasks like applying the multidimensional scaling algorithm are performed in R [28].

The navigator interface that displays the environment and enables to look around and move freely in the environment is written in C++ and uses OpenGL [29] for rendering point of view images based on 360-degree image input. Drawing the images on the screen, event handling and image loading are implemented using the OpenCV library.

The images from the Samsung Gear 360 camera [22] are stitched together to obtain 360-degree images using the Samsung Gear 360 Action Director application [4].

All the code is included to the thesis and can be accessed in the Graduation Theses Registry webpage [30].

6.1 Gathering Data

In our work, the data was captured using a small remote controlled car that had the Samsung Gear 360 camera mounted on top of it, shown in Figure 32. The car was used to capture image sequences of roughly parallel movement pattern as visualised in Figure 33.

After capturing and stitching the video, parallel image sequences need to be extracted, so that they could be used as input for the sequence mapping algorithm. In theory this task can be made automatic using some machine learning approach, but currently it is done manually. A tool was created that plays the video file and lets the user decide the starting and ending points of the image sequences. A user can press 'S' and 'E' keys whenever an image sequence starts and ends, respectively. As a result of this step, there will be a number of video files, each of them corresponding to a sequence used by the mapping algorithm. It is also left for the user to decide which sequences need to be reversed or rotated. In conclusion, this is the step that requires user input the most.

Figure 32: Remote controlled car with a 360-degree camera used for capturing video data for the virtual reality system.

6.2 Workflow

The first step is driving the remote controlled car in a reasonable sequence. In the current state of the algorithm it is important to do it properly, so that the sequences would be parallel. While developing the system, 3 rooms with total size of approximately 48 square metres were used. Driving the car in these rooms took approximately 20 minutes and resulted in approximately 15 minutes of video. The camera captures everything on its SD-card, so copying it on a computer is straightforward. The next step is stitching everything, this is a very slow step that currently takes several hours to complete with the Samsung software for a 15 minute video. Theoretically it could be done with the algorithm described in this thesis also, but the quality would probably be slightly lower, whereas the time consumption would be similar. After the stitching has finished, the data is copied to another computer, the Samsung software works only on Windows and the main algorithm development has been done on a Linux system. Of course, it should be quite easy to get all the algorithms working on the Windows machine as well, as all the used libraries work in Windows as well.

Next, all the video files from the previous step are processed by a program that enables the user to interactively select the video sequences. It asks the user to select the sequence group and the rotation of that group. It also asks whether the sequences should be reversed. After applying the specified rotations and sequence reverse operations the program saves the sequences as separate video files and computes distance matrices between them. Computing distance matrices makes this step is very slow: for the 15-minute video example that was separated into 26 sequences, it takes somewhere around 2 hours to complete. It might be possible

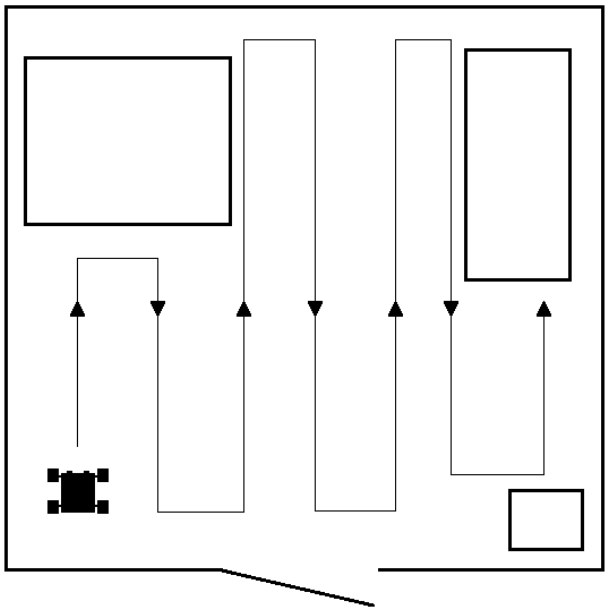


Figure 33: Remote controlled car is driven through the mapped area in approximately parallel sequences that are later used to create the final map for the virtual reality environment.

to speed up this process considerably by resizing images, but its effect on system accuracy needs to be tested first.

After that the R code is used to create parts of the map and to merge them together. While this step is slow, it should take less than an hour to complete in the described example case. The main time consumption is used on doing time warping on each pair of sequences in the same group. This step could be made faster in the future by implementing the whole process in C.

The last step is to run the program that parses the generated map file and extracts the required images from the video sequences as ordinary image files. This step is quite fast and takes less than 5 minutes to complete in the example case. When this program finishes the produced maps can be loaded into the virtual reality navigator. So in total, crating a virtual reality environment from three small rooms takes slightly less than 8 hours for the described example right now. Some parts can be made faster, but the longest step, the stitching, is hard to improve. While 8 hours seems a long time, it should be considered that creating a conventional virtual reality environment would still take much more time and be much less automatic.

Currently, the whole pipeline that starts after image stitching is split into three steps. They could be combined into one program quite easily, but the current design is intentional. The described system is a good first step towards creating image-based virtuality, but it is not a final product yet. Quite the opposite, this thesis serves as just a functional starting point. As most of the substeps need to be studied and improved independently from the whole system, the current modular design makes future development easier.

Conclusions, Future Work and Limitations

Conventional virtual reality environments are largely based on 3D models and creating them is time-consuming. This is a big disadvantage compared to image-based virtual reality, where a single image is able to capture the environment as seen from a single location. In image-based virtual reality the camera is fixed, however, so the movement can only be simulated by creating a dense grid of images captured from as many locations as possible. This thesis investigated possible methods for automatically creating maps for image-based virtual reality environments. A graphical user interface for displaying such environments was also created. Some existing methods were studied and tested, along with a novel method designed based on the specifics of our image capturing system.

The work described in this thesis could be improved in several ways in the future. Currently, the video sequences and generated map parts are manually rotated to have the same orientation, but it should not be too difficult to make that step automatic. Although the map generation was the main focus of this thesis, it too can definitely be improved, perhaps by replacing the greedy matching algorithm with a solver for some kind of an optimisation problem. The sequence extraction method, where the user needs to decide when a sequence starts and ends in an initial video containing several sequences, could also be automated. Doing so might turn out to be quite difficult, however. The most obvious improvement would be to replace the image similarity method with something more sophisticated, as it would enable speeding up the currently very slow pair-wise image comparison, while hopefully also improving accuracy and ideally providing rotation invariance in matching.

Another way to improve the work is to add support for virtual reality headset, as it would make the whole virtual reality experience much more immersive. Allowing the user to look around the virtual reality environment while wearing a head-mounted display should not be very difficult to achieve, as it would simply require to write a user interface to process rotational and positional sensor input from the headset and to use the headset display instead of the computer screen. It remains to be seen if such movement would feel realistic or not.

A big limitation of the current system is the large amount of computational time required and several ways to improve on that were discussed in the last chapter. Another limitation of the described system is that currently the captured images are monoscopic, captured with a single camera. Displaying the same image for both eyes inside a VR headset would not achieve a true 3D depth perception, which will somewhat lessen the virtual reality experience. Stereoscopic 360-degree cameras should mitigate this problem once they become available widely enough.

References

[1] I. E. Sutherland, "The ultimate display," *Multimedia: From Wagner to virtual reality*, 1965.

[2] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*, vol. 26. Springer Science & Business Media, 2012.

[3] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver, "Google street view: Capturing the world at street level," 2010.

[4] Samsung Developers, "Download Gear 360 Action Director." `https://resources.samsungdevelopers.com/Gear_360/02_Download_Gear_360_Action_Director`. [Accessed: 30. Sept. 2016].

[5] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, no. 1, pp. 59–73, 2007.

[6] Y. Xiong and K. Pulli, "Fast panorama stitching for high-quality panoramic images on mobile phones," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 298–306, 2010.

[7] Y. Xiong and K. Pulli, "Sequential image stitching for mobile panoramas," in *IEEE International Conference on Information, Communications and Signal Processing (ICICS)*, 2009.

[8] S. Schleimer and H. Segerman, "Squares that look round: Transforming spherical images," *arXiv preprint arXiv:1605.01396*, 2016.

[9] X. Deng, F. Wu, Y. Wu, and C. Wan, "Automatic spherical panorama generation with two fisheye images," in *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pp. 5955–5959, IEEE, 2008.

[10] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3d," in *ACM transactions on graphics (TOG)*, vol. 25, pp. 835–846, ACM, 2006.

[11] Y. Rubner, C. Tomasi, and L. J. Guibas, "Adaptive color-image embeddings for database navigation," in *Asian Conference on Computer Vision*, pp. 104–111, Springer, 1998.

[12] J. Sivic, B. Kaneva, A. Torralba, S. Avidan, and W. T. Freeman, "Creating and exploring a large photorealistic virtual space," in *Computer Vision and*

Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on, pp. 1–8, IEEE, 2008.

[13] T. Endo, A. Katayama, H. Tamura, M. Hirose, T. Tanikawa, and M. Saito, "Image-based walk-through system for large-scale scenes," *VSMM98*, vol. 1, pp. 269–274, 1998.

[14] A. C. Rencher, *Methods of multivariate analysis*, vol. 492. John Wiley & Sons, 2003.

[15] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[16] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[17] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering.," in *NIPS*, vol. 14, pp. 585–591, 2001.

[18] M. Gashler, D. Ventura, and T. R. Martinez, "Iterative non-linear dimensionality reduction with manifold sculpting.," in *NIPS*, vol. 8, pp. 513–520, 2007.

[19] J. Díaz, J. Petit, and M. Serna, "A survey of graph layout problems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 313–356, 2002.

[20] P. Shirley, M. Ashikhmin, and S. Marschner, *Fundamentals of computer graphics*. CRC Press, 2009.

[21] T. Georgiev, Z. Yu, A. Lumsdaine, and S. Goma, "Lytro camera technology: theory, algorithms, performance analysis," in *Proc. SPIE*, vol. 8667, p. 86671J, 2013.

[22] Samsung, "Samsung Gear 360." `http://samsung.com/global/galaxy/gear-360`. [Accessed: 30. Sept. 2016].

[23] A. Traumann, "Stitching 360-degree spherical images," December 2016. Research Seminar in Data Mining (MTAT.03.277) course in Tartu University, Final Report.

[24] R. C. Gonzalez and R. E. Woods, *Digital Image processing*. Pearson Prentice Hall, 2008.

[25] M. Müller, *Information retrieval for music and motion*, vol. 2. Springer, 2007.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[27] OpenCV , "Open Source Computer Vision Library." `http://opencv.org`. [Accessed: 30. Sept. 2016].

[28] R. C. Team *et al.*, "R: A language and environment for statistical computing," 2013.

[29] OpenGL , "Open Graphics Library." `http://opengl.org`. [Accessed: 30. Sept. 2016].

[30] Tartu University , "Institute of Computer Science Graduation Theses Registry." `https://comserv.cs.ut.ee/ati_thesis`.

Non-exclusive licence to reproduce thesis and make thesis public

I, Andres Traumann (date of birth: 28th of March 1992),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Semi-Automatic Method for Creating Virtual Reality Environments

supervised by Margus Niitsoo and Madis Vasser

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 04.01.2017