Implementing an optical multi-touch technique on a hemispherical display

Thomas Schoegje

Thomas.Schoegje@student.uva.nl
Student number 10068767

August 13, 2013

Supervisor(s): Dr. R.G. Belleman (UvA)
Signed:
Abstract

Multi-touch detection for flat surfaces has been well-covered in the literature, but using a surface with a different shape often introduces new problems for touch detection techniques. This thesis considers the application of different techniques to a hemispherically shaped screen and describes the complete implementation of a system based on Rear Diffused Illumination, an optical multi-touch technique. In this technique a camera is used to recognise touches on a diffuse surface as blobs. Three methods will be introduced for translating the location of these blobs on the camera image to locations on the actual display. These approaches are based on the following principles. The first approach requires the user to touch a lot of known points such that the position of any new blobs can be found by comparing their position to the positions of known points. In the second method the user indicates how the images projected onto the sphere are distorted due to the camera’s point of view, by touching four points. The final approach is based on finding a camera matrix which can be used to describe how points on the camera’s image map to a plane in world coordinates. The line (ray of light) from the camera through this plane intersects with the screen at the location of the touch.

Several experiments will be proposed for comparing these methods as well as rating the adequacy of the system’s touch detection as a whole. These experiments have not yet all been completed due to an as of yet incomplete implementation, but the results so far as well as future work will be discussed. The resulting improvements of this work might be presented near the end of august.
# Contents

1 **Introduction** ......................................................... 7
   1.1 Problem statement .............................................. 7
   1.2 Research objectives .......................................... 8
   1.3 Related work .................................................... 8
   1.4 Approach and thesis overview ................................. 9

2 **Methodology** ....................................................... 11
   2.1 Comparing known optical methods ............................ 11
      2.1.1 Shared premises ........................................... 11
      2.1.2 FTIR ....................................................... 12
      2.1.3 LLP ....................................................... 14
      2.1.4 DI ......................................................... 15
      2.1.5 Optical Imaging .......................................... 17
   2.2 Physical setup .................................................. 18
      2.2.1 Implemented setup ........................................ 18
      2.2.2 The image ................................................. 21
      2.2.3 Microsoft Sphere .......................................... 23

3 **Processing the camera image** .................................. 25
   3.1 Pre-processing .................................................. 25
      3.1.1 Background subtraction .................................. 25
      3.1.2 Filtering the proper blobs .............................. 27
   3.2 Recognising interactions ..................................... 27
      3.2.1 Recognising touches ...................................... 27
      3.2.2 Camera distortion ...................................... 27
      3.2.3 Tracking touches ...................................... 29

4 **Matching blob and finger positions** ......................... 31
   4.1 Coordinate systems .......................................... 32
      4.1.1 Cartesian coordinates ................................... 33
      4.1.2 Spherical coordinates .................................. 33
   4.2 Barycentric approach ....................................... 34
   4.3 Perspective approach ....................................... 35
   4.4 Calibration approach ....................................... 39

5 **Communicating interactions** .................................... 43
   5.1 TUIO overview ............................................... 43
   5.2 TUIO profiles ................................................. 44
      5.2.1 Screen coordinates ...................................... 44
      5.2.2 Cartesian coordinates .................................. 45
      5.2.3 Spherical coordinates .................................. 45
6 Experiments
6.1 Experiments
6.1.1 Physical specifications
6.1.2 Accuracy and consistency
6.1.3 Response time
6.1.4 Update frequency
6.1.5 Robustness
6.1.6 Comparison detection on flat screen
6.2 Results
6.2.1 Perspective approach
6.2.2 Calibration approach

7 In conclusion
7.1 Discussion
7.2 Conclusion

A Derivation mapping coordinate from image to plane
In recent years touchscreens have become much more ubiquitous, and flat touch surfaces have been accepted as an intuitive interface for certain types of media. Based on this observation one could wonder whether touch surfaces of other shapes could likewise offer more intuitive interfaces for particular types of applications. User interaction with touch displays has in fact not only been a topic for research with flat touch screens (such as in [1][2], to name some) but also with other shapes (in particular curved ones, in for instance [3][4] and [5]), concluding for example that that user input is more accurate on more convexly shaped surfaces[6]. When the research potential of a non-flat touch screen is combined with the interest that has been shown in acquiring a spherically shaped touch screen for public display on the UvA Science Park campus, it seems that there may be a real use for such a system - and because of the high cost associated with such commercial products this system would ideally be a custom implementation.

Using a non-flat screen introduces additional complexity to the implementation of the system, and although the implementation of flat touch tables is well-covered in the literature there is not as much information available addressing the problems introduced by using a differently shaped screen. In this thesis several solutions will be discussed to solve the problems that are specific to implementing a spherically shaped touch display. The solutions proposed have partially been implemented and their correctness has been tested as far as possible. The remaining work has been outlined and the results of a complete implementation might be presented near the end of August. The performance of the touch-detection will finally be compared to a popular technique used for detection on flat surfaces, which will serve as a point of reference for other touch techniques.

1.1 Problem statement

The goal of the project is to build a system with a hemispherical display which properly recognises where and when the screen is being touched, and which offers touch-based applications an interface to this information. The resulting system should preferably be inexpensive to build and suitable for either future research or public display. Whether the system has 'proper' touch detection is measured by the following metrics:

- the system needs to accurately detect the location of the touch
- it needs to detect touches on the entire surface consistently accurate
- the delay between the moment a user touches the screen and an application responds needs to be as small as possible (low response time)
- the system needs to be fairly robust against environmental conditions such as changes in lighting or the system being moved

Implementing this system can roughly be broken down into three parts. First it must be decided how touches on the surface will be recognised. This dictates the physical setup required
and what kind of input will be available from which touches need to be recognised. Implementing the software to interpret this input is the second step, and finally the application should offer an interface to touch-based applications. The methodology of the touch detection is decided by reviewing the existing literature for both spherical and flat touch screens, and choosing the method that translates best to a spherical surface. After this has been decided the focus of the project will be to compare different methods of interpreting the input obtained from this method. The performance of the techniques will finally also be compared to one used in flat surfaces.

1.2 Research objectives

In order to build the touch display it is important to first consider the following questions.

- How well do methods for flat surfaces translate to a spherically shaped surface?
- What makes touch detection on a spherically shaped surface different?
- How can these differences be dealt with?
- How do these solutions perform compared to a technique commonly used for flat surfaces?

1.3 Related work

Though the use of touch screens was popularised by the releases of the iPhone in 2007 and the Microsoft PixelSense (formerly known as the Microsoft Surface) in 2008, there have been implementations of flat touch displays since the 1960s[7]. This early touch screen used a capacitive method to detect user input - a principle still commonly used in consumer devices. In this technique a small current of electricity runs across the screen. When a capacitant object (an object which can store an electrical charge, such as a person’s skin) touches the screen this current is slightly manipulated, and by detecting where this occurs the location of the touch can be detected. This means touches of non-capacitant objects such as fingernails can not be detected.

If a consumer device does react to non-capacitant objects it is likely using a resistive touch-screen. Such screens are built by having two thin layers next to each other. Whenever something is then placed against the screen it will push these layers against each other, and the point of contact can be localised from the electronic circuit that is closed[8]. This method can require the user to press down harder on the screen than alternative methods do and the reaction time of resistive screens can feel slower than that of capacitative touch screens.

The most popular for Do-It-Yourself implementations however are a collection of optical techniques (sometimes called infrared techniques) which rely on pointing a camera to a surface, getting touches to show up as blobs on the image and then using image processing to determine how an object interacted with the screen. The popularity of these methods is thanks to the relatively low cost of the materials and ease of implementation (which is achieved partly thanks to open source software). These methods include Frustrated Total Internal Reflection and (Front and Rear) Diffused Illumination. The main difference between optical techniques are in how they approach their goal of having only touches show up on the camera’s image. Usually an infrared camera and sources of infrared light are involved. Several approaches and problems of these optical techniques have been summarised in the openly available book Multitouch Technologies written by people from the NUI (Natural User Interface) group[9]. A complete implementation of a flat touch table using Rear Diffused Illumination has also been described by UvA alumnus Laurence Muller in his master’s thesis[10].

The optical technique Rear Diffused Illumination has already been used in the past for creating a spherical touch display, namely for a system called the Microsoft Sphere[11]. Though the article about this system does not elaborate on the image processing or the calibration required, it does share its insights on how a certain physical setup allows for the technique to be used with a spherical display.
Although these are the most popular techniques they are not the only ones applied in practice. Surface Acoustic Wave technology for example sends ultrasonic waves over the touch surface, and a portion of this wave is absorbed when the screen is touched. The change in the ultrasonic waves registers the position of the interaction. The final method that will be mentioned is Dispersive Signal Technology. This approach uses the fact that a touch onto the surface causes ripples of mechanical energy throughout the material and uses (what are called piezoelectric) sensors in order to convert this mechanical stress at the edges of the material into electrical signals. These signals can then be analysed to find out where the touch occurred. As more obscure methods for touch detection are considered it becomes likelier that they lacked any strong advantages which could have lead it to popular adoption, hence no further techniques will be considered.

The literature generally does not seem as concerned with covering the actual implementation of spherical or curved touchsurfaces as it is with the effects these shapes have on user interaction.

1.4 Approach and thesis overview

The first step is to choose (an optical) method of detecting touches by considering which known methods could be adapted for use with a spherical screen. This choice, as well as the physical setup dictated by it, are discussed in chapter 2. Because an optical method is chosen, the setup will produce a camera image which needs to be interpreted, and the preparing of this image before it is interpreted is done in chapter 3. Using the open source software Community Core Vision this camera image will first be pre-processed such that the resulting image consists of white blobs on a black background, representing touches on the surface. The detected locations of these blobs on the camera image are not entirely accurate as the camera lens (like most lenses) introduces significant distortions into the image. These distortions can be compensated for by estimating the parameters of Brown’s camera model, which models the way the camera sees the world. In order to do this functionality of the open source software OpenCV will be used. Several snapshots of a known checkerboard pattern will be taken and analysed to find out how the lens distorted the pattern.

Now that this input has been pre-processed, the blobs’ coordinates on the screen have to be translated to locations on the hemisphere. After discussing how locations on a sphere can be represented chapter 4 will introduce three methods to perform this translation. The first has the user touch a lot of projected points during a calibration process, and uses these points as a reference for determining the location of any new unknown blobs. For the second method the user is asked to touch four points, and these will be used to correct the distortion on the image caused by the perspectives of the camera and the projector. After this compensation the blobs detected are on a known plane, and after measuring the position of the camera it is possible to find the line that connects these two points. By finding out where this line intersects with the sphere’s surface the original location of a touch can be found. In the final method requires the user to calibrate the system using a flat touch screen that is located on a plane that intersects the (hemi)sphere through its centre point. By translating positions on the camera’s image to points on the plane it is again possible to find the location of the touch using an intersection of a line and a sphere. With the position of the touches now known, chapter 5 discusses how to provide an interface to applications using the TUIO protocol.

With the implementation described it is time to turn to the evaluation of the system. Chapter 6 first compares the three methods that were proposed to interpret the system’s input before turning to the system in its entirety. Finally, chapter 7 reflects on the success of the project and considers possible future work.
There are several approaches to building an interactive surface. In this chapter some known methods for recognising touches on flat surfaces will be explored first, and it will then be considered how suitable these would be for a surface that is not flat. After settling on a methodology it will be considered how to implement this on a spherical surface, and the physical setup used for the experiments will be described.

2.1 Comparing known optical methods

There is a range of explored ways to recognise touches on a surface. Common methods for touch detection as mentioned in the last chapter were based on either pressure, the capacitativity of human skin, sound or optics using a camera (these are referred to as optical techniques). When just considering the hardware requirements alone however it quickly becomes apparent that not all approaches are suitable for building a spherical display. Finding either spherically shaped screens or flexible screens that would neatly fit the shape of a sphere is challenging and expensive when these screens also need to have the resistive or capacitative touch-detection built into them. Sending ultrasonic waves that bend along with the surface may also prove difficult. An implementation of the method that detects how the pressure of touches propagates through the material (using sensors at the edge) may be possible, but the shape increases the complexity of the signal analysis.

This leaves the optical techniques. Among these techniques are the most popular do-it-yourself solutions used for flat touch tables, as they are fairly easy to implement as well as cost-efficient. Aside from the (spherical) surface used these methods often do not require any materials that are hard to come by either, and optical methods do not require recalibration as long as the camera does not move and the environmental conditions (in particular the lighting) remain fairly stable. Considering these advantages as well as how Microsoft’s Sphere has already proven that optical techniques are applicable for spherical screens, further research will be delimited to the common optical multi-touch techniques.

2.1.1 Shared premises

All of methods that will follow share the same basic premises. By pointing a camera at the surface they want to notice new touches on the surface as blobs on the camera’s image. By extracting the relevant blobs from the image and tracking these over time it can be determined how the user interacted with the screen, after which it is only a matter of determining how a blob’s position on the image translates to the touch’ actual position on the screen. How exactly touches on the surface are translated to blobs on the image, and how the proper blobs subsequently are extracted from the image depends on the technique used.

Aside from just detecting touches a touch-display should also present information to the user on the same surface, in a way that is compatible with the touch-detection. Because the camera generally needs to see touches from the opposite side of the surface, these optical methods are
usually incompatible with using some kind of opaque monitor as a display. Instead images are often be displayed by projecting images onto a diffused surface (or a clear surface with a diffuse layer). By having the camera see images that are projected unto the screen it becomes difficult to recognise which blobs on the camera’s image are actually touches. This is resolved by detecting touches in the infra-red spectrum of light, and leaving the visible light for the projection of images. Detecting invisible wavelengths of light using an infrared camera makes it is easier to isolate the blobs from touch-detection from more interfering signals than just the projected image, without even having to incur some computational overhead. It should be noted that in the case of flat screens a (much less expensive) LCD monitor can be used as an alternative for a projector, if the plastic casing as well as the Infrared light blocking diffuser layers are discarded. These are not however very suited for different projections than the one used in the screen itself, and will not be considered any further.

The material used for the surface can usually vary from something like acrylic to glass to plastic, though some techniques require a material with specific properties. Regardless of the material used, it is important to have any diffusion be as colour-neutral as possible in order to avoid having strangely colour-distorted images on the surface. Usually a white/grey-ish colour is used.

In the case of a diffuse layer instead of an entirely diffuse material, it is important to note that there is a difference for the user whether this diffuse layer is placed on the underside of the surface (nearer to the camera) or whether it is placed on the topside (nearer to the user). This influences the distance between the projected image and the user’s finger, which in turn has an impact on how the user experiences touching the screen. If this distance is small (when the diffuse layer is on the outside) and the user presses on a point on the projected screen, it is clear from any perspective what point she is pressing. If this distance increases however (because the layer is on the other side of the material), a change in the user’s perspective means that she can see her finger’s position change with respect to the projected image. In this case she may be touching a button while, from her perspective, her finger is not put down on top of the button at all. This may not be very noticeable in practice, as with a decently sized screen the user’s perspective is favourable and the surface material may also be thin enough. It seems however that putting the diffuse layer on the outside is preferable for accuracy in the touch detection (though the layer’s integrity degrades more quickly that way).

This concludes what the methods share in common, and it is time to look at the specific methods and their (dis)advantages.

2.1.2 FTIR

Frustrated Total Internal Reflection (FTIR) is the name given to the multi-touch methodology as first described by Jefferson Han[12]. It is named after the main underlying physical phenomenon which is represented in Figure 2.1. When a wave (such as a ray of light) reaches a transition of one material to another and the refractive index of the second material is lower, the ray will be partially reflected and partially continue on a diverted path. This is represented by the red ray in the image. When the angle of incidence gets greater, there comes a point where the ray would be diverted such that it would continue along the boundary of the material. When the angle of incidence is greater than this critical angle the ray is not able to cross the boundary and instead is reflected back in its entirety, such as the blue ray. This is called total internal reflection.

![Figure 2.1: Rays (of light) at a transition of materials where material \( n_2 \) has the lower refractive index. \( \theta \) represents the angle of incidence. Because it is so large for the blue ray, total internal reflection occurs. The red ray’s angle is too low.](image-url)
Figure 2.2: Infrared light is mostly trapped through total internal reflection until a new material with a higher refractive index comes into contact with the surface and allows rays to become ‘frustrated’ and leave.

Now called frustrated. As the image shows, this can be used to find where the surface touches other materials using an infrared camera. By looking at blobs on the camera’s image (which looks like Figure 2.3) it can then be determined how the user interacted with the surface.

This method has some important disadvantages. First of all, the infrared light will not stay trapped in the material forever; the light will bleed out. A ray of light that has bounced around in the material for a while will eventually reach a boundary at an angle of incidence small enough to escape. This manifests itself as static on the camera-image, mostly because of rays that escape through the underside of the material towards the camera. Some rays that escape outwards may also be bounced back through the material onto the camera by objects above the surface, especially if this object is as closely above the surface as the rest of the hand is in the previous image. The longer the light travels and ‘bounces around’ inside the material, the more likely it is to escape (it has more attempts to do so), and therefore this static is not uniformly distributed. Actual touches however show up quite saliently however, and image processing can compensate for both the static and the faint images of hovering objects.

The second set of problems is choosing the proper materials. The surface needs a refractive index that is low enough, and especially when building a spherical touch-screen it may restrict the choice of materials. Finding one of a good shape is hard enough, but coupled with requirements on the materials there may only be few and expensive options. Using this methodology also means the types of materials which can be used to interact with the screen are limited, which could be a problem for types of applications. When designing objects for interaction with the surface for example (such as a pointer) the materials that could be used are limited.

Finally and most problematic, in a spherical surface it is not as easy to trap the infrared light. The curving surface means the light would have to be refracted at much more angles than a flat surface would demand (see Figure 2.4), which would bleed out along the way giving an non-uniform distribution of light on the image. Given the high refractive index it would also remain a question how much light would be able to escape at a touch on the surface. Whether there is a viable material to solve these problems would require more research. Especially if light is only sent into the surface such as in the previous image the amount of light that reaches the top will be little.

As there is still a tiny bit of room between a finger (or other materials) and the screen even if one presses it against the surface, often a ‘compliant layer’ (typically a layer of silicon) is used in order to increase the coupling of skin with the acrylic.
There is an alternative methodology similar to FTIR called *Diffused Surface Illumination* in which you use a special acrylic surface-material (notably of the brand Endlighten) for distributing the IR light evenly across the surface. This material has small particles in it acting as a lot of small mirrors that spread out the infrared light that is shone into it. Perhaps this would be more applicable for a sphere, but finding a spherical surface may prove hard - especially because such self-diffusing material is patented[13].

### 2.1.3 LLP

The *Laser Light Plane* technique involves shining a plane of infrared light as closely as possible over the touch-surface using lasers (see Figure 2.5). When an object then touches the surface, light from the plane will be reflected and found in the camera. This means a very smooth surface is required. Shooting planes of light is inherently not very well-suited for a spherical surface though. The use of lasers also makes the setup more dangerous to the eyes of the users, which could be damaged if the lasers somehow shone into them.

![Figure 2.5: In LLP a plane of infrared light moves right over the surface, and any touches of the surface first cross this invisible plane and show up on the camera.](image)
LED-LP is a variation on this method where narrow angled LEDs replace the lasers as can be seen in Figure 2.6. This method is less accurate as more irrelevant things (such as the hovering part of a hand) will reflect just as much light as the actual touches. The distribution of infrared light is not uniform here either, as the camera’s image near the lightsources will be brighter than at other parts. This method is still not suited for a spherical surface as the lightsources cannot illuminate the entire screen. When the idea of shining infrared light onto a surface is taken a step further however, the result is the next method for building touch-screens that will be discussed.

![Figure 2.6: In LED-LP there is not really a plane, but narrow angle LEDs cover the surface.](image)

2.1.4 DI

In Diffused Illumination (DI) techniques one shines infrared light on the entire touch-surface which has a diffusing layer. When something gets close to the screen it will divert the infrared rays seen on the camera’s image. There are two main flavours of diffused illumination which will briefly be discussed.

In Front DI there is a source of infrared light (such as infrared LEDs) in front of both the camera and the surface (see Figure 2.7a). This means that when a user puts his hand on the surface some rays will be blocked, and the hand will show up on the camera’s image as a shadow (see Figure 2.7b). Multiple sources of infrared light are adviseable so that items closer to the diffuse layer will have a darker shadow than items further away. One then only has to look at blobs that are dark enough.

![Figure 2.7: Front Diffused Illumination](image)

(a) Shine infrared on the surface and notice when rays are missing on the camera.

(b) In FDI it is the absence of infrared rays that that can be noticed on the camera’s image.
This technique is more suited for a sphere than the previous one (LED-LT) as one could place several sources of light around it. The spread of light would still be uneven on the sphere and depend on where you put the light sources. There is a lack of portability with mounted sources of light and this method cannot afford having people or things stand in front of light sources.

Some of these problems can be solved by looking at Rear DI (RDI), in which the sources of infrared light are behind both the camera and surface (see Figure 2.8a). Rays are sent out of the surface, and when something touches the screen the rays for that area are reflected back into the camera as bright blobs (see Figure 2.8b). The diffusing layer makes sure that bundles of light passing the surface will spread out more at the surface, meaning that objects closer to the surface reflect a more concentrated bundle of infrared light back at the camera. This helps filter hovering images and noise. For a spherical surface RDI means that the light is sent out from the inside of the sphere and that the screen is looked at from the inside of the sphere.

Also worth mentioning is how RDI is applied in Microsoft’s PixelSense technology, shown in Figure 2.9. In this setup infrared lighting comes from a panel that uniformly lights the surface above it, and any touches reflect back infrared light onto optical sensors integrated in a layer below it. These changes notably the optical sensors allow for the surface to be much flatter than when a regular camera is used.

One additional advantage of RDI is that the full view of items placed onto the surface allows for the detection of known objects, which is something many of its competing methods (such as LLP) cannot say. By having several objects that are known to the sphere new methods of interaction become possible. Commands could be programmed for items that can be recognised and of which the orientation can be determined. These are called fiducial markers. There are several applications for such markers. A touch display is for example not ideal for giving numerical inputs, and it would be possible to remedy this by having a selection wheel show up when a known marker is placed and allowing the rotation of the marker to change the selected number that is to be the input. Another example would be for a Google Earth-like application in which the user can navigate the world by dragging and pinching the screen. By taking up all the intuitive interactions simply for navigating through the application a barrier is erected from actually interacting with the world. A marker could be placed on top of the screen to lock the rotation along one axis, allowing for different interactions through the same basic set of interactions. The world could then be rotated by rotating the marker instead. It should be noted that in properly detecting these markers from the camera’s image a distortion because of the spherical shape should first be dealt with.

So far RDI is the only technique discussed for use with a spherical display without severe objections, but it still introduces some hurdles to overcome. The light shone on the inside of the sphere needs to be spread out uniformly over the curved surface for example, and this can get especially difficult when the light starts reflecting. In order to stop some of the reflection the
diffuse layer should be applied on the *inside* of the surface - which as discussed means that some user experience is risked if the screen is not thin enough.

2.1.5 Optical Imaging

*Optical imaging* is somewhat different from the previous techniques in that it usually does not position the camera below the screen. By sending infrared light over the surface of the screen from three directions and having cameras look at a plane just above the screen from the fourth side (see Figure 2.10a) any new touch will show up as a shadow or darker blob on each of the cameras their images. Figure 2.10b illustrates the view several cameras essentially have after preprocessing the image to find where the blobs are located: a sequence of values in which there either is a shadow at this point on the surface or there is not. After finding out which blobs on one image correspond to blobs on the second image it becomes possible to determine the position of the blob using triangulation and epipolar geometry\[14\]. Implementations can be done with two cameras (usually positioned at the edges of the fourth side) or more, though implementations with perspectives from only two points have only been able to properly track just two points at a time.

There are also similar techniques such as the patented *Planar Scatter Detection*, where infrared light is sent into the acrylic surface from two sides as it is with FTIR and where the optical sensors on the other sides look for disturbances in the infrared light\[15\]. Though the techniques get their optical views differently they both require similar signal processing of the one-dimensional signals illustrated in Figure 2.10b. As the blobs are found in only one dimension it is easy (more so than with RDI) to have situations where two blobs passing each other’s paths on one of the camera’s images and consequently confuse which blob continues on which path. This means stronger (and more computationally expensive) tracking techniques need to be employed. In the past an extended kalman filter has successfully been used\[16\]. Later in the thesis tracking techniques and their (dis)advantages will be considered more extensively.

Both the shooting of a plane over the surface of a sphere and using total internal reflection with one have already been discussed and it seems this technique may also not be suited for a sphere’s needs, but the approach of using several calibrated camera’s for identifying the position of a blob in an additional dimension will be worth considering for use with the RDI setup.
2.2 Physical setup

The different (optical) approaches for detecting touches on flat screens often turned out to translate poorly to systems with spherical screens. It even turned out that just one of them (Rear Diffused Illumination) has enough potential to warrant further investigation.

The next step in the implementation is to find a way to make this technique work with a spherical surface. This also raises the issue of which materials are required, and especially finding a suitable spherical surface is a relevant problem. Two solutions will be discussed, and the most striking distinction between them is in the surface they use. The first uses only half (or less) of a sphere as a surface, whereas the second one uses a sphere that is almost completely closed. It only has a hole in the bottom (on which the sphere rests anyway). The setup using half a sphere as the surface is the one implemented for this thesis. The results of implementing this system serve however as an excellent stepping-stone towards a system with an almost complete spherical display. The difference of surface changes the physical setup, but only introduces a few new problems on top of the old ones that would need to be compensated for.

The possibility of an approach using several cameras was mentioned, but both expenses and computational effort (in the form of handling and comparing the input of several cameras) can be spared by using online one. As the Sphere setup also uses only one camera this decision lets the implemented setup serve as more of a stepping stone to the Sphere implementation as well - and with it to a more completely spherical touch surface.

2.2.1 Implemented setup

This first solution is the one implemented as the spherical surface was more easily acquired. The surface used is half of a sphere and obtained from the company Globe4D. The setup dictates that this surface is mounted on top of a pedestal in which the rest of the hardware is hidden (see Figure 2.11a). Globe4D normally uses the spherical surface for their own touch-display solutions, though their setup did not otherwise inspire this implementation.

After the physical setup is discussed the consequences for the camera’s image that will be interpreted will be considered. Finally some additional difficulties will be discussed that have been introduced by using this setup.
The setup

A sketch of the physical setup used is shown in Figure 2.11b (though with the camera and projector used, this exact setup and in particular the box may not be possible). Notably this image does not show the actual computer to which the projector and camera are connected, as this is simply somewhere out of the way and not relevant for the method.

(a) The setup as a user sees it (a Globe4D display).

(b) The inside of the setup used. 1. Camera 2. Projector 3. Infrared pass filter 4. Mirror(s) 5. Infrared LEDs placed in a circle around the setup, aimed upwards to the surface.

Figure 2.11: Globe4D setup

The sphere is positioned above the ground by resting it on a surface that cuts of the edge. This also means a part of the surface is blocked from showing up on the camera’s image. This does not matter much for a hemispherical display as touches on the edge of the screen may have been nigh-impossible to detect in the first place. This is due to the fact that touches on different areas of the sphere do not always show up as the same size for the camera, as illustrated in Figure 2.12. The change in size of the blobs is due to the angle between the surface that is being touched and the camera. Because part of the sphere may not show up on the camera, care should be taken in making sure the user cannot not directly touch areas that will not respond. This could be done by obscuring a constant amount of surface from all sides on the bottom of the sphere, as done in Figure 2.11. The remaining screen that can be interacted with will be referred to as the touchable area of the screen.

The first important conflict of interests is that both the projector and camera would prefer to be positioned perfectly underneath the screen to avoid any distortions in their image. In order to solve this mirrors are used to virtually both place them at such a position without being in eachother’s way. Though these both mirror and slightly reduce the brightness (much more so when they get dusty or dirty) of the projected image, this does not affect the touch detection. Imperfect mirrors may introduce new more distortion into both images, but the effect will be considered negligible (and even if it does introduce some distortion this could programatically be estimated and compensated for by analysing how a known pattern is distorted).

Using mirrors has the additional benefit that the virtual distance of the imaging device to the surface can be larger than the distance of the surface until the ground, which makes it easier for the projector’s image to cover the entire surface and for the camera to see the entire surface. As the system is quite compact, fitting in the box of Figure 2.11, this distance is still too small.
for a regular (budget) projector to give a focused and undistorted image. Instead a short-throw projector is required, which specialises in such distances. If the available distance is still too small for either imaging device a different lens could be used.

The choice of lens used for the camera and projector has influence both on the physical setup, the touch detection and the image required when projecting onto the screen. It has influence on the physical setup as a wide angled lens allows the camera to be closer to the object by distorting the image viewed with something called radial distortion. When using a longer angled lens this is not so much the case, though the camera may have to be moved quite a distance from the surface. The distortion of a lens with too much of a long lens shows distant objects abnormally large and objects close by abnormally small, making it hard to discern relative distances. The influence on both the touch distortion and the image that is required to properly display an application is that the distortion will have to be compensated for. In this setup a wide angled lens is used for the camera, and the compensation will be a topic later. When the angle gets very wide it is known as a fisheye lens. When using any lens it is worth considering that the centre of the lens is more correct than the edges.

Another problem both the camera and projector share is that they have to be focused on a plane instead of a spherical surface, influencing both the projected image and the camera’s image. An improperly focused camera or projector can produce a somewhat blurry image, and for the camera this means that small blobs especially may blend into the background. Near the edge of the screen blobs have been shown to show up smaller, and in order to compensate for the increased level of detail required the camera should be focused around there. The projector should be focused more near the top of the sphere as the density (and therefore resolution) of the rays of light from the projector reaching this place is smaller than it is at the edges (nearer to the camera itself).

Both the camera and the projector used need to chosen such as to minimise the response time incurred between the touch detection and reaction on the projected application. The speed requirement in fact also dictates which PC hardware is suitable. These considerations lead to the choice of a Firewire camera, which can quickly communicate the images it detects at 30 frames per second. It is of course also the case that both have to be fastened tightly to the physical setup, because any movement during or after calibration will introduce errors into the touch-recognition and proper projection (requiring the user to re-calibrate the system). The chosen projector needs to be a short-throw projector with as much brightness as possible, as the diffusing surface makes the projected image dimmer.

The next major problem is how to uniformly flood the inside of the sphere with infrared light. As mentioned before a diffuse layer is applied on the inside of the sphere in order to stop some of the reflection, but the source and direction of the light sources play of course a much larger role. As a light source infrared LEDs are used which focus their light in a fairly narrow angle (as these were most readily available). By putting a diffuse layer in front of the LEDs the light could be spread out a bit more, but this is at the cost of the light’s intensity. The pattern suggested for the LEDs is as Figure 2.11b shows, shining at the other side of the screen. In the implemented setup however the illumination was already decent enough simply placing several LEDs beneath the screen aimed upwards. Note that the camera should not directly view the infrared light sources, hence the protection. Related to this it should also be noted that the diffused illumination technique requires that the camera’s image is static (or obscured) where no touches occur, which is the case in the closed box that is proposed. When considering the lighting of the sphere it should finally also be remembered that IR lights can be harmful the users’ eyes (especially as it does not trigger a protective blinking response). There are classifications for IR lights to inform about the danger the lights pose, and a set should be chosen that does not harm eyes much.

Lastly, in order to further ensure that the projecting and viewing happens at different frequencies of light a filter is placed in front of the camera which only lets infrared light pass through it.

Future images referring to the physical setup will use the simplification shown in Figure 2.14 instead of the earlier image. This simplification is used to make it more it obvious how rays of light (such as the blue one) travel. It is also for this end that the camera and projector are shown at their virtual location and that several components (such as the mirror) have been removed.
They are not positioned at their ideal location underneath the sphere as some inaccuracy is likely to occur, and by exaggerating this effect the consequences are easier to understand. It is noteworthy that this also means that blobs that occur closer to the camera are described with slightly higher resolution (leading to somewhat more accurate touch detection).

Figure 2.13: (Exaggerated) look at how the surface used is not equally thick at every point, but thinner at the top than at the sides.

The resulting setup is shown in Figure 2.15. The consequences of using this setup for the camera and projector’s images will be considered next.

2.2.2 The image

The non-flat shape of the surface has a significant effect on the projected image as well as the image the camera detects. When a projector is placed perfectly under the centre point of the sphere and is pointed upwards it projects its rays of light such as Figure 2.16 shows. When the image of a circle is projected from this position the rays of light are intercepted by the shape of the screen, and it can map the circle’s image onto the sphere. The way this projecting of a circle unto the sphere happens can be described and anticipated for in order to project images that do not look distorted on the display (such as Figure 2.17 suggests).

In reality the projector is not perfectly positioned however, and consequently the projected image does not map properly unto the sphere. The result is an image that is distorted such that
Figure 2.16: The projector in the ideal position can simply project a circle unto the green plane in order to project something unto the sphere.

Figure 2.17: The mapping between the circle and sphere can be modelled and used to distort images before projecting them, so that they look good once projected unto the sphere.

it is no longer a perfect circle: rays aimed towards the back of the sphere travel longer than they should for a normal image, resulting in the effect illustrated by Figure 2.18. In this Figure the grid (and therefore so could a circle that is to be mapped unto the sphere) could be projected to look as expected unto a plane, but has its imperfect position resulted in a distorted grid when projected unto a sphere. This is the same grid that has been projected onto Figure 2.15, and it illustrates the distortion caused by a projector that did not programmatically compensate for its perspective.

Handling the projection of undistorted images with a consistent brightness is outside of the scope of this project, though other students at the UvA have started work address such problems. The same distortions are also at play on the images detected by the camera however, and will need to be kept in consideration. Solving this projection is also somewhat relevant to the touch-detection for applications.

Figure 2.18: By changing only the surface the importance of the projector’s (and camera’s) perspective is shown.

If the image centre is not projected perfectly onto the sphere, meaning that the top point of the sphere and where the application thinks this point are do not match, this will not only mean that the projected image looks distorted. It can also be the cause of a large discrepancy between which points the touch-detection calls \((x, y, z)\) and which points the application refers to when it says \((x, y, z)\). As calibrating the projector’s image falls outside of the scope of this project,
it will sometimes explicitly be assumed that certain points are projected accurately. Extending the existing touch-detection implementation to give the information required for the calibration of the projector would not be much work however, as the user is asked to indicate relevant areas such as the edge and top of the screen during calibration regardless.

This setup also has several other properties worth consideration.

Difficulties

We now turn to some particular difficulties that come with in this setup, which follow from the properties of the spherical surface used. To help understand which problems a spherical and in particular the Globe4D surface poses problems its manufacturing process will be considered.

The creation of the surface involves having the surface material in liquid form and then blowing air against this from below in order to create the spherical form. In liquid form a material will strive to be flat and of a consistent thickness, but once the air gets blown against it the material will of course start to bulge which means it will cover a larger surface area. Consequently the material will be thinner and stretched out at the centre bulge, while it stays thicker near the edge (illustrated in Figure 2.13). It is also important that this manufacturing process is not done in a way that is completely free from external forces, and because of this it is possible that someone opening a door in the factory could lead to the slightest gust - which will affect the forming shape of the surface.

Two problems worth discussing arise from this process. The first is that it has already decided to put the diffuse layer on the inside of the sphere. When this is combined with the uneven thickness of the sphere, the user’s experience could be affected. He might interact differently and more accurate near the top of the sphere compared to near the sides, and this might not feel intuitive at all. This depends of course on how much the thickness changes, and for a smaller surface this may pose less of a problem than for larger surfaces.

The second problem is that the effect of the wind on the shaping of the surface makes every surface somewhat unique and makes it possibly require individual calibration. How relevant this is may again vary depending on the size of the screen used.

2.2.3 Microsoft Sphere

Though only a hemisphere is implementated as a touch-surface, it is possible to consider a setup using a more complete sphere. It turns out that Microsoft has already done research on such a sphere in the past[11], and their solution looks like Figure 2.19a. The main problems introduced on top of the ones for the other implementation are twofold. First one has to find a suitable surface (and possibly apply a diffuse layer) and lens. Second and most importantly, there has to be compensation for the fisheye lens used in the setup in order to do a projection of a 2D circle onto a sphere, such as shown before in Figure 2.17). Both this projection and the non-linear distortion the lens that was used introduces need to be compensated for.

The setup

The implementation relies on using a spherical surface with a hole in the bottom, in which a fisheye lens is placed. The lens can now be used for projection a sphere projected onto a plane. This way it is possible to project a 2D circle into the lens and the light rays will be scattered onto the entire surface of the sphere. This setup makes the most of this property by both projecting the image through the lens onto the surface and looking at the surface through the lens.

Because the projector and the infrared camera both want to use the lens and look directly into it, a cold mirror is introduced. A cold mirror lets infrared light pass through the mirror, which means the camera can be set up below the mirror looking at the camera, while at the same time reflecting the visible light the projector uses.

The source of the infrared light is a ring of LEDs around the lens, which attempts to cover the surface as uniformly as possible with infrared light. When a user touches the screen, infrared rays of light are reflected such as the image shows. Part of these reflections find their way back through the lens onto the camera screen, and so the touch-recognition can begin (though it is not straightforward to convert points on the 2D image back to their corresponding location on
(a) The setup used for the Microsoft Sphere. (b) The path of a light ray when projecting applications (left) or detecting touches (right).

Figure 2.19: Microsoft’s Sphere’s setup

the surface). Special care is taken so that this light can not directly shine into the lens and show up on the camera, and the diffuse property of the ball (or diffuse layer on the inside of the ball) makes sure reflections of the light are not too big of a problem. Figure 2.19b shows both how images are projected onto the screen and how touches are recognised.

Finally, infrared pass and infrared cut filters are applied again to help make sure that the blobs which show up on the infrared camera actually correspond to touches.

Difficulties

Programmatically there are two new sources of challenges. The first is that some of the problems from the first implementation are amplified, such as an infrared illumination which is not uniform and a susceptibility to inaccuracies in the shape or obtained during calibration (as more distortions have to be properly fixed using similar calibrational methods). The second is that the lens its distortion has to be compensated for (both at touch-detection and at image-projection), which is stronger than the one introduced by even wide-angled lenses. This problem has been addressed before in the literature[17] and code for the fisheye mapping is even available online for free.
CHAPTER 3

Processing the camera image

So far it has been decided to build a touchscreen by pointing a camera at a surface in order to see blobs from which it can be inferred how the user interacted with the surface. It has also been discussed how to set up hardware in order to get this camera image, and now the interpretation of the camera image will be considered. It turns out that this image requires some pre-processing, regardless of which optical technique is used. Even when the screen is not being touched some infrared light may end up into the camera, for example by a static lightsource. Depending on the optical technique (this includes rear diffused illumination), the camera may also pick up objects which are slightly above the surface. The goal of the pre-processing is to produce a camera image with only blobs indicating the actual touches on the surface.

After only the relevant blobs remain the (centroids of the) blobs will be found, labelled with an identity and tracked over time. The positions of the centroids found will not be exact as most lenses used by camera’s (especially ones such as the wide-angled one used in the experiments) introduce some distortion into the recorded image. Only after all this is addressed an effort can be made to find out how blobs on the camera’s image map to blobs on the surface.

Both of these tasks will mainly be addressed by the open source Community Core Vision (CCV 1.4) software, along with the help of a sample application of the open source OpenCV 2.1 software. All of the algorithms discussed (including the open source ones) are implemented in C++.

3.1 Pre-processing

As mentioned before it is now necessary to filter everything from the image which does not correspond to an actual touch. The pre-processing consists of first subtracting the ‘background image’ (a ground truth) from the image the camera detects and then applying several additional filters to the image in order to further clarify where and when the screen is actually being touched.

3.1.1 Background subtraction

In whichever methodology chosen, the camera image will not only ever show the points of contact with the surface. There may be other static sources of infrared light and the light from the light sources used in the setup are also likely to at least partly reflect back into the camera. The result is that the camera’s image will pick something up even when nothing is touching the screen, and this image may interfere with the blob detection. Such an image is often the same every frame, and a common way to solve it is to take a snapshot of what the camera sees when nothing is touching the surface and to subtract this background from every new frame that is processed (resulting in a black image when nothing new enters the frame). On Figure 3.1 the input image can be seen at the top left and the background image at the bottom left, leaving the other screens to be white on black. Items may however be added to or removed from the background, especially if the surface is put on public display. Applications will experience a lot of interference as long as new blobs are not properly recognised as being part of the background, and therefore a
Figure 3.1: CCV’s interface when started, showing the image at the different steps of the (pre)processing process.

dynamic background subtraction may be necessary. CCV has such a method implemented, and it simply takes an average of the background image of the previous frames and the image that is seen this frame. This is including blobs which actually represent blobs. The rate at which new blobs are assumed to be part of the background can be selected, but usually this should simply happen at a rate which is slow enough to allow for all the meaningful interactions a user could make. The cost of not being able to use the screen for 30 seconds is lower than the cost of having interactions be ignored. Dynamically determining the background does come at an additional computational cost however, so it should be avoided if it is not necessary. An alternative would be to allow the user to press a button whenever a new background image should be taken.

The method of background subtraction has at least one important weakness however. If there is a source of infrared light which is stronger than the infrared light used for the touch detection it will not register a finger moving onto the screen as a touch (it is after all not brighter than the background lighting). If dynamic background subtraction is on it will even start learning that the background is darker than it really is - with the result that as soon as the user removes his hand again the system does detect touches. This is quite troublesome for a device with possible prospects for public display, as many lamps produce infrared light. A possible approach would be to apply an infrared cut filter to all the lamps, although this may be impractical. A more robust method is called for.

The example with the infrared light source hints that a significant deviation in value may be more of an indicator of touches than simply reaching some threshold value. In the publication about the Microsoft Sphere it was mentioned how the calibration of their system involves taking one snapshot when nothing is touching the screen and one snapshot when the entire surface is being touched (perhaps by wrapping a blanket around it). Though this calibration is more tedious than the one used by CCV it does shift the focus to deviations in value, and even if there was nothing dynamic behind the algorithm it would at least be capable to cope with static sources in a way that CCV can not.
3.1.2 Filtering the proper blobs

As the different optical techniques create the blobs in slightly different ways, the filters required also vary somewhat per method. In the case of diffused illumination (see Figure 3.1 for the possible filters and their effect) first the highpass filter has to be set up properly, as touches closer to the screen show up brighter. This has sliders for blurring the image and filtering noise in order to make sure only actual touches (expected to show up bigger than random noise) gets shown. If the blobs of touches do not show up clearly enough to distinguish from false positives the amplify filter can be used to fix this. With it blobs which were initially hard to keep apart will start to look a lot more different (as higher values get increased more by the amplification). Using these filters it only actual touches should be left on the camera’s image, which means these touches can now be detected and tracked.

3.2 Recognising interactions

Given a black image with bright blobs indicating touches the different touches can be found, given a unique identifier and tracked over time. The position of the touches on the camera image is actually a bit distorted because of the camera’s lens and will therefore be corrected once detected. This is the last step of processing the camera’s input before work can start on translating detected touches to positions on the actual sphere.

3.2.1 Recognising touches

CCV detects blobs in the image and describes their location using their centroid, which can be determined at subpixel accuracy. It is worth remembering that the centre of a touch is not necessarily at the centre of the detected blob. Figure 3.2 shows that when a circle (or blob, or touch) is viewed from a different perspective that the centre of the touch is not the circle’s centroid. This inaccuracy is quite small however, and the computational effort required to fix this inaccuracy probably outweighs the benefits it would bring. This inaccuracy makes detecting the centroid (position) of blobs a bit more accurate for smaller blobs.

3.2.2 Camera distortion

As mentioned before it is usually the case that a camera gets a distorted image due to an imperfect lens. It is possible to compensate for this and even to correct the entire image before starting the pre-processing of the image, but this is computationally far less efficient than performing the calculations only for the few centroids detected in a frame. Most lenses introduce inaccuracies in two main ways: radial and tangential distortion. In radial distortion a barrel or pincushion distortion (or a combination of one transitioning into the other, known as a mustache distortion) is introduced into the image, which leads to pixels near the image’s centre being larger or smaller compared to pixels near the edge of the image (an example of barrel distortion is shown in Figure 3.3). Tangential distortion (also known as decentering distortion) is introduced as the optical centres of the various lens elements in the camera are not perfectly aligned to each other and the camera its imaging plane. There are some other distortions camera’s can introduce (such as discolouring distortions, as different wavelengths of light can be bent slightly differently by a lens), but they are not quite as relevant.

The most common solution to model what a camera sees is called the camera matrix, which is a pinhole camera model which converts world coordinates into camera image coordinates. This model can be extended to be Brown’s model, which can describe the distortions mentioned before using extra parameters. The pinhole camera model is used to model the way world coordinates are translated to coordinates on the camera. The position of a point in the three-dimensional world will be found on the camera’s image at the position obtained by going through the extrinsic
and intrinsic camera matrices. The intrinsic matrix models how the camera inherently views the world whereas the extrinsic camera describes how the camera’s position in the world affects the coordinates of anything seen.\footnote{Though some refer to only the intrinsic matrix with ‘pinhole camera model’, this term will be used interchangeably with the entire camera matrix}

\[
\begin{bmatrix}
  u \\
v \\
1
\end{bmatrix} =
\begin{bmatrix}
f_x & h & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
r_{11} & r_{12} & r_{13} & t_1 \\
r_{21} & r_{22} & r_{23} & t_2 \\
r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\]

A camera’s intrinsic matrix encompasses the focal length, pixel size, principal point and skew of the camera. The focal length and pixel size are represented by $f_x$ and $f_y$ ($f_x$ or $f_y = \text{focal length}_x$ or $y \ast \text{scaling factor}$). The principal point is translated to the centre of the camera’s image using $c_x$ and $c_y$ and the skew is indicated by $h$. As skewing of the image does not play a role in our situation, this will be ignored (it will be assumed that the camera is chosen such that there is a negligible skewing effect).

The extrinsic matrix of the camera indicates how the camera is positioned in the world: it consists simply of a composed rotation matrix and a translation matrix to indicate where and from where the camera is looking into the world.

Brown’s camera model\cite{18} (also called the Brown-Conrady camera model, as Brown extended Conrady’s model) extends this model by adding more non-linear intrinsic parameters to compensate for radial and tangential distortion. First the pinhole model is rewritten to look as such:

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = R
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + t
\]

\[
x’ = x/z \\
y’ = y/z \\
u = f_x \ast x’ + c_x \\
v = f_y \ast y’ + c_y
\]

After which an intermediary step is added to describe the distortions:
3.2.3 Tracking touches

Every frame the camera looks at the blobs present and gives all new ones a unique identifier. With every new frame it then wants to find out whether the blob still exists and how it moved. For every blob at time $t$ it searches for the blob at $t-1$ which has the smallest euclidean distance.
(if the distance is sufficiently small that the blob probably did not move the loop breaks early). It tentatively says there is a match between blob $a$ and $b$, and continues finding matches for the other blobs at $t$. If $b$ is also the best match for another new blob, the match with the smallest euclidean distance wins. If blob $a$ loses this then it is simply assumed to be a new blob and it gets a new identifier.

This works pretty well because the 30 frames per second are enough to make sure the blobs do not travel large distances, and the size of the blobs (fingers) are also large enough to play a role in increasing the possible distance between centroids in a frame. Though this is not the most accurate way to solve this (the blob with the second smallest distance to $a$ could be found), this does avoid performance problems in situations where a lot of blobs were matches with their second-best option (where one blob gets its best math stolen and in response steals the match from yet another one).

There are certainly smarter algorithms (for example predicting the likely path of a blob by considering which direction it went and with what speed) available which are computationally more intensive, but the amount of false positives with this method is small enough for warranting the choice of a quick algorithm.
CHAPTER 4

Matching blob and finger positions

The (pixel)position of blobs on the camera’s image can now be determined which means that it is time to find out how this position translates to the position on the surface that was touched. on top of a sphere and two coordinate systems (cartesian coordinates and spherical coordinates) will therefore briefly be discussed.

When pointing a camera at a flat touch table it looks the same from every angle (aside from the effects of a change in perspective). This is not the case for spherical surface, as discussed before. Even if only a plane is considered however, there are several factors that play into distorting the way the image looks to the camera, shown in Figure 4.1. First the projector sends out the image of the sphere. This image already has to be distorted in order to compensate for the sphere’s shape as well as the projector’s perspective (to look perfect on the screen even though the projector is looking at the surface at an angle its perspective first has to be compensated for). As the perspective compensation may not be perfect this will result in a slight distortion by the perspective on the image the camera will view (one such as exaggerated in Figure 4.1c). Finally this distorted circle will be looked at from yet another perspective by the camera, which results in the camera seeing an image such as figure 4.1d. Notably not every part of the surface shows up as the same size on the camera’s image, and the parts that are described with the higher resolution (bottom left of the image) may have slightly better touch detection than elsewhere for two reasons. Firstly blobs show up larger there, meaning that small blobs are more easily separable from just noise. It also means their location is described with more accuracy in the first place.

The image projected onto the surface (Figure 4.1b) could still be found through a perspective transformation in the case of a flat surface was used, but this is not as easy for changing a perspective in a three-dimensional scenery. Figure 4.3 illustrates how the shape of the sphere influences what the image looks like: the image that is seen is not the same as the one that was projected (i.e. the projected circle on the green plane or on the plane in the middle of the image). Note that the yellow plane goes through the centre of the sphere and is perpendicular to the ideal camera position.
Figure 4.2: The result (in black) when a square is projected onto a sphere without first compensating for distortion. The red shape is obtained by simply connecting the four points of the projected square. The purple dot is a place the user touches the screen.

(Exactly under the middle and pointed upwards). This plane will be referred to as the **centre plane**. These distortions also show that the mapping from a blob’s location to a location on the surface therefore is not as straightforward as it would be with a flat screen, and requires extending the CCV codebase.

It is also possible to use touch applications that are not intended for spheres, though at a cost. As the image was not distorted to show up properly on the screen a full-screen application would partly be cut off (only a circle-like section of it would appear on the screen) and the image would be distorted to look like Figure 4.2.

Three methods will be introduced in order to find the touch’ proper coordinates, nicknamed the barycentric method, the perspective method and the calibration method. Work on these methods has been integrated into the CCV source code. In the barycentric method the user is asked to touch a lot of points projected onto the screen as referencepoints, and the location of touches is determined by the surrounding referencepoints. The principle behind the perspective method is asking the user to touch four projected points which will be used to get an view that is not distorted by perspective. Using an intersection between the sphere and the line that goes through the blob’s position as well as the camera’s position it is then possible to find the correct point. In the calibration method the user is asked to touch several permanently marked points, which gives a coordinate system in which the physical sphere’s properties (size and position) are known. Then it is again just a matter of an intersection of a sphere and a line.

These methods will highlight and address eachother’s shortcomings, but first it will be considered how to represent points on the sphere’s surface.

Figure 4.3: The effect of the surface’s shape on the projecting and viewing of the application’s image. The coloured planes have the projected image on them with a distortion by perspective due to the camera’s position. Images on these planes could be viewed by the camera with only incurring a new distortion by perspective, but the coloured rays of light indicate how the shape of the surface makes sure that the image on the surface is quite different from both perspectives.

### 4.1 Coordinate systems

When a user touches the screen a blob shows up on the camera’s image. The location of this blob (or rather, its centroid) is described using the pixel coordinates of the image. The position of a touch on this image will be referred to as its **camera coordinates**. Every touch is on top of an image that is being projected onto the screen. The pixel projected at the same point where the user touches the screen will be referred to as the **screen coordinate** of the touch.
The most straightforward way to communicate the position of a touch is to somehow transform camera coordinates into screen coordinates and to send this over to the application. It turns out that a lot of existing touch applications do in fact request a normalised version of screen coordinates (such that values for width and height are in the range [0,1]), but this may not be ideal for applications intended for spherical displays. The image that is projected onto a sphere needs to be distorted in order to look undistorted on the screen, meaning every application would have to undistort the screen coordinates they receive at some point (before possibly also having to map the coordinate back to some other coordinate system). The main alternative for this is to directly give the coordinates of the touch on the sphere. This higher level of abstraction may also make it easier to develop applications for the system.

There are several coordinate systems available for describing points on a sphere and it is easy to convert coordinates from one system to another, but which is most ideal for the interface offered to applications is not so clear as different applications could use different systems internally, and there is no universal standard. Two of the most common systems for describing points on a sphere are the cartesian coordinates and spherical coordinates. Both have been implemented.

4.1.1 Cartesian coordinates

Just like pixel coordinates are a set of \((x, y)\) values it is possible to use a cartesian coordinate system for describing \((x, y, z)\) positions on top of the sphere, where \(x^2 + y^2 + z^2 = r^2\) (\(r\) being the radius of the sphere, and origin \((0, 0, 0)\) being at the centre of the sphere). The system is depicted in Figure 4.4. One advantage of the cartesian system is that it may be the most intuitive for most people as it is the system most commonly used throughout education. It is also equidistant over the surface of the sphere, making some operations such as comparing (euclidean or other) distances easier for applications. It should be mentioned that the equation \(x^2 + y^2 + z^2 = r^2\) does not hold exactly due to floating point inaccuracies, and that this eventually translates to less accuracy in the touch detection.

![Figure 4.4: The cartesian coordinate system as offered in the interface to applications.](image)

4.1.2 Spherical coordinates

An alternative to this cartesian system is using spherical coordinates, which (as the name implies) is specifically intended for describing points on spheres. Points are described using three numbers: the 'polar' angle \(\theta\), the 'azimuthal' angle \(\phi\) and the radial distance \(\rho\). In the form that will be referred to (visualised in Figure 4.5), the centre of the sphere is at the origin of the coordinate system and the surface of the sphere is at a distance \(\rho\) in all directions. The direction that corresponds to a certain point is described with the two angles. The angle \(\theta\) indicates how closely the point is positioned to the top or bottom of the sphere. It is 0\(\pi\) for the point that the top of the sphere and 1\(\pi\) for a point at the bottom of the sphere. The angle \(\phi\) denotes into which direction on the circumference the point is positioned, and ranges from 0\(\pi\) to 2\(\pi\).

![Figure 4.5: The spherical coordinate system as offered in the interface to applications.](image)
A famous example of a spherical coordinate system is the geometrical coordinate system, which has longitude latitude values corresponding to the $\theta$ and $\phi$ angles and uses them to describe points on the earth’s surface. The difference with this system is that the angles are measured from different points in the system. The fact that no one ever mentions the $\rho$ equivalent for this system hints at an advantage over the cartesian system for touch applications: it generalises easily across sphere sizes without requiring normalisation, which may save some computing time in calculations.

Now that the coordinate systems have been introduced it is possible to talk of how it is possible to go from camera coordinates to either the cartesian or spherical coordinate systems. Unless noted otherwise the methods assume that the a perfectly spherically shaped surface is chosen.

4.2 Barycentric approach

One straightforward approach to find out how camera coordinates correspond to projector coordinates is to introduce a calibration step in which the user is asked to sequentially touch a lot of projected points in a grid pattern such as in Figure 4.6. The user first has to move the grid in place as the projector does not know where the screen is standing and how it is oriented) and is then asked repeatedly to touch the screen at an indicated place. From this calibration pairs of points can be remembered: the projector coordinates of a projected point and matching the camera coordinates of the point the user touched. The points on the camera’s image can now be used as reference points of which is known how they map. Whenever a new touch then occurs it is located within a square of four reference points (and a triangle of three reference points), and as Figure 4.7 suggests it could be attempted to find the screen coordinates of this new value by looking at its camera coordinates compared to the reference points. This is what is what the barycentric approach aims to do. This method has been mentioned before at least by Muller in his thesis, and it is the method that CCV uses for flat screens, and its use with a flat screen will later on serve as a point of reference to compare the proposed techniques to techniques that are already in use. It works sufficiently well that people often do not undistort the camera image from the distortion caused by the lens, but instead increase the number of points to press during calibration.

![Figure 4.6](image)

Figure 4.6: The green plusses indicate the grid that is projected for the barycentric approach. The red and yellow lines indicate how points can be found of squares or triangles of reference points using this grid.

Barycentric coordinates are applied by saying that given a triangle, the location of any point $P$ on the plane of the triangle can be described as $P = \alpha \cdot A_{\text{position}} + \beta \cdot B_{\text{position}} + \gamma \cdot C_{\text{position}}$. 

34
where \( \alpha + \beta + \gamma = 1 \) in the event that \( P \) is inside the triangle. The system is applied by describing the location of new touches in terms of three of the surrounding reference points. Currently the program loops through all triangles until it finds the one in which the new camera coordinate fits. From here on the \( \alpha, \beta \) and \( \gamma \) values can easily be found because the sizes of the surface areas \( \alpha, \beta \text{ and } \gamma \) in Figure 4.7 are already representative of the weights of points A, B and C, and only need to be normalised to add up to 1. The only thing then left to do is to use the barycentric coordinates found with the camera coordinates on the screen coordinates which match those reference points.

![Figure 4.7](image)

Figure 4.7: Using barycentric coordinates, point \( P \) can be described in terms of the locations of three surrounding points. The influence A, B and C have on \( P \) is indicated by the values \( \alpha, \beta \) and \( \gamma \), which are equivalent to the size of the area they indicate in proportion to the entire size of the triangle. This can be done for both the camera’s image (left) and the projected image (right).

Though this method has been applied with success on flat screens it is not ideal for spherical displays. For example, if the square of a grid looks distorted on the camera’s image due to being projected onto the shape of the sphere such as in Figure 4.2, point \( p \) might not even be found to be in the triangle of screen coordinates that was actually touched. Distortions can be countered to a degree by increasing the amount of points in the grid, but it becomes quite tedious for the user to calibrate the system. It should also be noted that the user’s input probably is not going to be perfect, and might have a somewhat random as well as systematic deviations from the points he was supposed to touch. This could also mean that parts of the screen are more accurate than others. As a rectangle does not fit perfectly on the shape of a sphere this means that the grid will overlap the sphere. This in turn means the edge of the screen will not be covered in referencepoints. The method has been modified such that every referencepoint which falls just outside of the points that can be touched will be inaccurately estimated by finding the closest square which is entirely in the circle, and then assuming that the square with the unknown point has the same shape.

Implementing this method shows the limits of the techniques used for flat screens, and gives a point of reference to compare with the other methods with. Because of its role no spherical or cartesian coordinates will be calculated from it, but only the screen coordinates. It will only be used with applications that are not intended for spheres. This is the only approach that does not require the assumption of using a perfectly spherical surface.

### 4.3 Perspective approach

The starting point of the second method is that the task of mapping camera coordinates to coordinates on the sphere’s surface would be easier under the assumptions that the image on the sphere is projected perfectly and that the perspective of the camera would not introduce any distortion either (that it is perfectly positioned). In this case the touch detection would only have to compensate for the distortion caused by the shape of the sphere in order to get the circle that was projected onto

![Figure 4.8](image)

Figure 4.8: The perfect situation: the displayed image has no errors and the camera’s perspective does not incur distortion.
it, and a situation such as Figure 4.8 occurs. In it the camera is parallel to a plane that intersects the sphere through its centre point (the centre plane), and the application’s projection of a circle onto the sphere could be found intact on this plane. Note that this situation does not require an exact hemisphere, as this centre plane will always exist for a(n incomplete) sphere. The green plane would still be in the same location if the physical screen covered less or more area of the same sphere.

When someone touches the screen (represented by the blue ray of light in the image) a blob occurs on the plane that the camera sees. Under the assumption that in the three-dimensional cartesian coordinate system, as represented in that image, the location of the camera as well as the location and size of the sphere are known it is possible to find the coordinates of the interaction. The blue ray of light indicates that the position of the blob is known to cross both the position of the camera as well as a point on the centre plane (this is on the camera’s image), which means a line can be found on which the interaction took place. The furthest intersection of this line and the sphere is the position where the touch occurred. At least one intersection should always be found, as blobs should only ever occur on the surface of the sphere in the physical setup that was built.

Two unsubstantiated assumptions were made here that are relevant for the touch detection. These are 1) the location of points on the centre plane can accurately be detected and 2) the (virtual) location of the camera, the location and size of the sphere as well as the location of blobs on the centre plane are known in the same cartesian coordinate system. Compensation for these assumptions will now be addressed in that order.

As the camera is not perfectly positioned it is not the case that a perfect circle will be seen which looks the same image as the one seen in perfect situation on the green centre plane. The distortion incurred by the change of perspective can be compensated for programmatically using a perspective transformation. A perspective transformation is a matrix chosen such that coordinates on one quadrilateral can be mapped unto a quadrilateral viewed from a different perspective (illustrated in Figure 4.9), which is represented as such

\[
\begin{bmatrix}
    x' \\
    y' \\
    1
\end{bmatrix}
= \begin{bmatrix}
    a & b & c \\
    d & e & f \\
    g & h & i
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]

For estimating the values of the matrix first at least four point correspondences need to be found, point correspondences being the coordinates of the same point from both perspectives. This is done as the above equation can be rewritten to the following function with four point correspondences stacked on top of eachother (though it can be extended with any number of additional points)

\[
\begin{bmatrix}
    x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\
    0 & 0 & x_1 & y_1 & 1 & 0 & 0 & -y_1'x_1 & -y_1'y_1 & -y_1' \\
    x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2'x_2 & -x_2'y_2 & -x_2' \\
    0 & 0 & x_2 & y_2 & 1 & 0 & 0 & -y_2'x_2 & -y_2'y_2 & -y_2' \\
    x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3'x_3 & -x_3'y_3 & -x_3' \\
    0 & 0 & x_3 & y_3 & 1 & 0 & 0 & -y_3'x_3 & -y_3'y_3 & -y_3' \\
    x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4'x_4 & -x_4'y_4 & -x_4' \\
    0 & 0 & x_4 & y_4 & 1 & 0 & 0 & -y_4'x_4 & -y_4'y_4 & -y_4'
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d \\
    e \\
    f \\
    g \\
    h \\
    i
\end{bmatrix}
= \begin{bmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0
\end{bmatrix}
\]

If the point correspondences were measured perfectly there should be one non-trivial solution, but as this is usually not the case the solution will be found where the zero vector is best approximated. Several problems a perspective transformation has is that pixel values may both need to be extrapolated (when parts of the resulting image are larger than what they were on
the original image) and interpolated (when the colour at pixel 7.3 is required for example).

Using a perspective transformation it is possible to turn the distorted circle on the camera’s image into a perfect circle. It is important to note here that this is the case for the mapping of a plane upon which these points are located, and not a three-dimensional object. To find out how the circle looks on the camera’s image the user is asked, just like with the barycentric method, to move a grid around on the projector’s screen until it is properly on the screen. The difference is that this time the grid consists of a plane made out of the four points of a square which are on the inside of a projected circle such as shown in Figure 4.10. The user is asked to perfectly fit the circle around the edges of the touchable area on the screen (or equivalently, to maximize the surface area of the square that is projected on the sphere’s surface).

As the projector’s perspective can also distort the image that is shown on the sphere, the user is allowed during calibration to not only adjust the position and size of the grid, but also to perform perspective transformations using the four points, so that it can also compensate for the perspective at which it is projected.

Once this is (at least as close as possible to) perfectly projected the user is asked to touch the four points of the projected grid. These correspond to known points on a known circle as long as the circle is perfectly round and the radius is known (these are all \( \pm \sqrt{0.5 \cdot \text{radius}^2} \)). Using these point correspondences it is possible to find a perspective transformation which transforms the circle on the camera’s image into a perfect circle with known size and position. This allows for mapping blobs from their position on one circle to the other. The \((x, y)\) coordinates of the circle on the camera’s image can now be thought of as \((x, y, 0)\) coordinates if the (four points of the) new circle is chosen appropriately. This allows for a transition to a coordinate system in which the centre of the circle is at the origin and the radius of the circle is 100 (or any other value for that matter). Though the camera is actually viewing the centre plane from an angle the perspective transform allows the reasoning as if the detected plane was actually located on the centre plane (as one plane is simply mapped unto another). With this it is possible to know the position of the blob as well as the camera at the same time, but care should be taken that this image is not actually the same image as the one on the centre plane (illustrated by Figure 4.11): this positioning first has to be compensated for. This is possible as the coordinate of the blob can now be found both on the orange plane as well as its perspective transform onto the centre plane, as well as its ray of light traveling through the camera’s position.

With this, the first assumption has been compensated for. Note that this is not the same as what the ideally placed camera sees, as the sphere’s shape leads to more complexly different images from different perspectives. What has been achieved is all that was required however, as it is now possible to reason about the position of blobs as if they were on the centre plane, giving a known Z position.

The last assumption that needs to be addressed is that the radius of the sphere, the location of the camera and the positions of touches were known in the same coordinate system. It should be noted that the circle is perfect, but that it only represents the part of the sphere that can be interacted with; the touchable area (see the yellow intersecting plane in Figure 4.12). To find the real radius of the sphere the user is asked to measure both the radius of the sphere and the radius of touchable part in centimeters (see Figure 4.12 for clarification on which circles are meant). The perfect circle is normalised to fit the actual sphere, which already makes it in the
proper coordinate system. This leaves only the camera’s position to be determined.

Figure 4.12: Perhaps not half of a sphere screen can be used, but the part that is should be obscured until a plane perpendicular to the camera’s position. The yellow plane here is such an example. The diameter of the actual plane is the centre plane, and the diameter of the circle that is actually used is this yellow one. The four projected points must be on such a plane for the method to work. Note that the touchable area being at a different altitude has to be compensated for when intersecting rays with the sphere (as the origin is lower for the actual sphere, how much lower can be found using the radius of both planes and pythagoras).

This is simply the $(x, y, z)$ distance measured in cm from the origin of the sphere as well. The ideal situation is a $(0, 0, -Z)$ coordinate (as the surface is taken to be on the positive side of the $Z$ plane), but as long as the proper $(X, Y, -Z)$ location can be found the intersection is possible.

From here the intersection between a line and a sphere can be done in order to find the cartesian coordinates of the touch. Converting these to spherical coordinates is simply a matter of entering some values in known formulas.

The cartesian coordinates can also be translated to the screen coordinates as it is projected onto the centre plane by discarding the $Z$ coordinate. This value is then mapped onto the distorted circle which during calibration was projected onto the sphere to perfectly fit it (of which the perspective transform, radius and centre point were known) in order to finally get the screen coordinates as well as the spherical and cartesian coordinates of the point.

The key advantages of this method over the previous ones are the ease of calibration (a few physical distances need to be measured once, and then the user only ever has to place a grid and touch four points) as well as the fact that it is intended for accurately measuring every point on the sphere. This ease of calibration however comes at the possible cost of accuracy, as the error made in touching any of the points is amplified. Only if the points are perfectly touched or the user can compensate for the systematic errors a user makes in his touches (a user might for example always touch a point a little bit lower than requested: hence it is advisable for the user to walk around the sphere up to the point he should touch next before doing so) the calibration can be succesfull. One way to address somewhat random errors a user might make in touching the points is to ask her to touch these points several times and to use the average values. It is important the points are touched such that the centre (the top) of the sphere at least is accurately found, as the sphere on which touches are detected is otherwise slightly rotated compared to the projected image. Not only the inaccuracy of touches , but more importantly also the misplacing of the grid could endanger finding the proper centre point. Another disadvantage of the method is that unlike with the barycentric method, no surface larger than half of a sphere can be supported (as the furthest intersection is always found). Finally, it also is not ideal that the user has to measure physical distances himself as he can introduce more errors here (especially with regards to measuring the camera’s distance).

In the end this method seems like an improvement over the barycentric one in possible consistency and accuracy, though it allows too much room for error in achieving this accuracy.
and might in practice turn out to be worse. It may also be faster computationally as it is not necessary for every new blob to loop through an array of triangles and find whether the blob is in this triangle.

4.4 Calibration approach

The calibration approach is similar to the perspective approach in that it tries to find a coordinate system in which the circle is perfectly shaped and the location of the touch is found by the intersection of the sphere and a line that goes through both the camera and a known point on the centre sphere. It differs in that it does not have to trust the user to properly place the grid and instead has several points permanently marked on a surface, asking that the user touches those during calibration.

The coordinate system it uses in which the perfect circle is known will be referred to as the physical coordinate system (shown in Figure 4.13. In this cartesian coordinate system the physical sphere is kept in mind, with (0,0,0) being the origin and with coordinates being measured in centimeters distance from the origin (meaning the radius of the sphere in this system is equal to the physical radius measured in centimeters). The centre plane as indicated on the image lies on (0, x, y) and the x and y directions are chosen arbitrarily (though consistently).

![Figure 4.13: The physical coordinate system.](image)

![Figure 4.14: The sphere has to be elevated slightly higher than the flat surface in order to let touches on the flat surface (blue) and centre plane (orange) line up.](image)

It is now assumed that it is possible to temporarily remove the spherical display and to place a flat surface on the centre plane on top of which touches can be recognised. This screen has to be positioned perfectly, and this can be achieved in the physical setup because the platform around the edge of the screen that holds it up has a circular shape. By making sure the screen neatly fits into this before marking the proper points they will always be positioned properly.

Note that in order to get points which are actually on the centre plane, the platform for the flat surface has to be elevated slightly more than with the flat screen. This is as the thickness of the flat surface means that otherwise points slightly above the sphere’s centre plane would be measured, as illustrated on Figure 4.14.

It is on this surface that points will be marked to touch. To describe how points on the camera’s image correspond to points on a plane in the physical coordinate system the pinhole camera model will be used (Brown’s model would be inappropriate as it would re-introduce the lens’ distortion). The model was as follows:

$$ s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & h & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} $$
Though it describes how the world looks to the camera, it will be used to do the opposite using the same parameters that were estimated before. The intrinsic matrix as well as the rotation and translation matrices of the extrinsic matrix are all invertible, making it plausible the model could be used to describe points in the opposite direction if the unknown coordinates \(X, Y\) and \(Z\) as well as the unknown scaling factor \(s\) can be found. In order to get from the two-dimensional position to a three-dimensional position points are sought on the \(Z = 0\) plane (the centre plane). This reduces the amount of unknown variables. As the intrinsic matrix only changes the \(u\) and \(v\) values to new known values, the equation can be rewritten as

\[
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
= 
\begin{bmatrix}
  r_{11} & r_{12} & r_{13} & t_1 \\
  r_{21} & r_{22} & r_{23} & t_2 \\
  r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\begin{bmatrix}
  X \\
  Y \\
  0 \\
  1
\end{bmatrix}
\]

From this three equations and three unknown variables (\(s, X\) and \(Y\)) can be found

\[
sx = r_{11}X + r_{12}Y + t_1
\]

\[
sy = r_{21}X + r_{22}Y + t_2
\]

\[
s = r_{31}X + r_{32}Y + t_2
\]

Except for the cases in which these equations are linearly dependant (in which case a value could be changed to avoid problems at the cost of a slight inaccuracy), these unknown values can be found to be as follows (found as derived in Appendix A)

\[
\alpha = \frac{r_{21}}{r_{11}}
\]

\[
\beta = \frac{r_{32}}{r_{22} - \alpha r_{12}}
\]

\[
s = \frac{t_3 - \beta t_2 + \beta \alpha t_1}{1 - \beta y + \beta \alpha x}
\]

\[
Y = \frac{t_2 - \alpha t_1 - sy + \alpha sx}{r_{22} - \alpha r_{12}}
\]

\[
X = \frac{r_{12}Y + t_1 - sx}{r_{11}}
\]

It is such that it is possible to find the physical coordinates \((X,Y,0)\) which match camera coordinates \((x,y)\).

Now only the camera’s position in the physical coordinate system is necessary before the locations of touches on the sphere can be found. This could again be measured in the physical coordinate system (in centimeters from the origin), but in theory the extrinsic camera matrix already (and probably more accurately) describes how the camera is positioned with regards to the physical coordinate system. This can be found using \(0 = RC + T\), where \(R\) is the rotation matrix, \(T\) is the translation vector (together they make up the extrinsic matrix) and \(C\) is the position of the camera. This means the position of the touch can be found by doing an intersection of the sphere and the ray that goes through both the camera and a detected point on the centre plane.

The user will be requested to touch at least four points before the extrinsic camera matrix is estimated, though using more points will decrease the role of the user’s random inaccuracies in touching the points as well as increases accuracy for the algorithm with imperfectly marked points. During calibration the grid used for the barycentric method is projected to indicate which blob of the marked grid the user should touch next and whether a touch its location has been detected.
As it may not be ideal to use a flat surface during calibration (because the physical setup does not allow for easy replacement of the screen, or a flat surface is simply unavailable), there is an alternative of how the user can mark the points. Certain positions on the sphere would look the same to the camera as the required positions on the flat surface would, which is illustrated in Figure 4.15. This point is at the intersection of the sphere and the line that goes through both the camera and where the marked position on the flat screen would be. Marking the sphere’s surface is not ideal as it is not only permanently adding markers the user might be able to see, but it also requires manually measuring positions again (firstly of the camera and secondly of the intersection of the ray and sphere) which is hard to do accurately.

A final possible improvement is not to ask the user to touch the markers during calibration, but to automatically detect these from the camera’s image. This also means that a lot of points on the surface can be marked for increased accuracy (as the estimating method used benefits from having more points and the user is likely to be less accurate) whilst decreasing the effort required for calibration.

It is notable that if a flat screen is available during calibration, another method for finding the matching screen coordinates of a touch becomes possible. By treating the projector as a camera and finding an intrinsic as well as extrinsic camera matrix (for example by projecting the checkerboard pattern and looking how it is distorted using the camera), it becomes possible to first translate camera coordinates to physical coordinates and then to screen coordinates. How to accurately calibrate a projector has been a topic of research before[24][25], and can be done at subpixel accuracy.

The first advantage of this approach over the perspective approach is that the user is not required to properly place a projected grid, which already eliminates some room for error (this is one of the largest places of error in finding the proper centre-top point of the sphere). It also eliminates the need for the user to measure the location of the camera, not only making the position more accurate but also making the calibration easier. Unlike the perspective method this approach could be applied to a physical setup such as Microsoft’s Sphere uses, by positioning the plane used for calibration between the fisheye lens and the projector (as it is otherwise impossible to view the entire surface on a plane). This can be done as a position on the sphere will first have to travel through the fisheye lens again anyway in order to show up on the camera’s image. These advantages come at the cost of having a permanently marked touch-display (unless a flat screen can be used during calibration). The expected increase in accuracy and ease of calibration make this approach appear the most promising of the ones proposed.

Calibrating a spherical surface aside it could also be wondered how well this method performs to flat touch tables, as the plane that is calibrated could just be used for touches straight away. How comparable the performance of this method on flat screens is as that of the barycentric method is yet to be experimented with.
The last step for the implementation is to offer an interface to touch-based applications. A common solution for this is to use the TUIO (Tangible User Interface Objects) protocol, which is also implemented in CCV (TUIO version 1.1, specifically). In this implementation UDP messages conforming to the protocol are sent to a certain port on which the touch application listens. Three interfaces will be offered: one for the cartesian coordinates, one for the spherical coordinates and one for the normalised screen coordinates of a touch (which is what existing TUIO applications mostly use). This chapter will first discuss the protocol and then how the different interfaces will be implemented specifically in order to send different data.

5.1 TUIO overview

The TUIO protocol was made with both low latency and robustness in mind. The communication can be done using asynchronous UDP messages for a communication at a high speed at the risk of losing packets or them arriving in the wrong order. This would have terrible effects on the usability of the touch surface. TCP communication is also supported for a synchronous and more reliable connection, but this is of course much slower. As the UDP implementation is robust enough, this is the version that will be discussed.

Informed of the risk of packet loss, the following types of messages have been implemented for communicating information about touches: SET messages and ALIVE messages. SET messages are used to send information about every object’s state (such as its position or orientation) and are sent every time an interaction’s state on the surface changes. ALIVE messages indicate which interactions are currently present on the surface using a list of unique session IDs. The protocol is chosen like this instead of offering an event-driven interface (one where messages are sent at specific events, such as an ADD or REMOVE message when a blob is found or lost) as the latter is not well suited to an environment susceptible to packet loss. The TUIO client should infer events itself from the stream of messages it receives (by for example finding out at which frame a new session ID is present).

Every UDP packet that is sent contains a bundle of messages. Every bundle contains one ALIVE message, for if such a message were only sent once after the appearance of dissapearance of a touch, the packet with the message might get lost and the problem just mentioned could occur. The bundle may also contain some SET messages if they are called for. A bundle may also contain two other types of messages which are used for information about the bundle itself. It may have the optional SOURCE message which allows for applications to have several simultaneous input streams. In order to solve the problem of packages arriving out of order it will also have an FSEQ message, which indicates the order in which the bundles were sent.
The TUIO messages are structured as follows:

/tuio/2Dcur source application@address
/tuio/2Dcur alive s_0 ... s_N
/tuio/2Dcur set s_id x_pos y_pos x_velocity y_velocity m_accel
/tuio/2Dcur fseq f_id

While discussing the types of messages TUIO sends out it became clear which specific parameters should be used for each of the message types except in the case of the SET message. This is because different types of interaction with the surface might want to keep track of different types of information. Different types of interaction are distinguished from each other in the protocol by indicating the profile with which information is being sent. A profile is a string at the start of a message (above, this is /tuio/2Dcur) which notifies the receiver that the SET messages will have a certain set parameters (these have to be agreed upon up by convention).

It has been discussed that the spherical display should offer cartesian, spherical and normalised screen coordinates, depending on the needs of the application. Each of these sends somewhat different information and therefore needs its own TUIO profile. A closer look at TUIO profiles is therefore warranted.

5.2 TUIO profiles

A TUIO profile is used to indicate what kind of information is being communicated. Several default profiles are defined, though custom ones can be defined if necessary. Three types of default profiles are used: 2Dobj is used for describing uniquely identified objects on top of the surface, 2Dcur for cursors such as fingers and 2Dblb is an extension of the 2Dcur profile in which not just the centroid of the touch but also the bounding box around the object (size and orientation) is described. The default profiles for two-dimensional as well as three-dimensional surfaces are (with their parameters defined in Table 5.1):

2D Interactive Surface
/tuio/2Dobj set s i x y a X Y A m r
/tuio/2Dcur set s x y X Y m
/tuio/2Dblb set s x y a w h f X Y A m r

3D Interactive Surface
/tuio/3Dobj set s i x y z a b c X Y Z A B C m r
/tuio/3Dcur set s x y z X Y Z m
/tuio/3Dblb set s x y z a b c w h d v X Y Z A B C m r

Custom profiles are simply sent as such:
/tuio/\[formatString\]
For example:
/tuio/\sixyP set s i x y 0.57

Now the available profiles are known it is time to consider how to send the different types of information.

5.2.1 Screen coordinates

Communicating normalised screen coordinates is exactly what the 2Dcur profile already provides. It is notable that to properly support the profile the other parameters should also be calculated. If this is thought of as a waste computation time, a custom profile could easily be introduced which does not have these parameters (or the values of the default profile could be filled in with a constant value, but unless properly documented this could lead to confusion).

This profile will be used during the experiments. The development framework Kivy will be used for the testing applications. the framework allows for easy development of TUIO compatible
Table 5.1: The parameters used in the SET messages of the default TUIO profiles. Taken from the TUIO specification[26].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Session ID (temporary object ID)</td>
<td>int32</td>
</tr>
<tr>
<td>i</td>
<td>Class ID (e.g. marker ID)</td>
<td>int32</td>
</tr>
<tr>
<td>x, y, z</td>
<td>Position</td>
<td>float32, range 0 ... 1</td>
</tr>
<tr>
<td>a, b, c</td>
<td>Angle</td>
<td>float32, range 0 ... 2π</td>
</tr>
<tr>
<td>w, h, d</td>
<td>Dimension</td>
<td>float32, range 0 ... 1</td>
</tr>
<tr>
<td>f, v</td>
<td>Area, Volume</td>
<td>float32, range 0 ... 1</td>
</tr>
<tr>
<td>X, Y, Z</td>
<td>Velocity vector (motion speed and direction)</td>
<td>float32</td>
</tr>
<tr>
<td>A, B, C</td>
<td>Rotation velocity vector (rotation speed and direction)</td>
<td>float32</td>
</tr>
<tr>
<td>m</td>
<td>Motion acceleration</td>
<td>float32</td>
</tr>
<tr>
<td>r</td>
<td>Rotation acceleration</td>
<td>float32</td>
</tr>
<tr>
<td>p</td>
<td>Free parameter</td>
<td>type defined by OSC message header</td>
</tr>
</tbody>
</table>

applications over the 2Dcur profile. Kivy currently has not implemented all default TUIO profiles, but it is extended easily enough with custom profiles in its kivy.input.providers.tuio module. The default profile 2Dblb CCV1.4 uses is an example of one that Kivy does not understand, and so 2Dcur is used.

5.2.2 Cartesian coordinates

Sending cartesian coordinates of interactions is again straightforward using a default profile, the 3Dcur profile in this case.

5.2.3 Spherical coordinates

Unlike with the previous coordinate systems, there is no default profile for communicating spherical coordinates. The following simple profile is implemented, which uses both angles and the radius used in the coordinate system:

/tuio/spherical set s φ θ ρ

With the implementation now complete all the way from a physical setup to touch-detection to creating an interface for applications, it is finally time to evaluate the resulting system.
CHAPTER 6

Experiments

Finally it is time to test the applicability of the resulting implementation, and to do this four metrics will be introduced which will help to rate the performance of a touch display. In chapter 4 three methods have been outlined which enable the touch detection on a spherical display, and using the metrics introduced these will be compared. This means that they will be tested for completeness (detection works consistently everywhere on the screen), accuracy of the touch detection, the response time between a touch and the moment an application learns about this touch and the effort required of a user to set up/calibrate the spherical display. The system will also be tested for robustness against changes in the environment (such as in the lighting) and its updating frequency (how many per second the system can register changes).

Due to time constraints the experiments have not yet been performed. The results will be presented near the end of August as well as be available in written form soon.

6.1 Experiments

In order to measure how successful the implementation is that has been put forward it should first be considered by what properties this success should be measured, and then to experiment for these properties. The first measure is the accuracy of the touch detection performed, which allows for more meaningful interaction with the system. When measuring without a decent accuracy the result is a system which only has applications as a giant spherical button. The second property is partly an extension of this; the entire screen should preferably be usable with a uniform accuracy. This also makes the surface more intuitive to interact with. The third virtue a touch screen can have is the presence of a minimal response time between the moment a user interacts with the screen and the moment an application can respond to the interaction, as it is a real-time system. The fourth important property a system should strive for is that it requires minimal effort to set up or calibrate. Ideally the system is plug and play.

It is on these metrics that three approaches to touch detection from chapter 4 will be compared in order to find the most suitable one. There are however other important properties to the system still, which are general to all of the approaches. The system should aim to be robust enough to withstand different environmental conditions which might also change over time, as otherwise it is a somewhat impractical system that is also unsuitable for public display. Finally the updating frequency is also worth considering, this being the number of times per second a change in the touches can be found and communicated. If the system has a bad updating frequency it is possible a user’s interactions will not be detected, which could negatively affect the user’s experience with the screen.

Several simple experiments have been devised to test most of the metrics (the effort it takes to calibrate the system does not require an experiment), both for the complete implementation as well as for specific parts where applicable, so that the results can be compared to find out where improvement is possible. Before considering these experiments however it is worth specifying more information about the devices used in the implementation, as they influence the results.
After measuring these values it will be given a point of reference to systems actually in use by comparing it to the barycentric method as applied on a flat surface.

6.1.1 Physical specifications

The camera that is used is a Unibrain Fire-i Colour Firewire camera with a F2.5mm micro wide angled lens. It captures images at 30 frames per second at a resolution of 640x480. The projector is a XD500U-ST DLP ultra short-throw projector with a resolution of 1024x768. The computer used for processing the input runs Windows XP with 3.33GHz processor (Intel E8600), 3.25GB of RAM and an NVidia GeForce 9600 GT graphics card.

6.1.2 Accuracy and consistency

Two experiments will be outlined to test the accuracy an approach offers: one with the actual setup and one where all of the input is simulated to be near perfect, in order to separate the potential of the method from the user input (such as inaccurate touches on the surface) done. It will also help indicate how accurate the user needs to give input values.

A simple way to test the accuracy of the touch detection is to repeatedly project a point onto the screen, ask the user to touch it (to hold it actually, in order to make sure that accidental touches do not count) and to compare whether the detected touch location is at the same location as the location of the projected point. In order to get reproducible results a known pattern should be used to indicate which positions on the sphere should be touched, and the one used is shown in Figure 6.1. This pattern is chosen to test the consistency of the accuracy over the surface, and even to see whether some places can not be touched properly at all (perhaps due to insufficiently fixing the effects of the un-uniform lighting of the surface of the infrared light). The user is asked to touch all the points on the pattern several times, so that it can average the accuracy per point as well as for the entire sphere. The result will be a similar image but with the average deviation in normalised screen coordinates indicated near every point.

In order to do this with user input a simple touch application is necessary. This application is made using Kivy and interprets TUIO messages with the screen coordinate profile. This does mean the application does not know which part of the screen is cut off, and that the user has to indicate where to project a point himself (which is done by moving the mouse). In order to accommodate for this the points that have to be touched first have to be marked on the sphere such that the mouse can be moved to the proper position (such that the projected point is on the proper marker).

This experiment will be performed with each of the approaches as introduced, but as the accuracy of the user input (input such as touching indicated points on the surface or measuring the position of the camera) can play such a big role the potential and correctness of the methods will be tested separately.

This is done by considering a coordinate system representing reality, and to simulate the setup in this system with known locations for the projector, camera, sphere and touch inputs. There the sphere is positioned with its centre point at the origin, and it has a radius of 50. The touchable surface has a radius of 40. Due to the projection of the image falling outside of the scope of the project, the projection unto the surface will be assumed to have happened perfectly, from a projector at position (0, 0, -150). The camera is positioned at (5, 4, -150) and aimed along the point (2, 2, -40) in order to introduce a small distortion due to its perspective, which needs to be corrected by the approaches. In order to simulate the view of the camera a camera matrix is with known parameters is chosen which is consistent with the choices made above.
6.1.3 Response time

The next metric that will be measured is the response time (or latency) between the interaction of a the user on the screen and the response of a touch application. This will be done for individual parts in the pipeline to help show where improvement would be helpful. One rule of thumb is that the delay between an action and the feedback should be less than 100ms long. This advice may date all the way back to a set of guidelines proposed by Miller in 1968[27], which were based on his own experiences.

Wallclock time will be measured as this is more representative than for example CPU time of the actual delays that a user will encounter when the screen is in use. The wallclock time will be measured thousands of times and averaged for the pre-processing (preparing the image for blob detection), processing (finding and tracking blobs) and each of the approaches for finding the screen coordinates. As the barycentric method’s computational time depends on the number of reference points used its delay has been tested with several input values. After benchmarking implementations of the approaches as they were described there will be another experiment to see whether it is less resource intensive to instead create an array as large as the camera’s image, and to store per pixel coordinate to which spherical or screen coordinate this translates. This array has to be filled once, and once this is done any of the methods would take just as long to find the matching point for a new blob (at the cost of memory, a cost which increases along with the resolution of the camera(s) used. Note that although this may drastically improve the speed of the system if enough memory is available, it is also not at sub-pixel accuracy such as the centroids are actually detected. This could be compensated for a bit by drastically increase the array size in order to include 0.5 pixels.

To test the latency for the sending of TUIO messages as well as the duration of the entire application (for each of the three methods) another simple Kivy-based touch application will be used. The TUIO profile will be abused a little bit by every second sending the wallclock time from CCV where it should actually send a blob’s location. When testing the total duration the wallclock time is taken before CCV did any processing and for testing just the delay in sending messages the wallclock time is taken right before actually sending these messages. The measured time will include the time it takes the Kivy framework in particular to interpret its TUIO messages, which helps serve as an example of how long sending the messages will take in practice.

Though the delay incurred by the camera’s image detection is documented (30ms), this is not the case for the delay the projector introduces. This can be measured using a tool made by Muller for his thesis. To use it the refresh rate of the computer’s monitor should be set to 60Hz and the image should be mirrored on the projector. By projecting a bar that moves at a rate of 60 measurements per second and taking a picture (such as done for image 6.2), it is possible to determine how quick the projector was refreshing its screen by considering how many measurements its bar was behind the one on the monitor’s.

6.1.4 Update frequency

The amount of times the system can update per second depends firstly on the rate at which the camera can take its frames (which is 30 frames per second), secondly on the speed of the touch detection (if frames can not be processed at the rate of 30 frames per second a slower rate will have to be used) and finally on how quickly the touches can be communicated.

6.1.5 Robustness

The last metric that will be considered is how robust the system is against different environmental conditions as well as changes in those conditions. It has already been discussed that the system handles a source of infrared light on the background poorly. The testing for changes of the lighting will be done by calibration the system and comparing whether touches are still detected on the entire surface when the lighting changes by opening a window (not the accuracy is tested here but simply whether or not a touch is actually seen by the camera). Changes in the surroundings will be simulated by placing items nearer to the surface and again testing whether the performance of the touch detection changes anywhere on the screen.
Figure 6.2: Mirroring a moving image on both screens with the refresh rate known on one screen allows for finding out the rate of the other by measuring the delay.

With this, the implementation as well as the different approaches can now properly be assessed. The tests will also be performed using a flat screen with a barycentric approach in order to give the resulting implementation a point of reference to other systems.

6.1.6 Comparison detection on flat screen

Finally the latency and accuracy values will also be measured with the barycentric approach as applied to a flat screen.

6.2 Results

As not all methods have been properly implemented as of yet, the results will discuss the progress of the implementations as of right now. The focus will be on the correctness of the perspective and calibration approach (measured in the simulated coordinate system as mentioned before), as the barycentric one was largely already properly implemented before the start of the project. These experiments may yet be completed, and the results may be presented near the end of August.

6.2.1 Perspective approach

The current version of the perspective approach finds the $\phi$ angle accurate to 0.4 in most cases. The theta angle is estimated much more poorly, and is much more accurate on one side of the sphere than the other. This defect (which can go up to a 20cm miss on a sphere with a radius of 50cm) is the consequence of not yet compensating for the difference between the circle viewed being projected unto the centre plane and fact that this plane is not actually representative of the centre plane. The compensation for this has already been shortly addressed in the thesis.

6.2.2 Calibration approach

The calibration approach encountered a problem as the camera’s position has not properly been determined as of yet. It seems that as the camera matrix is estimated by the algorithm it can find different approximations of the situation, which results in different possible camera positions.
amongst other things. This problem should not be lethal to the approach, and will be addressed shortly.

A comparison of properties of the methods that do not require experimentation can be seen in Table 6.1.

Table 6.1: Comparison of methods for translating camera coordinates to screen coordinates. The values are measured in the hypothetical situation, meaning that all user input was assumed to be perfect. Overall accuracy does not include problem areas, which are areas where detection is drastically lower than other areas. ‘A fuller sphere’ refers to using a surface that is more than just a hemisphere, such as the Microsoft Sphere uses.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Barycentric</th>
<th>Perspective</th>
<th>Calibration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem areas</td>
<td>Edge of screen</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Calibration cost</td>
<td>Touch many points</td>
<td>Touch 4 points</td>
<td>[Touch 3 or more points](^1)</td>
</tr>
<tr>
<td></td>
<td>Positioning a grid</td>
<td>Positioning a grid</td>
<td>Placement of flat screen or permanently marked points</td>
</tr>
<tr>
<td>Underlying assumptions</td>
<td>None</td>
<td>Perfectly shaped surface</td>
<td>Perfectly shaped surfaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Touchable surface has a centre point from which the edge is always the same distance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accurately measured size of touchable screen and position of both camera</td>
<td></td>
</tr>
<tr>
<td>Applicable to a fuller sphere</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
CHAPTER 7

In conclusion

7.1 Discussion

Although no results are available, the results of early tests seem to be promising as to the successful touch detection on the surface. The only main problem with the touch detection seems to be a bad resistance to different lighting conditions. Future work could be done on remedying the weakness with regards to different lighting conditions, possibly with an approach based on storing one camera frame without anything touching the surface and one frame with the entire surface covered in a blanket (and thus reflecting back light).

Of the three methods discussed for interpreting the camera image, the calibration approach seems the most promising. This method seems accurate, unlike the barycentric approach, and less prone to errors thanks to bad user input than the perspective method.

An alternative method of approaching the touch detection altogether was briefly mentioned in chapter 2: to use two cameras looking at the surface from different perspectives. More research could be done to determine whether this would lead to better touch detection.

The interface of the system itself could also still be improved upon. Work could be done on robustly recognising gestures (regardless of how many fingers are used) and detecting the difference between inputs done by different users. Both of these areas already have research available to build upon.

7.2 Conclusion

In this thesis it has been discussed how a hemispherical touch-display could be made using a known (optical) multi-touch technique.

In order to achieve this implementation first the different techniques were compared, and it was decided that Rear Diffused Illumination was the only one with straightforward applications possible to a spherical display. The possible physical setups that were determined by the choice of RDI were subsequently discussed, and the version for the hemispherical display has been implemented.

It was then discussed how to accurately keep track of touches on the camera image by first using background subtraction and filters in order to find only blobs representing touches and then compensating for the camera’s inaccurate detection of these positions using Brown’s camera model. Once the positions of touches on the camera’s image were known three methods were introduced for translating these camera based coordinates to coordinates on the actual sphere, of which the calibration method seemed most promising.

Finally a method of communicating the detected interactions has been discussed.

The result of the work is a relatively cheap to build touch-display with good prospects of working. The differences between touch detection on a flat surface and a spherical surface have been discussed and dealt with. The only research objective left unanswered for now is how the solutions perform compared to a representative of optical techniques on a flat display. Though
further work is still needed for making the system more robust, the system has prospects of being perfectly usable.


## Image acknowledgements

<table>
<thead>
<tr>
<th>Image</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1, 4.4, 4.5</td>
<td>The total internal reflection, cartesian coordinates and spherical coordinates on <a href="http://en.wikipedia.org/">http://en.wikipedia.org/</a></td>
</tr>
<tr>
<td>2.2, 2.3, 2.5, 2.6 and 2.7a, 2.7b, 2.8a and 2.8b, 6.2</td>
<td><a href="http://www.sethsandler.com/multitouch/">http://www.sethsandler.com/multitouch/</a></td>
</tr>
<tr>
<td>2.10a</td>
<td><a href="http://www.touchscreenmagazine.nl/multitouch-techniques/optical-imaging">http://www.touchscreenmagazine.nl/multitouch-techniques/optical-imaging</a></td>
</tr>
<tr>
<td>2.10b</td>
<td>Flatfrog’s patent 7,442,914 B2 [28]</td>
</tr>
<tr>
<td>2.17, 2.19a and 2.19b</td>
<td>Microsoft Sphere’s publication[11]</td>
</tr>
<tr>
<td>4.7</td>
<td>Muller’s masters thesis[10]</td>
</tr>
<tr>
<td>4.9</td>
<td><a href="http://www.aurigma.com/docs/gm/Transformations.htm">http://www.aurigma.com/docs/gm/Transformations.htm</a></td>
</tr>
</tbody>
</table>
Derivation mapping coordinate from image to plane

\[sx = r_{11}X + r_{12}Y + t_1 \quad (A.1)\]
\[sy = r_{21}X + r_{22}Y + t_2 \quad (A.2)\]
\[s = r_{31}X + r_{32}Y + t_2 \quad (A.3)\]

where \(s\), \(X\) and \(Y\) are the only unknown values

\[\frac{r_{21}X}{r_{11}X} = \alpha \quad (A.4)\]
\[r_{21}X - \alpha r_{11}X = 0 \quad (A.5)\]
\[sy - \alpha sx = r_{22}Y - \alpha r_{12}Y + t_2 - \alpha t_1 \quad (A.6)\]

\[\frac{r_{32}Y}{r_{22}Y - \alpha r_{12}Y} = \beta \quad (A.7)\]
\[r_{32}Y - \beta (r_{22}Y - \alpha r_{12}Y) = 0 \quad (A.8)\]
\[s - \beta (sy - \alpha sx) = t_3 - \beta t_2 + \beta \alpha t_1 \quad (A.9)\]
\[s(1 - \beta y + \beta ax) = t_3 - \beta t_2 + \beta \alpha t_1 \quad (A.10)\]
\[s = \frac{t_3 - \beta t_2 + \beta \alpha t_1}{1 - \beta y + \beta ax} \quad (A.11)\]

\[(A.6) : sy - \alpha sx = r_{22}Y - \alpha r_{12}Y + t_2 - \alpha t_1 \quad (A.12)\]
\[Y = \frac{t_2 - \alpha t_1 - sy + \alpha sx}{r_{22} - \alpha r_{12}} \quad (A.13)\]

\[(A.1) : sx = r_{11}X + r_{12}Y + t_1 \quad (A.14)\]
\[X = \frac{r_{12}Y + t_1 - sx}{r_{11}} \quad (A.15)\]