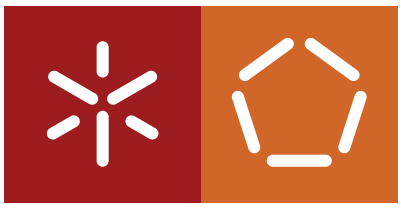


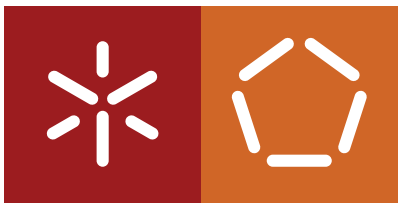
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Gonçalves da Silva

Qualquer Objeto é um Controlador de PC

Any Object is a PC Controller

January 2016



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Gonçalves da Silva

Qualquer Objeto é um Controlador de PC

Any Object is a PC Controller

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
Professor António Ramires Fernandes
Professor Pedro Miguel Moreira

January 2016

ACKNOWLEDGEMENTS

I would like to extend my gratitude to my supervisors, Professor António Ramires Fernandes and Professor Pedro Miguel Moreira, for all their help, guidance and above all, patience throughout the realization of this project.

To the Center for Computer Graphics, who initially proposed this project and provided a workspace and equipment. Special thanks to Nelson Alves and the entire team of CVIG for all their help.

To all my friends and colleagues, for all the help and company they readily provided.

And finally, to my family, for all the love, understanding and unconditional support they gave me on this journey, and throughout all the years of my life.

ABSTRACT

Computer technology continuously evolves at an incredible pace, with today's machines being barely recognizable when compared to the first computers ever created. Despite this, and in spite of the many advances in human-machine interactions, the main method for interfacing with computers has remained the same for several decades.

Tangible User Interfaces (TUIs) are an interesting alternative to the classical Graphical User Interfaces (GUIs) that dominate the field. This type of interface seeks to bridge the gap between the physical and virtual worlds by binding virtual data to physical objects. This allows users to directly manipulate digital information with more natural and intuitive gestures, simply by interacting with the corresponding objects. This concept provides a wide world of possibilities to create more engaging and immersive interfaces capable of greatly improving user experience.

Many implementations of TUIs were made, but they are usually very focused on specific applications, and many require hardware that is unique to them, making it expensive or hard to acquire.

The work herein presented tackles this problem and proposes the creation of a framework, capable of enabling the activation of everyday objects as controllers for diverse applications. The framework should make use of an accessible sensor such as a camera, to detect objects and assign them tasks. Direct manipulation of objects by the users should generate an event. Events should be able to be communicated in a clear way to client applications connected to the system.

Since the framework will serve for interface purposes, a key point is to be responsive, providing immediate reactions to user actions. It should also be simple to setup and use, and should allow for easy interactions between users and objects.

The framework should be as flexible as possible. Consolidating all the requirements, and balancing factors such as quality and computational costs, posed difficult challenges were decisions needed to be made and options discarded. Despite all the difficulty however, the final solution still managed to achieve acceptable results. It can track in real-time 3D translations and 2D rotations of multiple objects in a scene, posing some limitations only on the type of objects to be used and the position of the sensor. It is robust against occlusion and can handle multiple simultaneous users.

A demo application was also developed to help demonstrate some of the practical applications for this system.

RESUMO

A tecnologia de computadores evolui continuamente a um ritmo incrível, com as máquinas de hoje a serem quase irreconhecíveis quando comparadas com os primeiros computadores inventados. Apesar disso, e apesar dos muitos avanços em interações homem-máquina, o principal método de interagir com computadores manteve-se o mesmo durante várias décadas.

Interfaces Tangíveis são uma alternativa interessante às clássicas interfaces gráficas que dominam a área. Este estilo de interface visa unir o mundo físico e o mundo virtual através da criação de ligações entre dados virtuais e objetos físicos. Isto permite que utilizadores manipulem diretamente informação digital com gestos mais naturais e intuitivos, através da simples interação com os objetos correspondentes. Este conceito proporciona um vasto mundo de possibilidades para a criação de interfaces mais atraentes e imersivas, capazes de melhorar significativamente a experiência dos utilizadores.

Existem várias implementações de interfaces tangíveis, mas geralmente todas se focam em aplicações muito específicas, muitas vezes exigindo equipamento próprio, tornando-o caro ou difícil de adquirir. O trabalho aqui apresentado aborda este problema e propoõe a criação de uma estrutura capaz de permitir a ativação de objetos do cotidiano como controladores para aplicações diversas. O sistema deve utilizar um sensor acessível, tal como uma câmera, para detetar objetos e atribuir-lhes tarefas. A manipulação direta de objetos pelos utilizadores deve gerar eventos, que devem ser capazes de ser transmitidos de forma clara para qualquer aplicação-cliente ligada ao sistema.

Uma vez que o sistema vai servir para porpositos de interface, um ponto chave é responder rapidamente aos comandos do utilizador. Deverá também ser simples de usar e configurar, e deve permitir que o utilizador seja capaz de manipular os objetos de forma simples e confortável.

O sitema deve ser tão flexível como possivel. Consolidar todos os requisitos, e equilibrar fatores como qualidade e custo computacional proporcionou desafios complexos, que levaram a escolhas difíceis. No entanto, e apesar das dificuldades, a solução final conseguiu atingir resultados aceitáveis. O sistema consegue seguir em tempo real o deslocamento no espaço 3D e rotação no espaço 2D de múltiplos objetos em cena, impondo algumas limitações somente no tipo de objetos que possam ser usados e na posição do sensor. O sistema é robusto face a oclusões e consegue lidar com múltiplos utilizadores em simultâneo.

Uma aplicação para demonstrar as aplicações do sistema também foi desenvolvida.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES

1

INTRODUCTION

1.1 CONTEXT

The creation of the computer is a landmark in mankind's history. With innumerous applications in all fields, from entertainment to science, the computer quickly spread throughout the world. These days, we can find computers in all manners of shapes and sizes nearly in every household or industry of all developed countries. Computer's technology has also gone through huge improvements, as what started as huge and bulky machines can now be found in many small day-to-day use devices, like cell phones or laptop computers. Despite this, however, the means for humans to interact with computers as always been somewhat abstract and impersonal, sometimes even overly complicated. While in the early stages of computer development this was necessary due to hardware limitations, nowadays that limitation has since long been relaxed. Even though a great deal of research has been done on the field, the most used User Interface (UI) is still the GUI based on the Windows, Icons, Menus, Pointer (WIMP) paradigm, similar to what is seen on Microsoft Windows and Apple Os X, and making use of the same principles that have been around since the early 70's (Beaudouin-Lafon, 2004).

The reason WIMP interfaces are so popular is due to their ability to abstract actions and data, which makes it easy to represent on a monitor screen and simple to learn and use. They create the "desktop metaphor" where the screen is seen as a desk and the windows represent documents. A keyboard and mouse serve to receive user input, moving files and objects around the desk, selecting commands from menus or writing down text. This removes the need to memorize commands and shortens the learning curve, although a certain degree of memorization is still needed for a more advanced use of the interface, should the user feel the need to improve his efficiency. Unfortunately, it is this very ability to abstract that makes this type of interfaces so unsuitable to several other situations. As an example, it constricts the range of possible actions that can be taken to interact with a virtual object and limits the contact of a single user per

computer at a time, both aspects that are very important in certain applications like games and design programs.

Nowadays, developers are more focused in creating interactive systems that create responses in users, drawing them in and making them feel secure and motivated in their actions, in opposition to the old style of creating efficient systems without much care for user convenience. Good interfaces should be intuitive and easy to understand, allowing users to comprehend the interfaces' metaphors, in order to solve tasks in an efficient and confident manner, reducing user frustration and increasing productivity, a job that is not easy to accomplish. Usually this is achieved by trying to mimic the natural way humans interact with object in their day to day with the use of several metaphors, but what if we removed those metaphors and made use of those interactions and object themselves?

1.2 MOTIVATION

TUI are an interesting alternative way of communicating with computers. This new paradigm allows users to interact with virtual objects and information through the manipulation of objects at the physical environment. This new style of interface allows for several advantages over traditional UIs, such as permitting an object to be handled with both hands, or the manipulation of multiple objects at the same time by multiple simultaneous users, all in a 3D space and taking advantage of the movements that are natural and instinctive to humans. These factors could improve motivation, interest and comfort, while making it easier for users to learn and adapt to the controls (Ishii and Ullmer, 1997)(Posada et al., 2014). As an example of advantages of tangible devices, we need only to look at the mouse. It is much easier to move a pointer using natural movements with the mouse than it is to use the keyboard's directional arrows or even textual commands.

A large amount of research has been made in this area, with several applications coming forward that use TUIs for great effect in some areas. Nonetheless, these applications either require specialized equipment, are very particular in their function or both, with most practical examples simple associating physical object with digital geometric objects. There isn't a simple and practical framework that can be used for multiple purposes and applications, with easily obtained equipment and straightforward configuration. This can contribute to keep developers away from this type of interfaces due to technical and monetary difficulties (Cottam and Wray, 2009). A workaround to the need of specialized equipment that would be interesting to explore would be by making use of a technology that has, in the last few years, become quite widespread,

namely speaking, the camera. Cameras can nowadays be found everywhere, from cell phones and laptops, to streets and public buildings, with diverse qualities and serving numerous purposes, cameras are cheap and easily accessible. One particular type of camera of interest would be the depth sensing camera. While a relatively recent technology, it has already produced several devices of diverse qualities, and has even begun spreading to mobile devices (Simonite, 2013).

The depth sensing camera assigns a measure of depth in a per-pixel basis along with color data, thus mapping the surface of a 3D area. Depth information can be used to detect the tangible items and process their movement as information. Reasonably priced, and easy to obtain and use, this camera could substitute the need for more complex equipment in a TUI using application. With this, only the choice of items to use as tangible devices remains. While some applications make use of complicated devices or Augmented Reality markers, this question lingers: Why not use simple everyday items as controllers?

1.3 OBJECTIVES

The main goal of this dissertation is the creation of a modular framework capable of enabling the activation of common daily use objects as input controllers, allowing the establishment of practical TUIs for a variety of applications. It is important for the system to be simple to install and operate in order to provide a more interesting and smooth user experience.

A RGB-D camera was chosen to serve as the only sensor for this project, and computer vision algorithms will be used to detect actions performed upon the objects. Accounting for all these facts, several conditions were identified as necessary for the framework to act as intended. Structurally, the framework should embody the following characteristics:

The framework:

- Should use readily available and reasonably priced equipment.
- Should be simple to setup and use.
- Should be fast to respond to user input.
- Should be able to accept a great variety of objects.
- Should be able to communicate events to client applications.

These five points are the groundwork that the system should be built upon. Functionally, it is expected that the framework:

- Is able to detect and register unknown objects to track.
- Is able to attribute tasks to objects, such as making them acts as dials or sliders.
- Is able to detect events applied to the objects, such as translation or rotation.
- Is able to achieve robust results under controlled environments.

Consolidating all these facts is not an easy task, and decisions need to be made in order to create a balance between quality, speed and user comfort. Robust algorithms tend to either be computationally expensive or rely on complex models, built with prior knowledge of the used objects. On the other hand, more lightweight methods can fail when faced with issues such as object occlusion or variable environment conditions. The rest of this dissertation will focus on attempting to solve these problems and develop a working prototype that can meet the necessary requisites.

The quality of the implemented solution will be analyzed and evaluated in terms of obtained results. A demo application will also be designed to exemplify the capabilities of the framework.

1.4 DOCUMENT STRUCTURE

This document is organized in the following way: the first chapter introduces the main topic of research, describing its context, motivation and the objectives of this dissertation. The second chapter will present the state of the art in Tangible User Interfaces, as well as methods used to describe, detect, recognize and track objects.

The third chapter describes the goal to achieve and challenges to surpass, followed by a proposal for an architecture and an analysis of the methods and techniques required for an implementation.

The fourth chapter presents and analyses the obtained results, discussing strong and weak points of the system. It also presents a proof of concept by means of a demo application. The final chapter presents main conclusions and final thoughts as well as interesting directions for future work.

2

STATE OF THE ART

2.1 TANGIBLE USER INTERFACES

In the early days of computing, many sacrifices had to be made in return for computing power, making user comfort and convenience rank very low in the list of priorities. User interfaces were incredibly primitive and hard to use, requiring a great deal of work and patience from users to execute even the most simple of tasks. The first interfaces, known as Batch Interfaces, used special machines to write commands in the form of punched cards or paper tape, which was then fed to the computers, which, in turn, printed back the output in the same cards. It was a very slow and unforgiving process, where the lack of any real time interaction turned even small errors in massive losses of time and resources (Raymond and Landley, 2004).

Command-line Interfaces made an appearance in the late 60's, and were a colossal improvement over Batch Interfaces, requiring the users to simply write down the commands directly in the computers with the aid of a keyboard. With a video display that showed the output of the programs, this was a combination that increased the interactivity to near real-time, producing a massive increase in productivity and facilitating immensely the tasks of the users. While still a very flawed interface, being strictly abstract and requiring a lot of memorization, it's an interface that can still be found available today in most computers today, namely for development environments(Raymond and Landley, 2004).

Graphic User Interfaces made an appearance around the same time and went through several changes over the years. The most common were based on the WIMP paradigm created in the early 70's by Xerox researchers. Popularized by Apple Macintosh in the early 80's, this type of interfaces is still today's most used system for human-machine interaction (Raymond and Landley, 2004; Reimer, 2005).

There were made several attempts over the years to improve the GUIs that we use today, but only a few actually made it into popular use. Distinguishable amongst those, we have the mouse

and the recent touch screen technology that has become very popular with mobile devices. Both of those technologies, that brought many improvements to human-machine interactions, have something in common that made them very well liked by users everywhere: they are tangible devices that give users a more natural and direct way of controlling their actions in a computer. By moving the mouse with their hands, the pointer on the screen will move accordingly; by touching an icon, an action will result; by dragging something, users can organize their desktop. It's this simple, natural and straightforward approach to User Interfaces that is the core of Tangible User Interfaces (Kim and Maher, 2008). Figure 1 illustrates the differences between TUIs and the classical UIs.

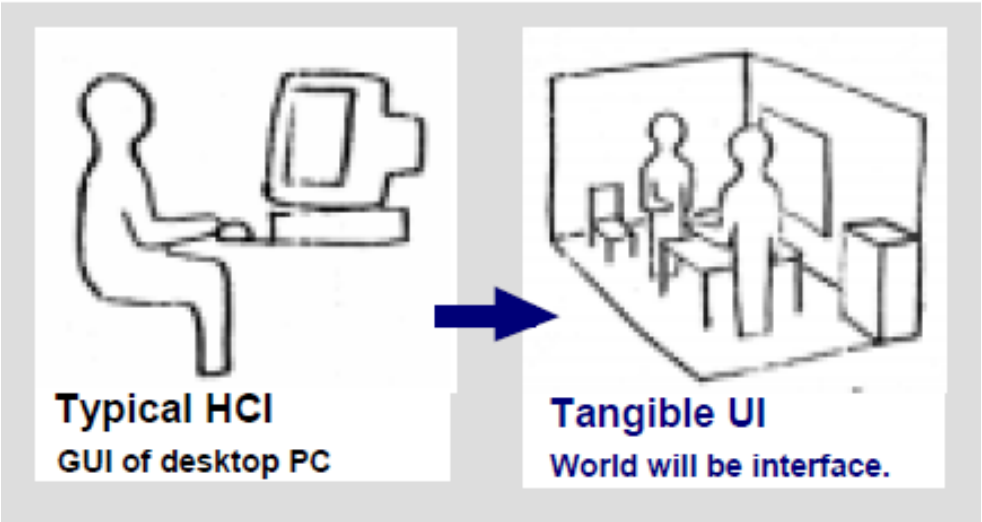


Figure 1: GUI vs TUI (Ishii and Ullmer, 1997)

Initially called Graspable User Interfaces (Fitzmaurice et al., 1995), they were later renamed Tangible User Interfaces by Hiroshi Ishii and Brygg Ulmer, important pioneers in this field (Ishii and Ullmer, 1997). Their vision for TUIs was to bridge the gaps between virtual and physical environments, coupling bits to everyday objects and allowing users to directly manipulate foreground data while still being aware of background information through the use of sound, light or others physical signs. They introduce three key concepts to allow such a system to work: interactive surfaces, coupling of bits and atoms, and ambient media.

Interactive surfaces consist in turning any available surface within a work area, like walls, desktops or floors, into active interfaces that users could make use of. Coupling bits and atoms coincides with turning common objects like books and cups into devices that manipulate virtual information. Lastly, the concept of ambient media would be to make use of media effects to

transmit information of background data to the periphery of the user's senses. This combination of concepts could potentially turn the entire work space into a user interface, where users could interact with desired data while still being aware of all other important background information (Figure 2).

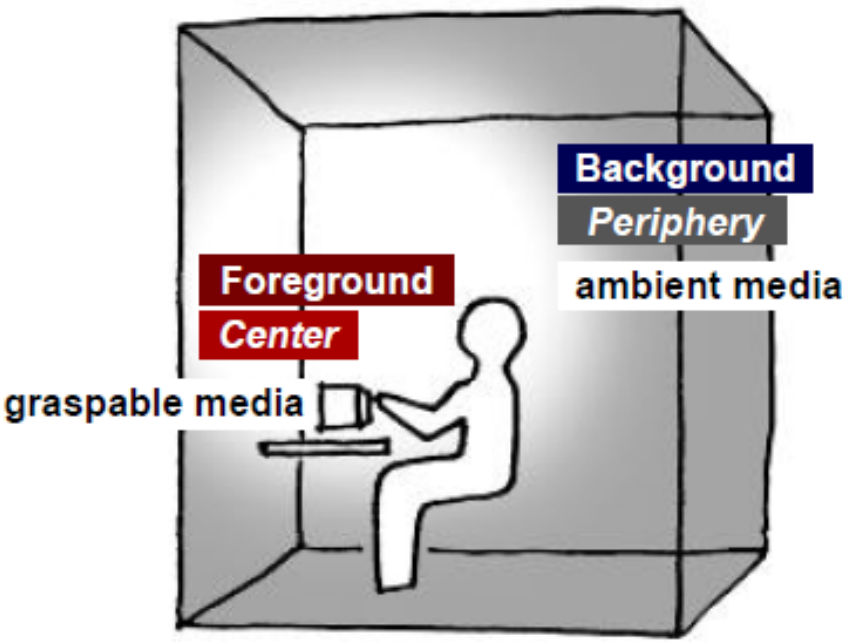


Figure 2: Diagram of user's attention within the work space (Ishii and Ullmer, 1997)

Three prototypes were proposed to explore these concepts, with *metaDESK* and *transBOARD* centered on foreground of the users attention, and *ambientROOM* more focused on the background interactions. The *metaDESK* design is a system that tries to recreate the desktop metaphor of GUIs in a physical instance (Ullmer and Ishii, 1997). The standard WIMP components like windows and icons are transformed into lens and physical icons or "phicons", which are then connected to the system through a group of optical, mechanical and electromagnetic field sensors. A practical example of this system would be the *Tangible Geospace* prototype. This application makes use of tangible models of building or other landmarks to manipulate Two Dimensional (2D) or Three Dimensional (3D) graphical maps. The models serve as restriction points for the map and moving or rotating them on the desk will move and scale the map accordingly. Other devices can also be used to perform several tasks, like an arm mounted monitor to navigate and visualize the 3D representation of the space, or a small lens to manipulate image overlays (Figure 3).

Figure 3: Tangible Geospace (Ullmer and Ishii, 1997)

While the *metaDESK* explores the concept of ”coupling bits and atoms”, the *transBOARD* studies the notion of interactive surfaces. It makes use of the *Microfield Graphics SoftBoard*, a seemingly normal white board that uses infrared lasers to scan the activity of tagged pens and erasers (Ishii and Ullmer, 1997). Magnetic tagged paper cards act as physical icons in this system, and can be attached to the board to store anything written in it. These cards can then easily be moved to different computer screens to display the drawn information. In addition, real-time monitoring of the board's activity can communicate information through the Internet to various clients, with the use of a Java applet (Figure 4).

While both examples deal in an interesting way with the direct interactions between the user and its chosen task, or the foreground information, both experiences can be enhanced with the addition of information of background interactions. The *ambientROOM* can, through the use of ambient media like sounds and lights, complement the use of the previous interfaces by communicating information at the periphery of the user's senses (Ishii et al., 1998). The idea is that while we perform certain tasks, like talking to a coworker or writing down a report, not only we are getting information from what we are doing at the moment but we also absorb other types of parallel information, like the current time of day from the amount of daylight available or the current weather state by the sound of rain.

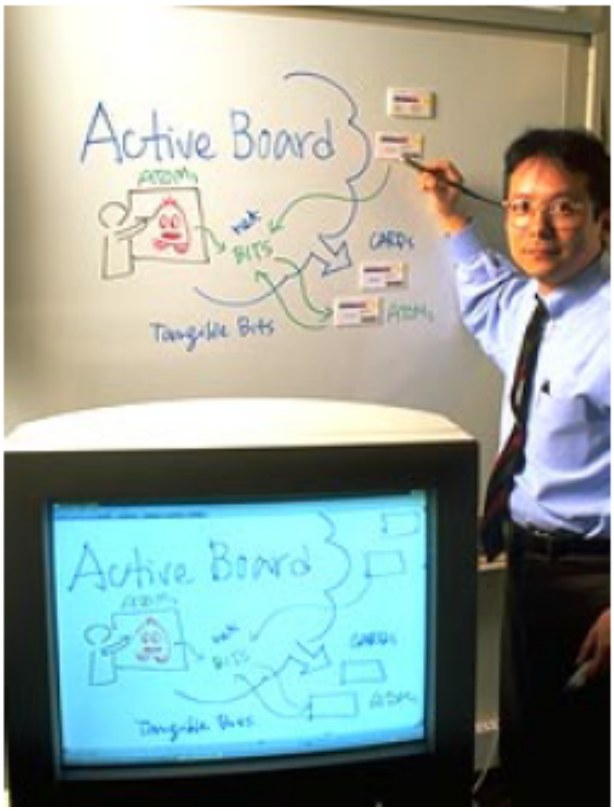


Figure 4: The transBOARD system (Ishii and Ullmer, 1997)

Designed to take advantage of these subconscious clues, the *ambientROOM* resembles *Steelcase*'s Personal Harbor workroom with added MIDI-controllable facilities, and is both compact in size and comfortable to work in. A simple use case scenario for this system would be to associate the sound of rain drops to the number of hits on a website. The steady stream of rain would remain ignored by the user as a normal amount of visitors to the site, while a more noticeable change to heavy rain or even a lack of rain would be noticed by the user and indicate that the website is having either a sudden increase or lack of visitors. This function could, for example, be turned on or off by moving a physical object near to, or away from, another device (Figure 5).

As stated before, TUI is a field of study with a lot of interest, and as such, there are a great many number of TUI applications that have been developed over the years. The destined domains of application are also greatly numbered, with examples of applications being found in areas as varied as information management, simulation, modeling, education, entertainment, collaborative work, and many others.

Figure 5: The ambientROOM (Ishii et al., 1998)

A noticeable example would be the *URP* system to urban planning (Ben-Joseph et al., 2001). This application makes use of augmented reality to project a map of wind trajectories, solar shadows, traffic and other matters of interest, that react accordingly to physical markers of buildings. Aspects like position of the sun or direction of the wind can also be adjusted with physical tools, like moving the pointers of a clock tool. Other physical tools can also be used to apply different materials to buildings, like making a building out of glass and causing it to cast not only shadows but also reflections. A system like this simplifies urban planning immensely, by allowing architects to simulate building styles and configurations, while taking in consideration all these important factors, and can also be a very useful tool for academic use (Figure 6).

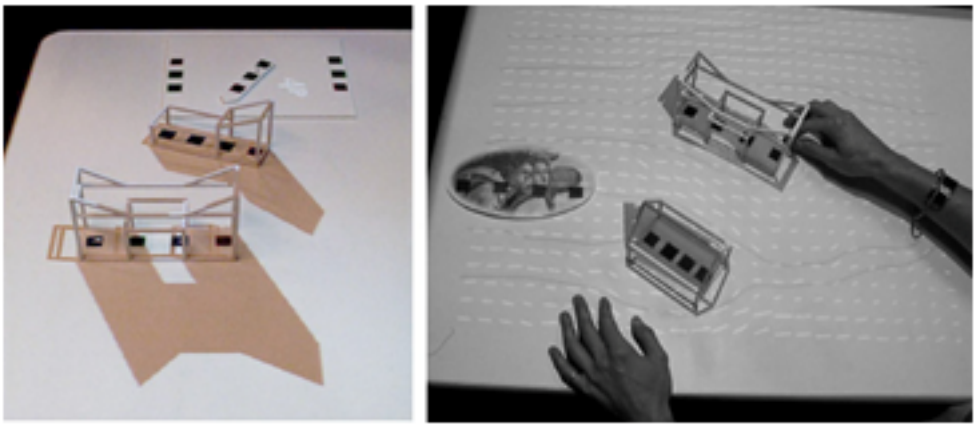


Figure 6: URP (Ben-Joseph et al., 2001)

A similar application would be the *Illuminating Clay* interface, an application created for landscape analysis (Piper et al., 2002). In this system, an interactive clay model can be modeled to alter the topography of landscapes. This model is analyzed with a laser scanner to generate a depth map which can then be fed to a library of landscape analysis functions to generate desired results. The output can be projected over the model to generate real time visualization of changes on the geography. Several other outputs, such as results from other functions or cross-sections of the model, can also be simultaneously projected over the workspace surface to compare results. This system can be used to investigate such thing as landscape erosion, water flows or solar shadow projections (Figure 7).

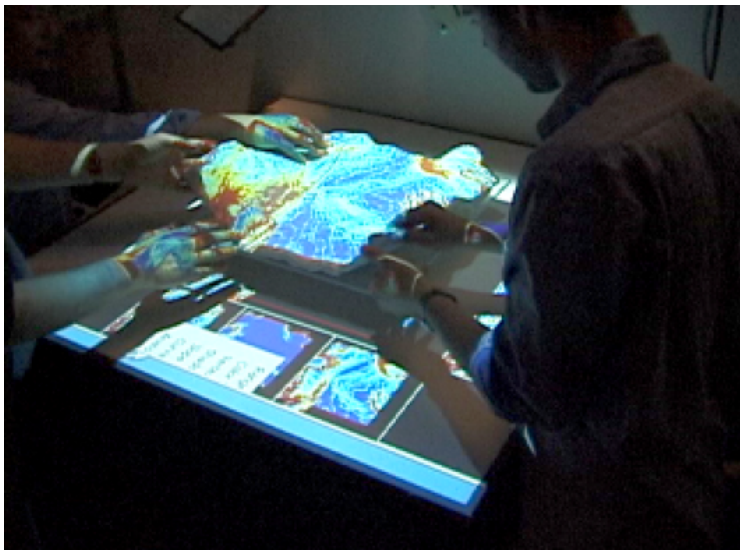


Figure 7: Illuminating Clay (Piper et al., 2002)

Another example would be the *mediaBlocks* system (Ullmer et al., 1998). This tangible interface allows users to easily manipulate groups of online media objects, like videos or images. It consists of a group of small tagged blocks that can serve as containers for lists of online media. By fitting media devices such as projectors or cameras with special slots where the blocks can be inserted, we can, for example, record a video into an online space and bind it to the block. The block can then be easily moved and inserted into a projector or monitor, which will playback the associated video. Inserting blocks in a computer permits users to simply exchange the content of the blocks with basic GUI drag-and-drop operations. A special device can be used to load several blocks at the same time, facilitating the binding of the content of several adjacent blocks to a new block. A second device can also bind the position of a block to its contents, allowing the user to browse the media within the block by moving it left and right (Figure 8).

Figure 8: mediaBlocks (Ullmer et al., 1998)

Tiles is another TUI application (Poupyrev et al., 2001). It makes use of Augmented Reality (AR) to overlay virtual object on marked magnetic tiles, which can be seen using head-mounted displays. A metal whiteboard is used as a workspace to hold the tiles and users can rearrange their position as they wish. A book is used as a menu containing virtual objects, which users can assign to tiles. Tiles can contain operators like store or delete, or data. Moving an operator tile next to a data tile can execute that operation on the respective data (Figure 9).

A setup similar to *Tiles* is used to implement an AR game system (Ulbricht and Schmalstieg, 2003). A simple glass table with a camera bellow is used to detect markers while avoiding occlusions caused by the players. Each marker can represent a game object, like a catapult or a windmill. The whole table is the game space that can hold virtual objects. Players can control the objects represented by the markers but not virtual objects. Virtual objects on the other hand, can be affected by the presence or position of physical objects. In the example game, each player held a catapult, a windmill that they could control, and a set of balloons, represented by virtual objects and thus uncontrollable. The goal of the game was to pop the opponent's balloons while protecting your own. Since players could not affect the balloons directly, they had to use the catapults to destroy them or the windmills to move them. Objects represented by markers could also affect each other. For example, the air current produced by a windmill could be influenced by the current produced by another windmill. A system like this could be used to create interesting and engaging games (Figure 10).

A theme that has gathered interest in the area of TUI is music and many applications have been developed in that area. The *musicBottles* TUI turns bottles into containers for sound and music

Figure 9: Tiles (Poupyrev et al., 2001)

(Ishii, 2004). Tagged and corked bottles ”contain” music played by different instruments. When the bottles are placed on top of a special table, a soft light will turn on and light up the bottle. When the bottle is uncorked, the light's color and intensity will change dynamically according to the bottle, and the music represented (or ”contained”) by the bottle will play. When the cork is placed on the bottle again, the music will stop and the light's color will return to normal. If the bottle is taken of the table, the light will turn off (Figure 11).

Following the theme of music, a very important mark in the history of TUI is a system that was developed as an electronic music instrument. The *reacTable* is an application that received many awards and gathered a lot of interest for the field of tangible interfaces. Presented as a round table, it lights up with colors and visual effects when in use. Special physical icons are recognized by the table, representing different instruments and sound effects. Different object can be grouped to create distinct effect combinations, and their relative position to each other affects the resultant sound. Visual effects are displayed beneath the objects representing sound frequency and volume, as well as their association between each other. Rotating objects can change the frequency of the sound they produce, and the volume can be changed by touching and dragging fingers on the volume slider displayed beneath the object. A more recent version

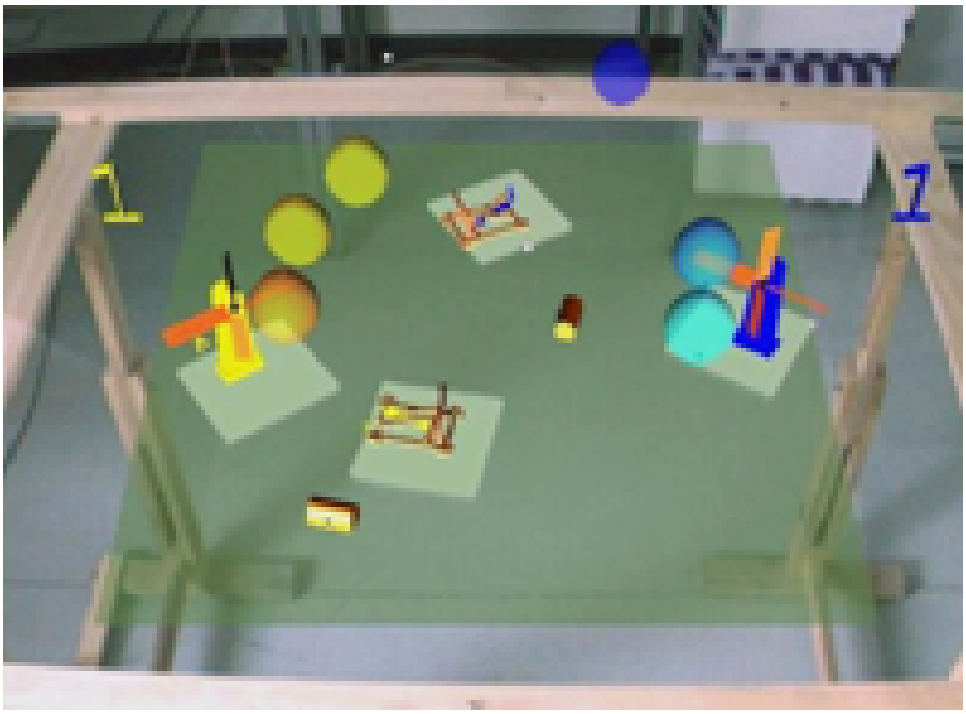


Figure 10: The Tabletop Game (Ulbricht and Schmalstieg, 2003)

Figure 11: musicBottles (Ishii, 2004)

of this system is available for *iOS* technology, dismissing the table altogether (Kaltenbranner

et al., 2006). The tracking technology behind the *reacTable* was released as the open source *reacTIVision*, attracting the interest of several developers, and further increasing the interest and awareness for tangible user interfaces (Figure 12).

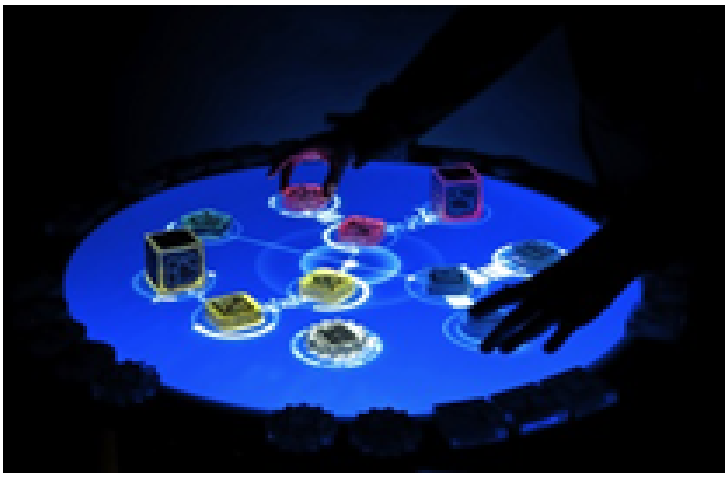


Figure 12: The reacTable (Kaltenbranner et al., 2006)

Microsoft has also entered the TUI race with *Microsoft Pixelsense* (previously known as *Microsoft Surface*). It is an interactive surface platform that allows direct interaction, multi-touch contact, multi-user experience and object recognition (Microsoft, 2012). The user or users can interact with the interface using multiple contact points simultaneously, and can use tagged object, digital or not, as input devices. For example, users can use a normal brush to paint a digital picture, or transfer pictures in and out of a cell phone just by placing it on the surface (Figure 13).

The field of tangible interfaces may still be growing, but already has a great number of applications in existence. Despite this, most applications are designed for specific tasks, and usually require specialized, hard-to-acquire equipment. Because of this, the prototype presented by *Ayotle* and *Digital Labs* is of great interest (Ayotle and Labs, 2013). Designated *Anytouch*, this application makes use of a depth sensing camera and a computer vision software designed by *Aytole* to turn any object, independent of size or surface, into a tactile device (Figure 14).

2.1.1 *Categories of Tangible User Interfaces*

Despite the significant number and variety of TUI in existence, most of them can be grouped in only four distinct categories: spatial, constructive, relational and associative. Spatial systems are

Figure 13: Microsoft Pixelsense (Microsoft, 2012)

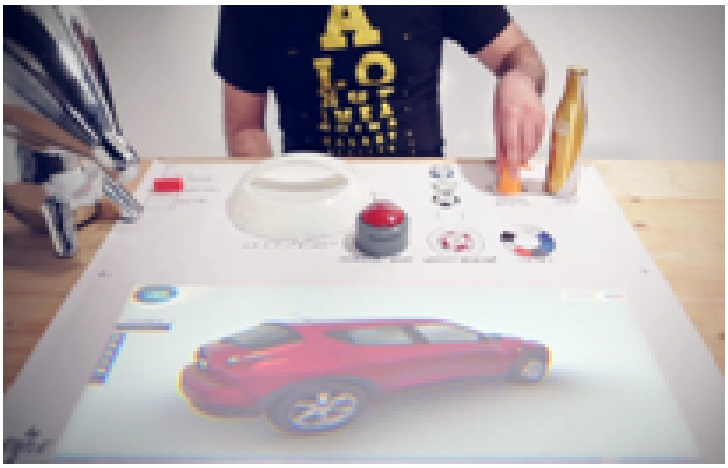


Figure 14: Anytouch (Ayotle and Labs, 2013)

tangible interfaces that take into consideration the spatial positioning and orientation of multiple physical objects, usually on a horizontal surface. Examples of these systems are *metaDesk* and *URP*. Constructive TUIs, like the *transBoard* or *Tiles* applications, use modular blocks to build computationally interpreted physical systems. Relational systems give computational interpretations to logical relationships, like adjacency or sequence. A good example of a relational TUI is *mediaBlocks*. Finally, associative systems associate individual physical objects to digital data without creating relationships between physical objects. An example of a system like this would be *musicBottles*. It is possible for an interface to fall in more than one of these categories. In fact,

interesting interfaces should try to combine different strategies in their implementation (Ullmer and Ishii, 2000).

Concluding, Tangible User Interfaces present an interesting alternative to the classical interfaces available. They provide important elements for human-machine interaction such as direct manipulation of data through use of natural and intuitive motions, easy cooperation between users on the same or different tasks, perceptive representation of results and are fun and motivating to use. A great deal of study has been made in the field and yet much room for growth remains, promising to deliver a lot more interesting applications in the future. Sadly, TUI systems are somewhat lacking in the market, and most of what is available makes use of high priced or hard to acquire equipment. This denotes a clear need for a framework that can bridge the gap between users and applications in a simple and inexpensive manner.

2.2 OBJECT DETECTION

In order to use objects as controllers we need to detect them and track their movements. There is a huge myriad of methods to achieve this, using many different types of sensors and equipment. For this project, it was decided to use a camera as the sensor, as it is a piece of equipment that is cheap and easy to obtain and use. It will allow full perception of the scene and objects with minimal and non-intrusive or complicated equipment. They are also small, lightweight and mobile, allowing the system to be moved and configured quickly and simply. Depth sensing cameras were chosen over normal cameras for added depth information, which can be useful for solving several problems inherent to the topic.

2.2.1 *Object Features*

While using a camera to detect objects, we enter the field of computer vision, and object detection and tracking is an area of great importance in this field. Despite this, and despite all the research that has been made to this day, it is still a great challenge to produce quality results with systems that are both practical, intelligent, and work at real-time. There are a myriad of problems that can interfere with the detection, from cluttered background, occlusions, deformable objects, light conditions, and many others. Usually these problems are solved on a case to case basis, with different approaches being used for different systems. This typically results in great successes in controlled environments but poor results in uncontrolled ones (Prasad, 2012). Because of this,

great care is needed when developing a system of this type, with as many variables as possible needing to be taken into consideration.

Many algorithms were proposed for object detection, with some being better at dealing with certain difficulties than others. They tend to differ not only on the approach they employ for searching, but also on the features they try to detect. Selecting the right features is critical for obtaining good results, as some features can offer good performance in one area while suffering in another. Due to this, many algorithms actually make use of multiple types of features to circumvent the problem. The most common types of features can be grouped into three categories: color, edges and texture (Yilmaz et al., 2006). The color of an object is influenced by both the properties of the light sources and the object's surface. When white light, which is a mixture of all the visible colors of the spectrum, hits an object, we will see the color(s) that the object's surface reflects better. As an example, a red object absorbs all other specters of color that are not red, reflecting the red light to our eyes.

In a digital sense, color is represented as a group of numbers, whose meaning is dictated by a color model. The *RGB* color model is one of the most common found in images, and consists of three numerical components per pixel, each number representing the color intensity for red, green and blue on that pixel. Color features are somewhat vulnerable to shadows and light intensity variations, so a lot of research has been made to try and find color models that more robust to those changes. The *HSV* color model is one that is commonly used for image processing, with each of its three components representing the hue, saturation and value of a pixel. This model separates the image intensity from the color information, allowing for more accurate feature extraction (Chen et al., 2007). Many other models exist but, like when choosing the type of features to use, choosing a color model must be done on a problem to problem basis, as there is still not one that can claim to be the most efficient for all image processing tasks.

A simple way to represent the color distribution of an object in any color space is through an histogram. Histograms can be constructed in different ways and may have distinct meanings, for example, a grey-scale histogram can represent the distribution of pixel intensities. An histogram has a set of bins, that represent a color value or group of values and act as counters. The image pixels are analysed, and whenever a pixel falls within a bin, that bin's counter is incremented. The final product is a count of the color values of the image. Two color histograms can then be compared using simple distance functions to study image similarities. It is of note that light changes can be disruptive, and color histograms give no indication of an object's texture.

Edge features are derived from object boundaries. Boundaries usually create sudden changes on an image that can be detected by proper algorithms. Edge features have the advantage of

being less sensitive to shadows and light variations when compared to color features. Many edge detection algorithms exist, but perhaps the most widely used is the Canny Edge Detector. The Canny detector starts by smoothing the image with a Gaussian filter to diminish noise. Afterwards, edge detection filters are applied to the image to find horizontal, vertical and diagonal edges, which are then used to find the intensity gradient of the image. The edges are then thinned out by suppressing low gradient values, which is followed by thresholding the image between to discard all pixels that have a low chance of being edges. Finally, disconnected edge pixels may be are suppressed as being considered as noise. The final result is a image with the pixels detected as edges (Canny, 1986). Figure 15 shows the results of the canny algorithm.

Figure 15: Canny results

Edges also provide information about an object's shape. A good way to do this is using image moments. In mathematics, a moment is a statistic of a group of points, calculated in a way to represent a specific property. In the same way, image moments are weighted averages of pixel intensities, and can provide information about the image, like total area and centroid coordinates. A moment M_{ij} can be defined as

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

were $I(x,y)$ is the pixel intensity. From here, the central moments μ_{pq} can be calculated as

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x,y)$$

where $(\bar{x}, \bar{y})$ are the coordinates of the centroid represented by

$$\bar{x} = \frac{M_{10}}{M_{00}} \qquad \bar{y} = \frac{M_{01}}{M_{00}}$$

These moments are invariant to translation, but not scale nor rotation. To solve this, Hu (1962) described seven invariant moments that solve this issue. Having the scale invariant moment η_{ij} be

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+\frac{i+j}{2})}}$$

Hu's seven moments can be obtained from the following equations

$I_1 = \eta_{20} + \eta_{02}$

$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$

$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$

$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$

$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$

$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$.

Texture features are obtained by analyzing the variations on a surface or region. There are several different ways of obtaining descriptors for texture features, including co-occurrence of intensities, and wavelet decomposition. Co-occurrence of intensities consists in obtaining a matrix of identical colors or intensities within a region or direction. Wavelet decomposition analyzes an image as if it was a signal and decomposes it into several sub-bands. A comparison of several techniques for wavelet decomposition can be found at Ma and Manjunath (1995). Texture features are also more invariant to light variation and shadows, but usually require a bit more processing than color or edge features (Figure 16).

Another interesting texture descriptor is the Local Binary Patterns (LBP) operator. The original method analyses the eight immediate neighbours of each pixel in a 3x3 area. Each of these neighbouring pixels is thresholded with the center pixel in the following way:

$$\sum_{p=0}^{P-1} s(g_p - g_c) \times 2^p,$$

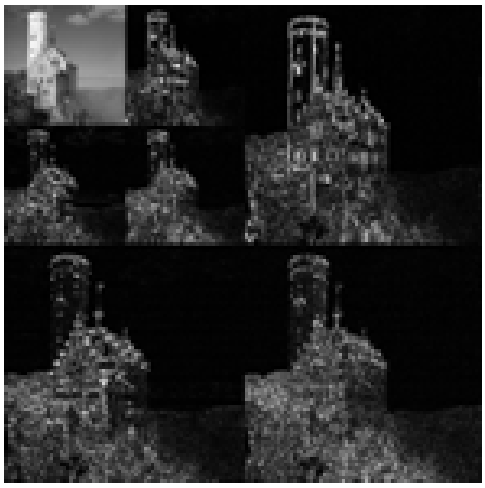
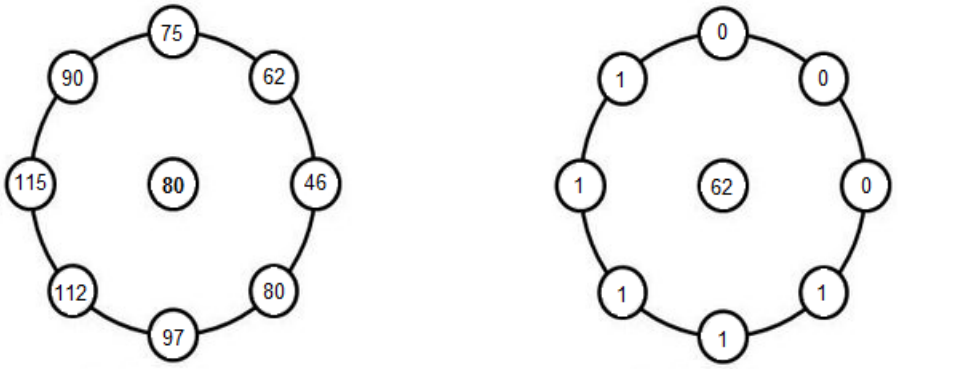


Figure 16: Example of Wavelet Decomposition

$$s(g_p - g_c) = \left\{ \begin{array}{ll} 1 & g_p \geq g_c \\ 0 & g_p < g_c \end{array} \right\}$$

where g_c is the center pixel and $g_p(p = 1..P)$ are the neighbouring pixels.



$$0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 0 \times 2^7 = 62$$

Figure 17: Example of Local Binary Pattern operator

Figure 17 shows an example of a LBP calculation. This method can generate up to $2^8 = 256$ different labels to describe a neighbourhood (Ojala and Pietikäinen, 1999). Since the original LBP deals with the adjacent pixels, it is not capable of dealing with scale differences. A proposal made to resolve this problem makes use of circular areas of variable radius. As such, instead of using the adjacent pixels, the neighbouring pixels are chosen with the following formula:

$$(x + Rcos(\frac{2\pi p}{P}), y - Rsin(\frac{2\pi p}{p}))$$

Interpolation is used to solve coordinates that do not correspond to actual image pixels (Ojala et al., 2002). Another extension to the classic LBP makes use of uniform patterns to encode specific characteristics of the image, reducing the length of the feature vector and making it invariant to rotation (Ojala et al., 2002). A LBP pattern is uniform if, when traversing it circularly, there are found no more than two bitwise transitions from 0 to 1 or vice versa. This allows for the codification of 58 different patterns representing features such as corners, edges or flat zones (Figure 18).

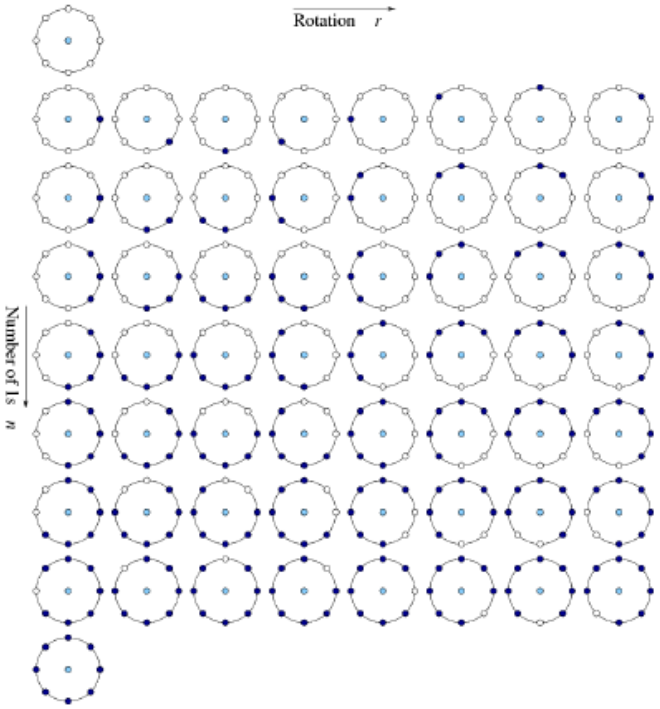


Figure 18: The 58 LBP uniform patterns in a (8,R) neighbourhood (Ojala et al., 2002)

A different approach to make the LBP operator more invariant to rotation was proposed by Mehta and Egiazarian (2013). Instead of using fixed weights, it shifts them according to the dominant direction (D) of each neighbourhood, that can be defined as:

$$D = argmax|g_p - g_c|$$

$$p \in (0...P-1)$$

The LBP operator will then become:

$$\sum_{p=0}^{P-1} s(g_p - g_c) \times 2^{mod(p-D,P)}$$

where $mod(x, y)$ is the modulus operation.

Even if the image is rotated, the dominant direction remains the same, resulting in the same calculated value. This makes the operator more robust and can be used both with or without uniform templates.

2.2.2 *Object Detectors*

Sometimes incorrectly assumed to be synonyms, object detection and object recognition are actually two distinct fields of computer vision. Object detection attempts to reveal the presence of objects, while object recognition tries to discern known objects. Both types of algorithms can be used to perceive an object in an image or set of images, a critical step in object tracking. There are a vast amount of solutions that were proposed to attain this objective, each with its own advantages and disadvantages, with the most popular capable of being grouped in four main types: Point detection, segmentation, background modeling and supervised classifiers (Yilmaz et al., 2006).

POINT DETECTORS Point detectors work in three steps. First, they search through an image for recognizable features of interest. It is important that these features are robust and more invariant to global perturbations such as changes in lighting, scale and point of view. Next, patches around those key-points are analysed and it's descriptors are extracted. Finally, those descriptors are matched against others to obtain a score that dictates the quality of the match. While mostly used for tracking and object recognition, this type of algorithms can also be used for other tasks like image stitching.

The Scale-Invariant Feature Transform (SIFT) algorithm (Lowe, 1999), is one of the most reliable point detectors available (Mikolajczyk and Schmid, 2005). Object features are detected through a technique called Difference-of-Gaussians (DoG), where the image is repeatedly smoothed and sub sampled with Gaussian filters of different scales. A Gaussian pyramid is computed from the differences between the images, and the points of interest are selected from the maximum

and minimum values, with the points of lowest contrast removed. The area surrounding these key points is analyzed to obtain the magnitude $m(x,y)$ and orientation $\theta(x,y)$ of the gradients.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \arctan\left(L(x+1,y) - L(x-1,y)\right) / \left(L(x,y+1) - L(x,y-1)\right)$$

Finally, the features are created by composing a histogram with the orientations of the gradients surrounding each key point (Figure 19).

Figure 19: The SIFT detector

Despite its reliability, SIFT is rather slow, especially for use in real-time applications. As an alternative, Features from Accelerated Segment Test (FAST) is a corner detection algorithm that is known for presenting good results while also being fast to execute (Rosten and Drummond, 2005). It makes use of circular filters to decide when candidate points are corners. If a set of N continuous pixels is brighter than the candidate pixel plus a threshold value, or darker than the candidate minus a threshold value, then the candidate is a corner. A high speed test in a few select pixels is used to eliminate non corner candidates and machine learning can be used to improve results. The patches surrounding the key-points generated by FAST can then be studied to generate the descriptors.

Binary Robust Independent Elementary Features (BRIEF) is a good example of an algorithm that can be used for such a job. BRIEF works by comparing the intensities of pairs of pixels within a patch, and generating binary values according to whether the first point has an higher intensity or not. The pairs of points to be compared can be chosen using different methods, like

uniform or Gaussian probability (Calonder et al., 2010). Another example is the Speeded Up Robust Features (SURF) algorithm, a point detector that attempts to strike a balance between quality and speed. It uses an approach rather similar to SIFT, but approximates the small Gaussians with box filters and makes use of the integral image to accelerate the computation (Bay et al., 2008).

BACKGROUND SUBTRACTION Background Subtraction is another category of object detectors, and Frame Differencing is the most basic of its implementations. It works by having an image of a static background, which is then subtracted from other images of the same scene but with the added objects in the image. The result will be an image of only the foreground objects in the scene. While simple and fast, this technique is extremely vulnerable to interferences like noise, shadows and lighting variations.

A more effective way to accomplish this effect is to use a dynamic model of the background instead of a static one. This will reduce those problems as the background will slowly incorporate any new changes to the scene, like the variations of light caused by the passing of time for example. A minus to this technique is that modeling the background this way will slowly incorporate foreground elements into the background, making detected objects disappear if they remain still in the scene for too long. For this reason, background subtraction is best used for detecting moving objects.

Creating a model of the background can be as simple as using a running average. A weight value or learning rate is used to decide how much each pixel of a new image will contribute to the background. Wren et al. (1997) use a similar running Gaussian average in their approach. They define each pixel by a mean μ_t and a variance σ_t^2 with the assumption that the initial image is composed only of background. Thus:

$$\mu_0 = I_0$$

$$\sigma_0^2 = h$$

Were I is the pixel's intensity and h is a default value. If we take t as the current image and ρ as the learning rate, we can update the background with the following formulas:

$$\mu_t = \rho I_t + (1-\rho)\mu_{t-1}$$

$$\sigma_t^2 = d^2\rho + (1-\rho)\sigma_{t-1}^2$$

With the background model in place, we decide if each pixel belongs to the background or foreground by defining a threshold constant k and obeying the following rules:

$$\frac{|(I_t - \mu_t)|}{\sigma_t^2} > k \longrightarrow Foreground$$

$$\frac{|(I_t - \mu_t)|}{\sigma_t^2} <= k \longrightarrow Background$$

The problem with single Gaussian models is that background scenes are very difficult to model, as constant movement of background elements, like leaves of trees, or sudden reflectance and shadows cause a lot of noise and interference (Gao et al., 2000). A more effective and popular approach to background subtraction is the Mixture of Gaussians (MoG) model (Stauffer and Grimson, 1999). This method models the background pixels with a mixture of k Gaussian distributions:

$$P(X_t) = \sum_{i=1}^{k} \omega_{i,t} N(X_t | \mu_{i,t}, \sum_{i,t})$$

Where:

$$N(X_t | \mu_{i,t}, \sum_{i,t}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\sum_{i,t}|^{1/2}} \exp\left(-\frac{1}{2}(X_t - \mu_{i,t})^T \sum_{i,t}^{-1} (X_t - \mu_{i,t})\right)$$

$\omega_{i,t}$ is an estimation of the weight of the mixture, $\mu_{i,t}$ is the mean, $\sum_{i,t}$ is the covariance matrix and N is a Gaussian probability function.

The Gaussians are updated using a running k-means approximation. Some variations of these algorithms make it so that each background pixel is only updated if the correspondent current pixel is considered background. Since background subtraction is relatively fast to execute, it is often used to detect regions of interest for more complex methods of object recognition. A limitation of this technique is the requirement of stationary cameras, as movement would distort the background model (Figure 20).

SEGMENTATION Segmentation consists in dividing an image into several parts or segments. Perhaps the most basic form of segmentation is Thresholding. This technique uses limits or thresholds to define the behavior of pixels. As an example, binary thresholding turns all pixels bellow a certain value of intensity to black while turning all others to white. Multiple threshold values can be used to segment different intensities, which is useful to isolated specific colors or objects (Figure 21).

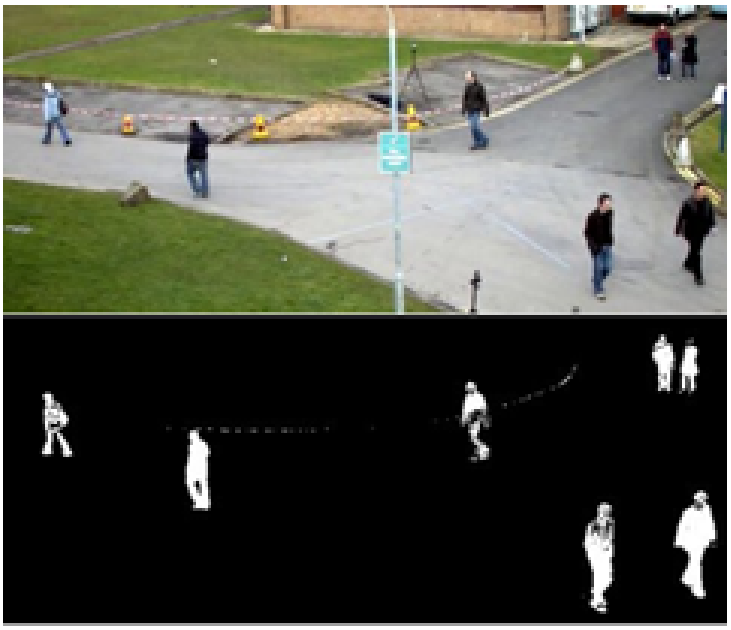


Figure 20: MoG Background Subtraction

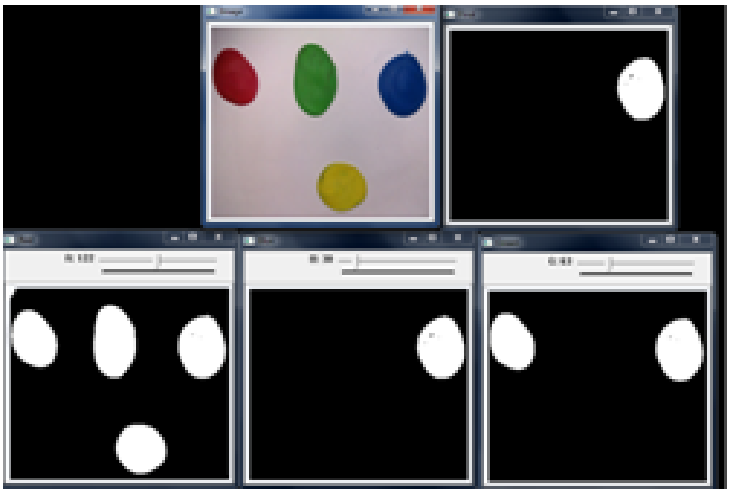


Figure 21: Segmenting an image by color through thresholding

This type of segmentation can also be used on depth maps to isolate objects at a certain distance from the camera. It is also possible to use dynamic values for thresholding, a technique known as balanced histogram thresholding. Otsu's algorithm is a good example of this method (Otsu, 1975). It iterates through all possible threshold values, separating the pixels into two clusters. It then calculates the mean of the clusters and the square of the difference between the two. Finally

it multiplies that number by the number of pixels in one cluster times the number of pixels in the other cluster. The ideal threshold value is the one that minimizes the variance of each cluster (Figure 22).

Figure 22: Otsu's thresholding

Another approach to segmentation is the mean-shift algorithm. To use this method it's required to define a range and a maximum distance. Each pixel in the image is then replaced by the mean of all the pixels in the neighborhood within range, which are within the defined distance in terms of intensity. Comaniciu and Meer (2002) use a mean-shift implementation for interesting results.

Graph-cuts is another technique for segmentation. It imagines pictures as graphs with the pixels being the vertices, and tries to cut the image into subregions based on weights derived by aspects such color or texture similarity. Shi and Malik (2000) define the weights by color and spatial proximity. The image is divided into two segments and for each segment the process is repeated until a threshold value is reached. This type of approach can have high memory requirements, especially for larger images.

The Watershed algorithm is a segmentation technique that imitates the geomorphological concept of a drainage basin. While there are many different versions of this technique, one of the most common is the marker based flooding algorithm proposed by Meyer (Beucher and Meyer, 1992). Different labels are given to each markers placed on the image. These are either manually placed or chosen from local minima of the image gradient. Then, starting from each marker, the neighboring pixels are added to a priority queue sorted by pixel intensity. The pixel at the top of the queue is then removed and its neighbours analysed. If all already processed neighbours share the same label, that pixel is then given that label. We add all unlabelled neighbours to the

queue and repeat the process until the queue is empty. The pixels that remain unlabeled are the watershed lines. This method is rather fast, but can easily lead to over-segmentation.

SUPERVISED LEANING Supervised learning is a class of algorithms that is mostly used for object recognition, and require user supervision to function. In this approach, the objects need to be learned by the system, normally creating a pair between a set of features and an object class. Learning the features from different views of the object or the creation of a set of templates for the object is required for a correct detection from different camera angles. Due to the high amount of samples required that need to be manually labeled by the user, it may be important to try and reduce the workload. An interesting solution for this problem was introduced by Blum and Mitchell (1998). They purpose the training of two classifiers for independent features with a small set of labeled data. After the training, each classifier will assign training data for the other classifier. This technique provides accurate results while reducing the amount of manual interactions needed for training classifiers.

An example of a classifier for this class of object detectors would be the Support Vector Machine (SVM) algorithm. It uses two types of data for training, positive and negative samples signifying that, in the context of image analysis, the sample contains or not the object. The data will then be grouped into two clusters and new samples will be compared against this model. The cluster that the new sample falls into will generate a response from the algorithm, stating if the sample contains or not the object.

DEPTH CAMERAS When using depth sensing cameras, object detection is usually built upon well know algorithms for 2D images, but makes use of the extra depth information. The added information can be extremely useful to solve many problems, like dealing with occlusions or detecting objects in dark areas or with colors that meld into the background. Owens (2012) combines depth map information with a Histogram of Oriented Gradients (HOG) detector, a method for object detection similar to the SIFT algorithm. It start by smoothing the point cloud and extracting its normals. Then segments connected components and extract regions of interest from the image. Finally applies the HOG detector on the regions and scores the results based on the histogram intersection and size of the segment.

Spinello and Arras also use HOG detection and combine it with a Histogram of Oriented Depths (HOD) to create a robust human detection algorithm. HOD works similarly to HOG as it analyses changes in depth and encodes them into a histogram of oriented depth gradients. This approach combines the rich features of color and texture from color images, which are vulnerable

to interference from changes in illumination, with depth information, which is more invariant to lighting changes but weak to noise (Spinello and Arras, 2011).

Another method breaks the point cloud into an adjacency matrix using a distance metric, and calculates the average normal for each cell using RANdom SAmple Consensus (RANSAC). Then it iterates through each point in the data set and, using the adjacency matrix as a guide, compares the point with its neighbors. The comparison is made based on color, normal and distance, and similar points will be marked as part of the same segment (Li et al., 2012).

Hernández-López et al. (2012) complements color segmentation with depth segmentation to detect objects. A color image is converted to the *CIE-Lab* color space in order to resist interference from illumination shifts, and a color segmentation based on a threshold value is performed to select objects of specific colors. This information is combined with information about depth segments to trim off undesired elements from the color segmentation.

2.3 OBJECT TRACKING

The goal of object tracking is to discover the trajectory of an object as it moves across a scene. An object is detected on a set of images or video frames and the positions are used to calculate the trajectory. The recognition of the object can be made as part of the object tracker or by using a separate object detection algorithm and feeding the position to the tracker. Objects to be tracked can be represented in various ways: a single centroid or a group of points *(Fig.23: a,b)*, ideal to represent objects with a small scale on the image, geometric shapes *(Fig.23: c,d)*, excellent to represent rigid objects, articulate shape models *(Fig.23: e)*, a group of independent moving shapes connected by joints, skeletal models *(Fig.23: f)*, and silhouettes *(Fig.23: g,h)* and contours *(Fig.23: i)*, best to represent deformable objects (Yilmaz et al., 2006).

The type of representation chosen for the object is important as it limits the range motions the system may track. For example, if an object is represented by its centroid, only translations can be detected, while geometric representations can work with affine and projective transformations.

There are three main categories for object tracking algorithms: point tracking, kernel tracking and silhouette tracking. In point tracking, objects are represented by points, and external means of detection are required to obtain the object's position. The information about the object's movement is calculated by analyzing and comparing the position of the points between frames. Finding the corresponding points between frames can be a complicated problem, particularly in the face of occlusions, false positives, or entries and exits of objects in the scene. Correspondence can be calculated by either deterministic or probabilistic methods.

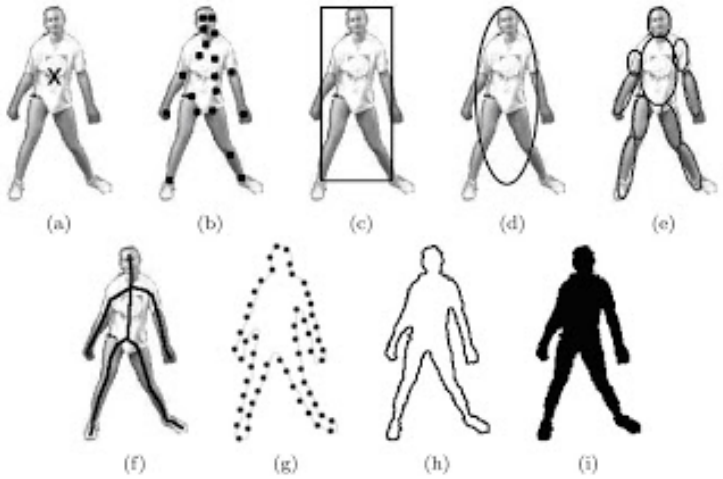


Figure 23: Object representations

Deterministic methods attribute a cost to each correspondence between points and then try to minimize the solution to obtain the final result. The cost for each association is calculated taking into account several conditions, such as proximity, velocity and object rigidity. The minimization is calculated using search algorithms like the greedy search method (Shafique and Shah, 2005).

Probabilistic methods take uncertainty into account in their calculations. Image noise or sudden motions of the object can generate problems for calculating correct correspondence, so this type of algorithms tries to model the properties of the object, taking into consideration variables such as position, velocity and acceleration. They try to find the current state of an object by building a probability density function based on the previous states of the object. The Kalman Filter can be used for this when the state variables follow a Gaussian distribution (Rosales and Sclaroff, 1999).

The Kalman filter is in fact a very useful tool for object tracking. It makes use of a state model built from previous measurements in order to predict a future state of the system. A noise factor (white noise, following a Gaussian distribution) is taken into account during calculations in order to improve accuracy of predictions, and new measurements can be added in order to improve the state model, creating an iterative cycle of prediction and correction. In essence, a the state $\bar{X}\tau$ of the model at any time can be obtained with

$$\bar{X}\tau = \tau_{-1} + \mu + \epsilon\tau$$

where τ_{-1} is the previous state, μ is the movement of the object or other type of perturbation of the state, and $\epsilon\tau$ is the noise. Given this, it is fair to assume that the any measurements $\bar{Z}\tau$ obtained shall be equal to

$$\bar{Z}\tau = C\bar{X}\tau + \epsilon\tau$$

where C is a function of measurement assumed as constant. Thus, the predicted following state X_{est} can be calculated with

$$X_{est} = \bar{X}\tau + K(\tau - \bar{Z}\tau)$$

where $Z\tau$ is the actual obtained measurement and K is gain factor known as Kalman gain (Welch and Bishop, 2006). Figure 24 show a demo where the Kalman filter is used to predict the mouse cursor's position.

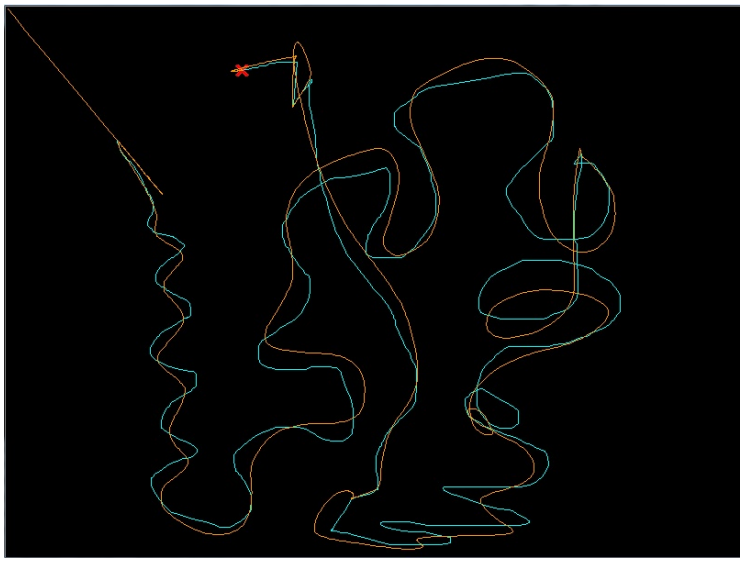


Figure 24: Kalman filter sample: Blue line is mouse trajectory and orange line is kalman filter trajectory

Particle filtering is an alternative to non linear systems or systems with noise that does not follow a Gaussian distribution. This filter works in a similar way to Kalman but uses a random Monte Carlo style sampling method to calculate a prediction of the object's state. The generated samples or particles are given a weight based on the received measurements, and low weighted particles are discarded. Finally, a resampling is used to generate more particles in the smaller, higher likelihood distribution area (Nummiaro et al., 2003).

These methods only take into consideration the tracking of single objects in the scene. For tracking multiple objects, it is necessary to associate each measurement of an object to its respective object state, something that can be difficult in situations where multiple objects are in

close proximity. Multiple Hypothesis Tracking (MHT) is a technique that is often used to solve this problem. This algorithm uses several images instead of only two to generate a group of correspondences, and decides the final position of each object by the most likely correspondence. This method can handle occlusion and the entrance or exit of object on the scene (Reid, 1979).

Kernel tracking usually works by analyzing the changes the area around the detected object undergoes between frames. It is a technique that can be used to track parametric type motion of objects. The differences between kernel tracking algorithms reside on the type of representation used for the objects, the number of objects tracked and the methods used to calculate the motion of the objects. Template matching is the simplest method of kernel tracking, and consists in comparing the region of interest with a template or series of templates. The templates are usually in the form of object features like color or intensity histograms and gradients (Birchfield, 1998).

Since template matching is a brute-force method, and as such is computationally expensive, the region of interest is typically reduced to improve performance. Another way to lighten the computational load of searching is to use the mean-shift algorithm. The mean-shift tracker represents the region to be tracked by a histogram and tries to find the region in the image that maximizes the similarity, which is calculated using the Bhattacharya coefficient:

$$\sum_{u=1}^{b} P(u)Q(u)$$

Where b is the number of bins, P the object histogram, and Q the window around the hypothesized object region.

The Kanade Lucas Tomasi (KLT) tracker is another kernel tracking method. It iteratively computes the translation of a region centered on a point of interest and evaluates the new region by calculating the affine transformation between the two. If the difference is small the algorithm continues tracking otherwise the feature is rejected (Shi and Tomasi, 1994).

A problem with these trackers is that they track single objects, and thus do not take into consideration the interactions between different objects, like object-to-object occlusions. A method of dealing with this problem is modeling the image as a set of layers, consisting on one layer for the background and one layer for each object. Each layer consists on a shape and motion model, and a model of appearance based on intensity. The layering itself occurs in two steps: first compensate the background motion modeled by projective motion, and estimate the object's motion from the compensated image using 2D parametric motion. Then calculate the probability of each pixel belonging to a layer based on the object's shape and motion features (Mishra et al., 2012).

Contour tracking consists in evolving an objects initial contour iteratively between frames. This approach requires that some part of the object's contour must be within the object's region of the previous frame. This type of tracking can be performed by either modeling the contour shape and motion using space state models, or minimizing the contour energy using direct minimization techniques. In the space state models method, the object's state is defined by the shape and motion of its contour. The state is updated every time the contour's posteriori probability is maximized. The probability is calculated according to the previous state and the distance of the contour from observed edges (Yilmaz et al., 2006). Minimizing the contour's energy makes use of direct minimization techniques such as greedy methods or gradient descent. The contour's energy can be defined either by optical flow (Cremers and Schnörr, 2003) or by appearance statistics generated from the object and the background regions (Yilmaz et al., 2004).

An advantage of silhouette tracking methods is the ability to handle very diverse object shapes, as well as the merging and splitting of objects, like a person picking up or dropping a ball. A disadvantage is that most methods do not deal directly with occlusions. This is usually resolved by assuming constant motion and acceleration of objects.

Just like in object detection, depth sensing cameras can be used to complement traditional tracking algorithms with depth information and improve results. Due to poor scene information, the most efficient trackers tend to rely on heavy calculations and models resulting in poor performance. The use of more robust information can simplify the models and calculations necessary, increasing performance and solving certain problems like partial occlusion.

Nakamura (2011) combines color with depth information on the traditional particle filter to obtain fast and accurate tracking. Another technique is to use a particle filter tracker to generate guesses, and feed them to a Iterative Closest Point (ICP) tracker (Kim and Kim, 2008).

ICP is an algorithm that minimizes the difference between two point clouds. This method of tracking tries to find an object in a point cloud, iteratively rotating and translating the object until it either finds the best match or fails. The mean-shift generated guesses will speed up this process.

A more complex but interesting method is proposed in Park et al. (2011). They use a group of Dominant Orientation Templates (DOT) together with depth information to generate a 3D model of the object, dismissing the need to train a model for tracking and pose estimation. ICP is then performed to track the object in the scene. DOT is a fast method for acquiring templates. The templates are composed of discretized gradients where every pixel of the template contains the dominant gradient in a small surrounding area.

2.4 SUMMARY

This chapter exposed the concept of TUI and demonstrated several interesting implementations of the idea. They can provide engaging and immersive alternatives for human machine interactions, but are usually oriented to solve very specific tasks and require very specialized equipment.

Several computer vision algorithms to describe, detect, recognize and track objects were also studied, and their strengths and weaknesses analyzed, in order to find the best means to achieve the goal of this dissertation. A challenging task, made more complex by the combination of characteristics required by the project.

Color and shape descriptors present good invariance to scale and rotation, and though color descriptors are not very robust to light changes, different color models can be used to minimize the impact. Texture descriptors can provide very useful information about the objects, but it is not usually invariant to rotation and scale without the help of more complex methods such as supervised learning.

While many of the studied methods can achieve the desired solution with robust results, they present other disadvantages, such as being too computationally expensive, making use of complex object models, or needing full time visibility of the tracked object. For this project, a combination of lighter, less robust algorithms will be favored. The combined methods will help cover for each others weaknesses in order to provide more reliable results, while maintaining acceptable processing speed.

The next chapter will make use of the gathered information to describe the architecture and implementation of the framework proposed in this paper.

3

ANY OBJECT IS A PC CONTROLLER

The idea behind a TUI is to bring the physical and the virtual world closer together. To allow the user to grasp and manipulate virtual data as if handling an actual physical object. In order to achieve this, complex and sometimes expensive technological equipment is required. Furthermore, due to their intricacy, TUI are generally oriented to resolve a single task.

The aim of this dissertation is to confront this problem, by studying and developing a solution capable of enabling the use of everyday objects as controls for tangible user interfaces. The system should use readily available and reasonably priced equipment, and should be able to communicate with client applications, appointing the objects as controllers for data manipulation.

A simple practical example of this could be controlling a character in a video game. Moving a can around on a table could move the player character, rotating a wallet could select options from a radial dial, and a simple pen could act as a slider for configurations such as sound volume. The player would be able to move the character not by pushing buttons, but by actually grabbing it's physical representation, and moving it around using more natural gestures. He could manipulate virtual sliders and dials as if he was actually grabbing them with his own hands. This would increase the level of control the player would have, and also increase the immersion, making the player feel more connected with the virtual world.

To achieve this task, the creations of a modular framework was found to be a good solution. The adaptable nature of a framework would allow for any modifications and improvements necessary to accommodate distinct scenarios. An RGB-D camera was recognized as a sufficient sensor for this project, so an approach based on computer vision algorithms will be adopted.

Several specifications were identified as requirements for this framework to function. Unfortunately, despite the extensive research in the field of tracking applications, integrating all these specific conditions together creates a set of complex problems that need to be solved for these goals to be met. The next few sections of this chapter will describe in greater detail the require-

ments expected from this project, as well as the architecture and implementation of a system designed to attempt to fulfill them.

3.1 REQUIREMENTS

By analyzing the goals of this projects, there were several requirements that were deemed essential for this framework to function.

Overall, the framework:

- Should use readily available and reasonably priced equipment.
- Should be simple to setup and use.
- Should be fast to respond to user input.
- Should be able to detect one or more objects over a flat surface.
- Should be able to accept a great variety of objects.
- Should be able to track the 3D translation of one or more objects over an area.
- Should be able to track the 2D rotation of one or more objects over an area.
- Should be able to recover from partial or total occlusion of the objects, even during movement.
- Should be able to attribute tasks to objects, such as making them acts as dials or sliders.
- Should be able to transmit events acted upon the objects to client applications.

The proposed system will be designed with respect to these requirements.

The argument for the importance of these conditions can vary according to different factors, from limitations of hardware and software, to a question of user experience and comfort. It is imperative to keep in mind these circumstances when building the framework. The next few sections will attempt to explain the reasoning behind these decisions.

3.1.1 *Equipment*

Many TUI use complex equipment built specifically for those applications, making it necessary for users to either build or buy those devices. Furthermore, they would be of use only for that system, requiring more investment when using multiple types of tangible interfaces. The use of a single sensor like a camera, that is economical and easy to obtain, would be immensely more attractive prospect. Better yet, a camera is an equipment with many distinct utilities, not being restricted to be used by this framework.

3.1.2 *Simplicity*

As an interface, it is important that users can set up the application quickly and without too much hassle. If a system is too complex or takes excessive effort to configure and use, it will be dismissed in favor of other, more straightforward options. Because of this, creating a product that users can just pick up and use will make it far more appealing, and encourage it's continued employment.

3.1.3 *Responsiveness*

An important condition for any good interface is that it must be quick to respond to user input. When a user performs an action, it must see an immediate reaction from the system. This brings up an important issue of quality versus performance. More robust solutions tend to have poorer performances, so it is vital to strike a good compromise between the two. Since the interface framework and the client application can run in the same machine, it is especially important to build a lightweight solution without a steep resource consumption.

3.1.4 *Object Detection*

Object detection is a complex field of computer vision where no solution exists that solves all problems. Usually, each existing method answers very specific questions, and needs to be studied and adapted on a case by case basis. For this system to work, it will require that, at a bare minimum, objects can be detected on top of a flat surface. This surface will serve as the user's workspace, where he will manipulate the interactive objects. It can be anything like a desk, a

table, or even the floor. Further extensions to the detection can be considered, but this case will always be minimum requirement.

3.1.5 *Object Variety*

The goal of this framework is to turn any common day object into a tangible controller for diverse software applications. While there will certainly be limitations on the types of objects the system is able to handle, it is still important that the framework can utilize a great variety of objects. Because of this, it is relevant to try and disassociate the objects from the system, and try to make it as generic as possible. This will be make it so that the user does not need to utilize specific object types, but can instead make do with any item he finds convenient.

3.1.6 *Translation Tracking*

One of the more important aspects of the framework will be to find were each object is at any time. Object position is a powerful indicator that can be used to generate events. Moving an item freely on a surface may move an object in a virtual space, moving it left and right may control a toggle or a slider in application, and moving it to a particular position in relation to another object may represent indicators such as wind speed or light intensity and angle. When using depth sensing cameras, the system should also be able to track the object's height or elevation, to provide further data for the system.

3.1.7 *Rotation Tracking*

The orientation of an object is also another valuable source of information for event generation. Rotating a physical object can translate into events such as orienting a virtual item or changing the position of a dial. Since the objects appearance can change drastically depending on it's orientation, the framework will only require the tracking of it's two dimensional rotation. This happens because appearance is a vital aspect of object tracking, and can radically influence the availability of tracking methods.

3.1.8 *Object Occlusion*

Object occlusion is a challenging obstacle that can disrupt many tracking algorithms. Since the sensor used in this system will be a camera, it is unavoidable that object must be visible for events to be generated. Despite this, completely restricting the possibility of occlusion occurring is also not viable, due to the very nature of the framework. Since the system will be used as a framework, the object will be handled by a human user, who will eventually cause an object to become occluded, either by hiding it with his hand when grabbing it, or covering it with is arm or body when moving another object in the scene. Prohibiting situations like this would provide for a very poor user experience, and handling the tangible items would not be as smooth and natural as expected by a TUI. For this reason, it is important to build a system that can partially handle, or at least can recover, from object occlusion, especially during more problematic situations when it occurs during the object's movement.

3.1.9 *Object Functions*

While information about the object's position and orientation are important, it is also advantageous to distribute functions or tasks to each object from the beginning, so that only useful information is tracked. Giving certain object specific tasks, such as slider or dial, can not only make the user more aware of what actions to perform on the object, but also help filter proper data for the client applications. As an example, if an object acts as a x-axis slider, there is no use in tracking orientation or y-axis and z-axis coordinates.

3.1.10 *Event Transmission*

The purpose of this system is not to serve a specific application, but to act as a framework that can provide tangible interfaces for any number of applications. As such, the framework must be able to generate clear data representing the actions performed on the tracked objects, and send it through a well established communication channel, to a client application.

3.2 ARCHITECTURE

Balancing all these requirements together and produce a robust solution can be a very challenging problem. This section will delineate the proposal of a framework that attempts to meet these

objectives. The framework requires a camera to be oriented towards a workspace, like a desk or table.

The user can register objects to be tracked in the system by placing them one by one on the workspace. When registering a new object, it is possible to attribute a particular task to it. Currently defined tasks are sliders in all three axis, dials, free tracking, area tracking and relative distance tracking.

A slider task calculates the position of the object relative to two other positions. The system will keep track of the object when between those two coordinates and generate a percentage value based on the distance between points. The dial task tracks only the object's rotation relative to it's original orientation. Free tracking will follow the object through the workspace, keeping note of it's coordinates and orientation. Area tracking follows the same principle, but the system will only keep note of events applied to the object within a circular area defined by the user. Finally, relative distance tracking will keep notice of the distance between two registered objects.

The framework will keep track of registered objects, while ignoring any new objects added to the scene. It is possible to stop the tracking process to register new objects in the system. Depending on the task assigned to each object, events such as moving or rotating an object will generate a message that will be transmitted to client applications. The messages will be in the form of UDP packets, containing data concerning the event and the object that generated it.

The objects must be visible by the sensor in order for events to be generated. Nonetheless, the system should be capable of recovering from problematic situations, such as short and long term occlusions caused by the user or another object, entrance and exit of objects in the scene, and occlusions during the object's movement.

The framework will be composed of five main modules, Camera, Setup, Detection, Tracking and Communication. The Camera module establishes connection with the sensor and retrieves color and/or depth data. The Setup module is an optional part where preprocessing can be performed on the retrieved images. This is also where camera calibration and image distortion can be performed should such be required by the user or the client application. This data will then be fed to the Detector module, where the objects will be detected and segmented. That information will be used by the Tracking module to analyze actions performed on the objects and generate corresponding events. Those events will be transmitted to a client application though the Communication module.

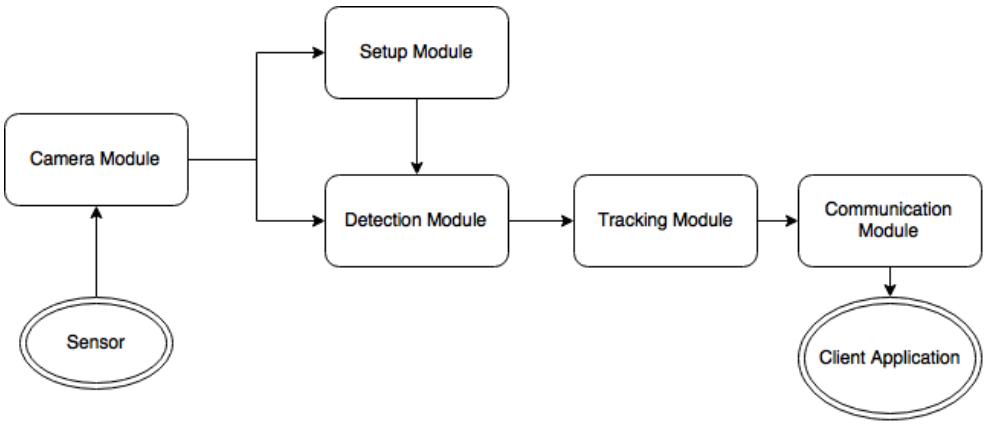


Figure 25: Diagram of the framework

3.2.1 *Camera Module*

This module simply focus on retrieving data from the chosen sensor. The default module allows the framework to communicate either with a normal webcam or the Microsoft Kinect depth sensing camera. It can retrieve color and depth data and transmit it to the other modules, bridging the gap between hardware and software.

Additionally, this module also offers the possibility of retrieving data from previously recorded videos. Depending on the video format, both depth and color information can be obtained, allowing the system to function even without a connected sensor.

3.2.2 *Setup Module*

This optional module can be used to perform several preprocessing operations on the retrieved images, such as color space conversion or noise reduction. It is also possible to perform camera calibration with the classic method of homography calculation with a checkerboard pattern. The calibration data can is saved to a file so it can be reused. When triggering image calibration this module will also perform image distortion using the calibration parameters.

3.2.3 *Detection Module*

Color and depth data will be used in this module to detect and segment objects within the workspace. The detection and segmentation of the background can be performed manually with

user input, or the system may try to automatically calculate the best result. The module will then produce a binary mask image separating the foreground from the background.

3.2.4 *Tracking Module*

The main module of the framework, it has two main phases of functioning. It allows users to register new objects for tracking, and tracks the registered objects. When registering new objects, users can also attribute them certain tasks that will modify the tracking methods used and the generated events. During the tracking phase, it will take the information obtained in the previous module and try to identify the objects in the image. The movement of each object will be analyzed and an event generated accordingly. The event will be in the form of a byte array, which will be handled by the Communication module.

3.2.5 *Communication Module*

The final module of the framework. It establishes connection with client applications and transmits messages generated by the tracking module. The default module uses the UDP protocol to send data packets to a single application.

3.3 IMPLEMENTATION

Having described how the framework was structured, it is time to portray its implementation. Several important design decisions had to be made in order to coordinate all the initial requirements, and the issues inherent to each. The framework was written in the *C++* language, and makes use of the open source libraries *OpenCV* and *OpenNI2*. *OpenCV* is a library that is extensively used for image processing by the community of computer vision enthusiasts, while *OpenNI2* offers a practical way to interface with the sensor.

3.3.1 *Setup*

It was decided early on the development cycle that the system would use an RGB-D camera that would be placed on a top-down view, perpendicular to the surface plane (Figure 26). This would allow several advantages, such as the full view of the entire workspace and tracked objects, reduc-

ing object-to-object occlusion and simplifying the object description. Using a top-down perspective and limiting the rotation tracking to two dimensions, the system only sees one perspective of the object.

This is an important step, as it removes the need to create a model of each object, a complex and time consuming effort that goes against the requisites of the framework. This way, the user can quickly register several different objects with little effort, and immediately start using the application.

The added depth data will also help the segmentation process and allow the system to track 3D translations of object in the workspace.

Due to the way the segmentation works, it is important that the workspace consists in a flat surface. It is also convenient that, when registering new objects, the workspace be clear of clutter, which will greatly aid the registering process. During the tracking process any detected clutter will simply be ignored.

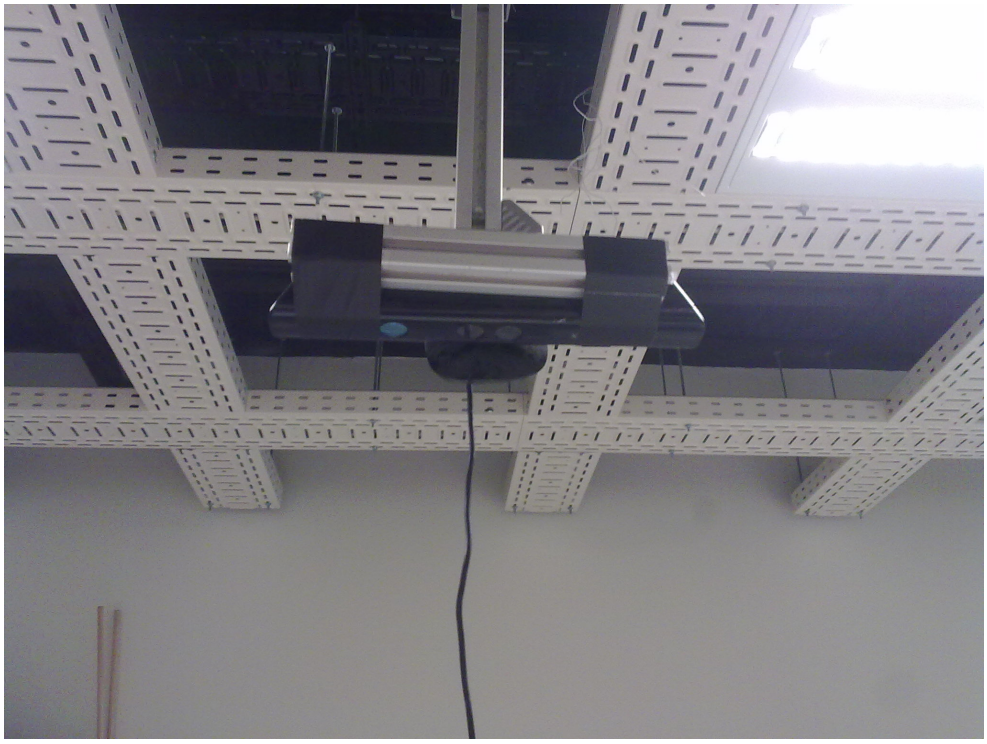


Figure 26: The ceiling mounted Kinect. It is facing down at a perpendicular angle to the surface of a desk bellow.

Should it prove necessary, camera calibration can also be performed during setup. Calibration removes the natural distortion added to the images by the camera lens and helps when determin-

ing the relation between the images' and the real world's coordinates. The calibration process follows the method proposed in Zhang (2000). A chessboard pattern is shown to the camera, and its corners are detected. Since the real measurements of the chessboard are known, a simple homography matrix can be calculated. This is repeated several times, each with the chessboard in a different position or perspective, and then a homogeneous linear system is used to capture the relationships between views.

3.3.2 *Object Detection*

Object detection is one of the first and most important steps in the tracking process. A good, clear segmentation of the foreground objects from the background allows for a smooth operation of the rest of the system.

The framework uses depth segmentation to filter the object on the workspace. Since the camera is placed over the surface at a perpendicular angle, it is possible to make use of the depth map to detect and extract the workspace, leaving only the object in the image. After the segmentation, a threshold is applied to the image to create a binary mask where areas with objects are marked in white, and the background areas are marker in black (Figure 27).

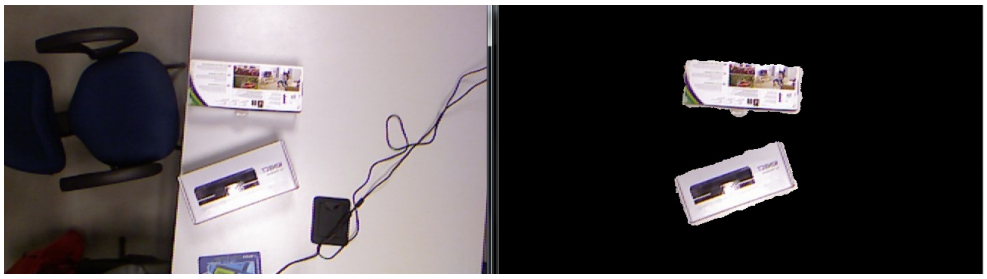


Figure 27: An example of segmentation. This image combines color information with the segmentation mask to produce the right side image.

Automatic segmentation of the surface can be achieved by building a depth histogram to find the most common depth value, which will correspond to the depth of the workspace. The system then lowers that value by a small margin to compensate for depth error and removes all pixels above that threshold. The padding value is adjustable and depends on the distance from the sensor to the surface. As an example, according to Khoshelham (2012), the depth error of the Kinect camera at 2.5 meters is around 10 millimeters. For more accurate segmentation, the framework also offers the possibility for the user to manually adjust the results.

While this method does require that the camera be at a mostly perfect perpendicular to the surface, it also offers several advantages over other processes. Depth segmentation is extremely fast to execute, provides a clear separation of the object from the foreground, is invariant to illumination changes, and does not require user supervision.

Other options were also taken into consideration. Background Subtraction did not produce satisfactory results due to its susceptibility to changes in light conditions. An attempt to combine depth information with the color image was made, based on the algorithm described in Fernandez-Sanchez et al. (2013), but the results were still rather unsuitable, especially regarding the shape of the detected blobs.

Plane detection algorithms were also briefly considered, but analysis of results obtained in several papers proved them to be either too unreliable for this situation or too computationally expensive (Haines and Calway (2014),Feng et al. (2014)).

3.3.3 *Blob Splitting*

After the foreground mask that segments the objects from the background has been created, there is still an issue that remains. Foreground areas appear as white blobs on the mask, but in certain circumstances, such as when two objects are too close together, or when the user is holding an object, two different blobs can merge together as one. This makes it difficult to separate the various objects to track (Figure 28).

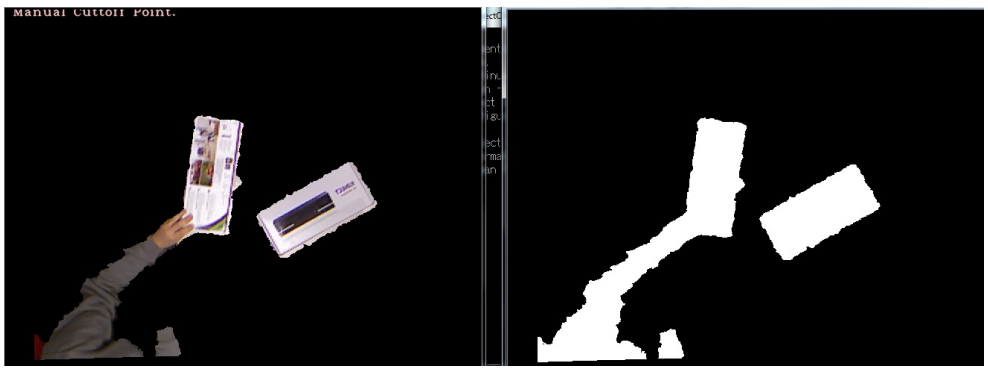


Figure 28: Example of connected blobs

To solve this problem, the framework makes use of the watershed algorithm described on subsection 2.2.2. This method makes use of markers placed on the image to segment different

objects. These markers are generated automatically through a process know as distance transform.

Distance transform uses a distance formula, like Euclidean or Manhattan, to calculate the distance between each pixel and the nearest zero-value pixel. This formula is applied to the segmentation mask, creating a greylevel distance mask were the central areas of the blobs are brighter, and the borders fade to black (Figure 29). A simple threshold is applied to the distance mask to remove the darker borders of each blob, leaving only the mass centers behind. These mass center can serve as markers for the watershed algorithm to separate the connected blobs from each other (Figure 30).

Figure 29: Right: segmented blobs, Left: respective distance transform

Figure 30: Image of resulting generated markers

3.3.4 *Object Tracking*

After obtaining all the separated blobs, the system needs to identify them. As no previous model or information about the objects is known, the framework makes use of color and shape descriptors to identify each object.

Color histograms were chosen to describe the object's color (refer to subsection 2.2.1 for details). To create it, the area surrounding the object is extracted and converted to the HSV color space, a color model that boasts a higher tolerance to illumination changes than the commonly used RGB format. Furthermore, to make the histogram more resilient to changes in lighting, only the hue and saturation values of the image are used.

To describe the shape of the object, the moments of the object's contour are calculated, and from there Hu's invariant moments are extracted. The process to achieve this was described in more detail in subsection 2.2.1.

The advantage of using these descriptors is that they are both extremely fast to create and match, and boast invariance against translation, scale and rotation. When an object is registered in the system, the framework extracts and stores these features for later comparison. To match descriptors, simple distance formulas such as Hamming or Manhattan can be used to calculate the similarity distance.

To avoid matching all objects against all blobs, the framework also makes use of the object's position. The system uses the centroid of the object, extracted from its contour moments, to determine the object's coordinates. To calculate its height or elevation, the system makes use of the depth data, analyzing the depth of the object in relation to the camera. Since the depth information presents too much noise, the system cannot directly use the depth of the object's centroid. Instead, the average depth of all of the object's pixels is calculated, providing a much more stable value.

Having calculated an object's coordinates, the system can now store the last known position of each object. Due to several issues such as occluding an object while moving it, moving an object at high speed when using a camera with a low rate of acquisition, object with crossing trajectories or moving one object too close to another, concepts such as nearest neighbor and last know position do not offer reliable information.

To help solve these issues, the framework makes use of the Kalman filter to keep track of each object's movements (see section 2.3). This reduces the search area of the system to the blob closest to the Kalman's predicted position. Because the movement of an object can suddenly

end, the framework also checks the last know position of the object should the first match fail. Whenever the object is detected, the Kalman filter is updated with the object's current position.

Despite this, there are still several situations were the Kalman filter can fail. For example, the collision between two moving objects or the sudden erratic movement of an object, especially under occlusion, can make the system lose track of the object. Because of this, the framework presents the option of, should the object not be near neither the Kalman prediction nor the last known position, performing a validation checkup by comparing the object's descriptors with those of the remaining blobs.

3.3.5 *Object Orientation*

After finding the object's position, the framework needs to determine its orientation. Since the system has no previous knowledge of the objects, it has no information about its facing. Also, since the object can be occluded while being rotated, the system cannot use previous frames as reference either. To solve this issue, the framework provides two different methods, one based on the object's shape, and another on it's texture.

The first method analyses the shape of the object and searches for a point that could be declared the *front* of the object. Any vertex that is farthest or closest from the center that the others is a candidate. This makes objects with regular shapes like squares or circles invalid for rotation calculations using this method. The system traces a line between the centroid of the object and the object's *front*, and uses it to calculate its orientation in relation to its original facing. To find the rotation angle, the following method is used: Let (x_1, y_1) and (x_2, y_2) be the two points that define a line. Also, let $atan2(y, x)$ be an arc tangent function where

$$atan2(y,x) = \left\{ \begin{array}{cc} arctan(\frac{y}{x}) & \text{if } x > 0 \\ arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ arctan(\frac{y}{x}) - pi & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{array} \right\}$$

The angle of the line can be obtained with

$$\theta_1 = atan2(y_1 - y_2, x_1 - x_2)$$

Obtaining the angles of the two lines and calculating their difference will provide the object's rotation in relation to it's original orientation.

To validate if an object can be used in this method, the system gathers 24 image frames containing the object. Then, it rotates each image in increments of 15^{o} degrees, and tries to calculate the object's orientation in relation to the first image. If the calculated value closely matches the degree of rotation of that image, it is considered a success. If the percentage of successes reaches an acceptable value, the orientation of that object can be determined using this algorithm. Since this method is fast to calculate, the framework attempts to validate all registered objects.

The second method makes use of the object's texture. Since it is a lot slower than the previous method, the user needs to explicitly declare that he wants that object's rotation to be tracked using this process. This technique uses SURF to find keypoints in the object, and then, once the object is found during tracking, it extract it's keypoints again and calculates the homography matrix between the two sets (subsection 2.2.2). From there it is as simple as extracting the rotation component from the matrix. To calculate the homography between two sets of points the following method is used: Let X_1 and X_2 be the coordinates for the two sets. It is known that X_2 can be obtained with

$$X_2 = RX_1 + T$$

were R is the rotation matrix and T the translation vector. Then, it is assumed that X_1 lies on the plane defined by

$$n^T X_1 = h$$

were n is the plane's normal vector and h the distance to the plane. Thus, by substituting into the previous formula

$$X_2 = (R + \frac{Tn^T}{h})X_1$$

and the matrix

$$H = R + \frac{Tn^T}{h}$$

is known as the homography matrix which takes the shape of

$$H = \begin{bmatrix} cos\theta & -sin\theta & -\frac{t_1}{h} \\ sin\theta & cos\theta & -\frac{t_2}{h} \\ 0 & 0 & 1 \end{bmatrix}$$

From here the rotation angle θ_r can be obtained by using the previously defined $atan2$ function

$$\theta_r = atan2(-sin\theta, cos\theta)$$

Since this method is computationally expensive, the application only applies it when the object is moved. To detect movement, the system performs frame differencing and compares the resultant mask with the segmentation mask. If movement is detected within the area of an object, the application calculates it's rotation.

One last method for determining an object's orientation also makes use of shape characteristics. Taking a pen as an example, it is possible to get an overall idea of it's orientation even without knowing which side is the tip. Other objects possessing similar height and width ratio differences are good candidates for this algorithm. Using the least-square method for curve fitting described in (Fitzgibbon and Fisher, 1995), the ellipse fitting the object's contour can be found. Calculating the orientation of the ellipse relative to the major axis can provide a general idea of the object's facing. The orientation of an ellipse can be obtained from the equation for ellipse rotation:

$$A(x-h)^2 + B(x-h)(y-k) + C(y-k)^2 = 1$$

from here, the angle of the ellipse can be obtained with:

$$\theta = arctan(\frac{C - A + \sqrt{(C-A)^2 + B^2}}{B})$$

The limitation of using this method is that it can only provide an orientation up to 180 degrees, as varies between $(\frac{-\pi}{2}, \frac{\pi}{2})$ or (-90°,90°). On the other hand, this algorithm is computationally inexpensive and can be employed for objects that do not fit the criteria for the other methods.

3.3.6 *Communication*

When any object is moved according with the rules set by it's task, the system generates a message to be transmitted to a client application. The default module makes use of UDP sockets to transmit the packages.

Standard messages contain the id of the object that generated the event, the coordinates of the origin point of the object, that is, the position where it was first registered in the system, the current position, and the current orientation in degrees.

Assigning tasks to objects can modify the contents of the event messages, in order to communicate only important information to the client applications.

3.4 CONCLUSION

Implementing the framework was a challenging problem, since it was necessary to conciliate all the diverging requirements. Because of this, certain choices had to be made and restrictions had to be placed on the system. The system uses a single RGB-D camera as required, is simple to setup and use, and the lightweight methods it uses allow it to be run in real-time. It can detect objects placed on a surface, and accepts a great variety of them, but there are a few limits. First, as explained before, very thin objects may have some trouble being detected depending on the distance between the camera and the surface. The RGB-D camera also cannot correctly detect transparent object, something that is a limitation of the hardware itself. Finally, since the shape of the used objects is an important descriptor for tracking, the system can only handle rigid body objects.

The system can track the 3D translations of the registered objects, and is robust against multiple types of occlusion. There are a few issues however. As previously explained, the tracker performs an exhaustive search on all blobs when one object is lost. This makes it so that objects can still be tracked even when moved during occlusions, for example, when the object being moved is hidden by the user's hand. This also means that multiple objects can be manipulated and occluded at the same time with no issue. The problems emerges when using identical objects, which would break the system. While they can still be tracked by turning the exhaustive search off, the application would be reliant entirely on positional data.

Another similar issue happens when using object that present comparable characteristics without being identical. If two objects possess identical shape and color distribution, they still pose a problem for the framework, as texture information is not taken into account. While it is easy for the application to detect these cases, it becomes far harder to resolve them. The reason for this is that texture descriptors tend to not be completely invariant without the use of computationally heavy algorithms or object models.

An object model could be built using machine learning, making the tracker much more robust, but it would come at a great cost. Creating a reliable model takes a tremendous amount of pictures, as well as a huge workload for the user for every object registered. This is something that goes against the stipulated goals of this dissertation. It would be possible to ease the process by using online learning, making the model be built during it's operation by the user. This would

have the draw back of the user needing to pay careful attention to the object's manipulation, and could still lead to the creation an incorrect or poor quality model. The application's performance speed would also take a heavy hit, growing worse with multiple objects being tracked with this method.

An example of features that could be used to build a model is histograms generated from LBP operators. Several attempts were made, using the invariant LBP methods described in subsection 2.2.1, to use LBP histograms to describe an object's texture. Unfortunately, generating a single histogram of the whole object does not produce robust results. A better solution would be to split the image into sections and calculate the histogram of each. Matching histograms this way would not be rotation invariant, and trying to find the combination that results in the most favorable values is a very slow solution. A better idea would be to build a model of the object using a machine learning algorithm, and then try to match the histograms with the model.

A different option would be to use point detection algorithms to validate the object's texture. Since these methods are rather heavy, the framework could detect similar objects and apply this solution only when needed. Upon registration, the color and shape of an object would be matched against previously registered objects, and whenever two matches are found to be too good, those object would be marked as similar. To further reduce the workload, the object could be tracked under the normal methods, and use this validation only when the system either loses the object or when two similar objects are too close.

The problem in this situation lies in validating the matches. It is very difficult to validate a match without knowing anything about the tracked object. A matcher compares two sets of descriptors and returns a set of distance values dictating how good those matches are. The homography matrix between keypoints with good matches could also be calculated and it's integrity validated to obtain more information. Despite this, it is still very common to obtain false positives and negatives. Also, objects would need to have rich textures for this method to work. At least four good matches would need to be detected for homography to be calculated. This would once again be limiting the type of object that can be used.

Concerning rotation, the system offers three different methods to be applied for distinct objects. Each method has its limitations, so none of them can be used for every case. As required, users can also attribute tasks to each object, and any detected event will be communicated to client applications connected to the system.

In the next chapter, several tests will be performed on the framework, and the results will be analyzed in more detail.

4

RESULTS

The previous chapters presented the goal of this dissertation, explained the inherent challenges and described the steps taken to overcome them, outlining the design and implementation of a framework that proposes to meet that intent. This chapter will focus on testing the framework, validating it according to the initial requirements, and identifying the main limitations of this implementation.

The framework is evaluated according to several aspects, including execution speed and tracking accuracy.

To illustrate the concept and to perform the tests, in accordance with the theme of the project, several common daily use objects were collected. Some examples of gathered object can be seen on Figure 31.

The computer used to run the tests was a low end laptop, with a *Pentium®Dual-Core CPU T4400 @ 2.20GHz* processor and 4Gb of RAM memory. A *Kinect 360* camera was chosen as the sensor, and mounted on the ceiling (Figure 32) while overlooking a wide table that served as the workspace and interaction surface (Figure 33).

4.1 OBJECT DETECTION AND SEGMENTATION

The conducted tests demonstrated that the segmentation works fairly well, showing invariance to changing light conditions, being fast to execute and providing a good, clear separation of the objects from the background (Figure 34). The process is simple to perform and mostly automated, though users may optionally provide input in order to improve results. By manually adjusting the threshold value, users obtain direct control over the results, raising or lowering the segmentation plane for tasks such as clearing small blobs born from depth noise or raising the height at which objects are detected (Figure 35). Before registering the objects in system, users will be given the chance to verify the segmentation results and perform any desired adjustments.

Figure 31: The objects used for the main tests. From left to right, top to bottom, a small square box, a wallet, a cardboard triangle, the box of the *Kinect One*, and the box of the *Kinect 360*

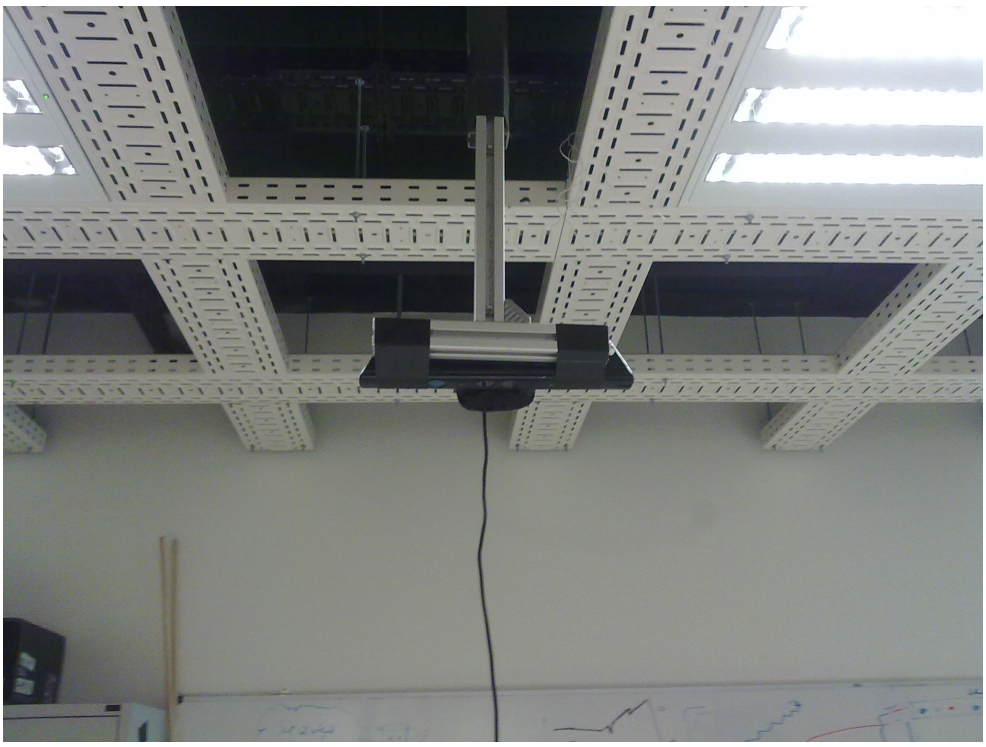


Figure 32: The Kinect camera mounted of the ceiling.

Figure 33: Two users using boxes as tangible controllers.

Figure 34: Segmentation Sample.

The addition of the watershed algorithm also serves to improve the quality of the segmentation even further by splinting connected blobs and displaying the full shape of the object. As seen in Figure 36, the results of the watershed method are of good quality, suffering only occasionally from limitations of the method itself. Over-segmentation proves to be an issue only when severe artifacting related to depth information loss appears, a sample of which can be seen in Figure 37.

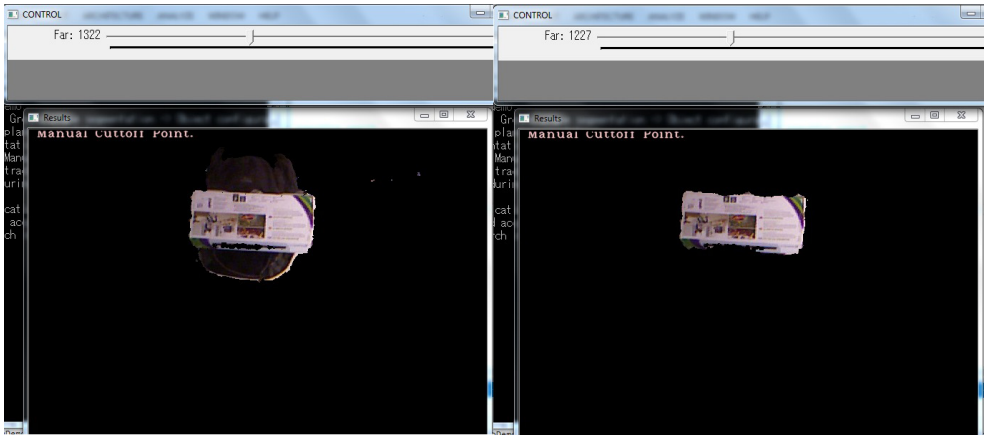


Figure 35: Sample of manual segmentation with user input.

Soft edges between the user's hands and the object he holds can sometimes affect the results of the segmentation, especially when the object presents a similar tonality to the user's skin.

A possible way to improve watershed results could make use online training to build a model based on previous segmentations. Derivaux et al. (2010) uses a similar method by using fuzzy classification to build a greyscale map and applying the watershed algorithm on that map. This would of course incur an additional computational cost.

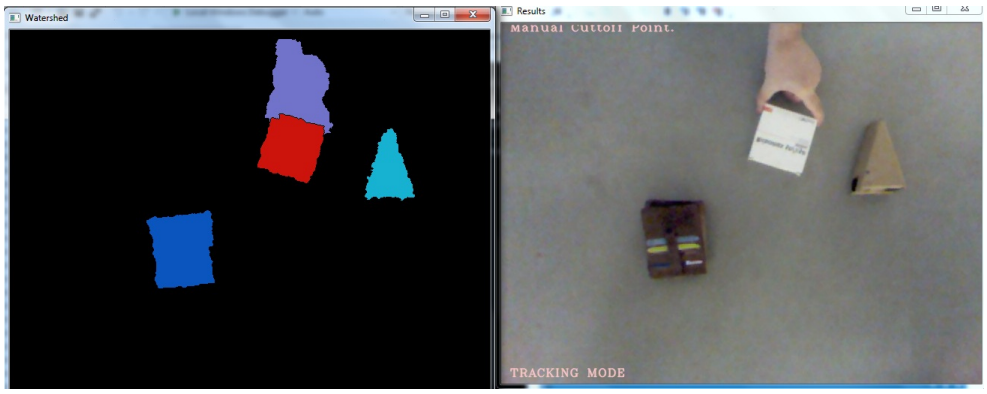


Figure 36: Watershed Sample.

The distance between the camera and the workspace can be adjusted, shortening the distance to provide better quality when segmenting objects of small size, or increasing it when dealing with big objects to obtain a bigger interaction area.

This does not mean that the system runs without issues however, with some problems born from limitations of the used methods, others inherent to the chosen hardware itself. One of the

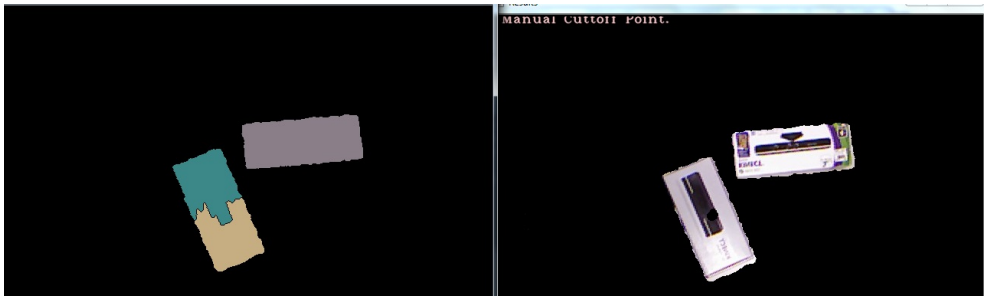


Figure 37: Watershed over-segmentation due to artifacting.

requirements of the framework is that the camera be set at a perpendicular angle to the surface of the interaction area. Too big of a deviation will degrade the quality of the segmentation, or even causing it to fail entirely. This is due to the fact that, when the camera is at a perpendicular angle to the workspace, all of the surface's pixels will share similar depths, barring small discrepancies related to noise.

This method creates a segmentation plane that runs parallel to the surface, discarding all depth values above it. Changing the angle of the camera will change the perceived depth values of the surface, and the segmentation plane will start to intersect the interaction surface, causing regions past the intersection point to start appearing in the results. An example of this issue can be seen in Figure 38 where, due to the angle of the camera, the segmentation plane runs too close to the upper part of interaction surface. Noise created by reflecting light leads to errors in depth values of small regions, that end up intersecting the segmentation plane and showing up in the final mask.

A possible solution to deal with slightly oblique camera angles could be the use of a depth model of the surface. During the setup phase, the system could build a map of medium depths registered by the sensor, and use it later to segment the background from the foreground objects.

The depth data that the Kinect camera provides is also very noisy. This is especially noticeable near the borders of objects. Transparent and reflexive materials can also present problems, leading to loss or inaccuracy of depth information, and even flat surfaces provide uneven depth values. The problem worsens when the distance between the camera and the surface increases. This can prove to be an issue when using very thin objects, as they may not be detected correctly when laid on the interaction surface due to inaccurate segmentation. Figure 39 shows some examples of loss of depth information when using this camera.

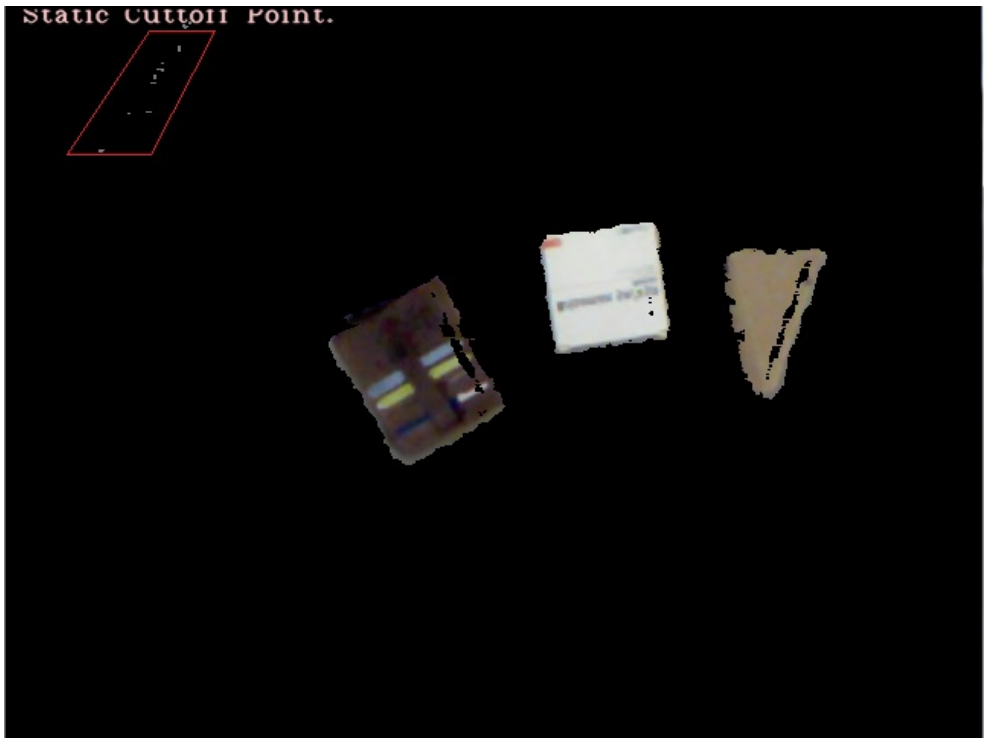


Figure 38: Example of a segmented image. Marked in red error caused by a non perpendicular angle of the camera with the surface.

4.2 OBJECT TRACKING

The application presents suitable results when tracking multiple distinct objects. On the tested hardware setup the application demonstrated that is able to perform very fast, which is perfect for a real-time interface that needs to respond to user input. It maintains a steady execution speed of around 60ms per frame, most of which spent on preprocessing methods, suffering little penalty from increasing the number of tracked objects. Up to eleven object were simultaneously used without experiencing a too noticeable performance decrease. This clearly demonstrated the ability to implement such a system even in low end / low cost machines. Figure 40 and Figure 41 show two examples of multiple object tracking using this system.

The framework is currently not optimized, so further improvements could still be achieved. As an example, taking advantage of multiple processor cores and parallel processing or using the graphics processing unit could lead to better results, and it is a consideration to keep in mind for future work.

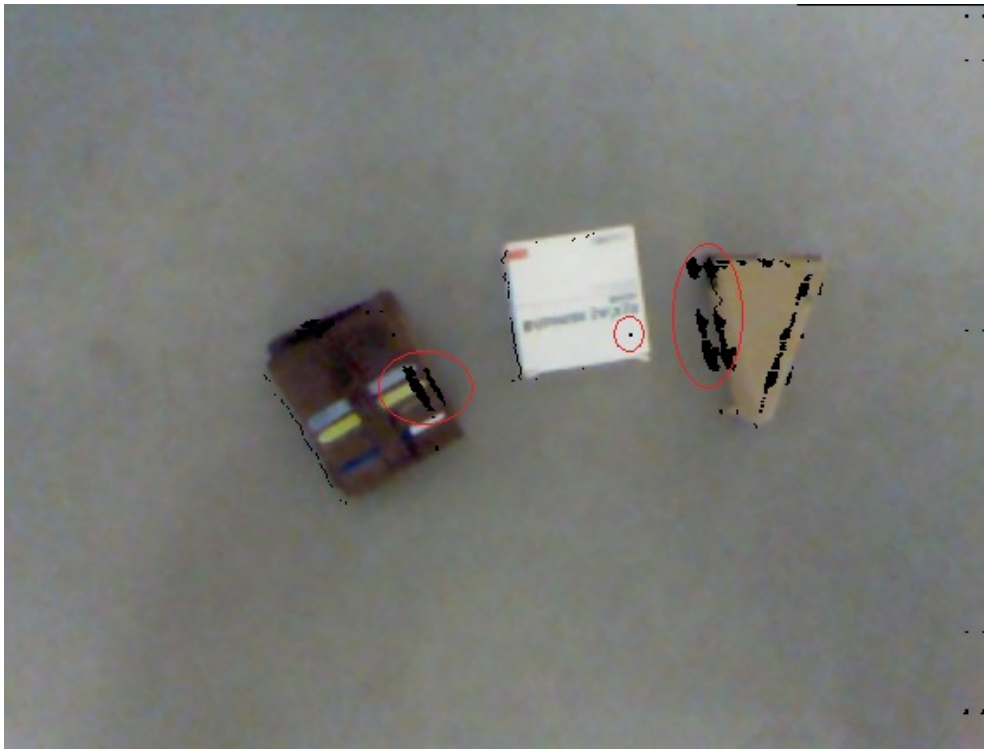


Figure 39: Kinect Errors: Image generated by combining color and depth information. Red circles indicating loss of information

The framework can recover well from short and long term occlusion, entrance and exit from the object in scene, simultaneous occlusion of multiple objects and occlusion of objects during movement. It can also recover from segmentation errors when the user is holding an object. These aspects, when combined with the simplified object registration, makes it easier for users to work with, and also allow users to freely hold objects without worry of disrupting the system.

As seen in Figure 42 The presence of objects not registered on the system is simply ignored, and does not affect the tracking process.

The tracking of objects with similar appearances remains a problem, though limited success was achieved by maintaining the objects far apart and turning off the exhaustive search of blobs that is executed when the Kalman filter fails.

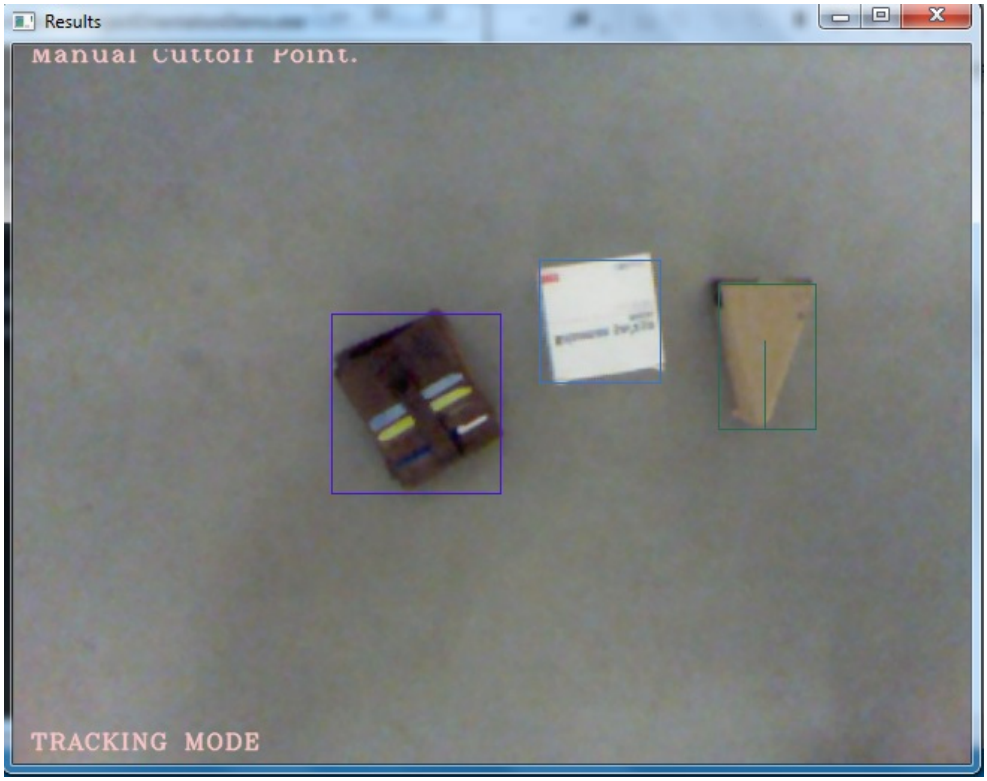


Figure 40: Tracking Sample

4.3 OBJECT ORIENTATION

All implemented methods to determine an object's rotation present reasonably accurate results with only a small error margin (refer to subsection 3.3.5 for details). Shape based rotation tracking is very fast and is constantly calculated, but requires objects with distinctive irregular shapes.

Since this method executes in real time, small divergences with the contour detection and depth noise near the object's borders can cause interference with the calculations, causing the angle to constantly shift even when the object is not moving. The severity of this problem differs from one object to another, being more accentuated in objects with sharp corners. The method used to automatically identify if an object is valid for this algorithm can also fail due to the same reason. Figure 43 shows the shape orientation tracking method in action.

Texture based rotation tracking can be used independently of the object's shape, but requires a texture rich enough that at least four good matches are found, so that the homography matrix can be calculated. Since the algorithm is computationally expensive, it is only performed when any sort of movement is detected in the object's area. This can lead to erroneous values resultant from

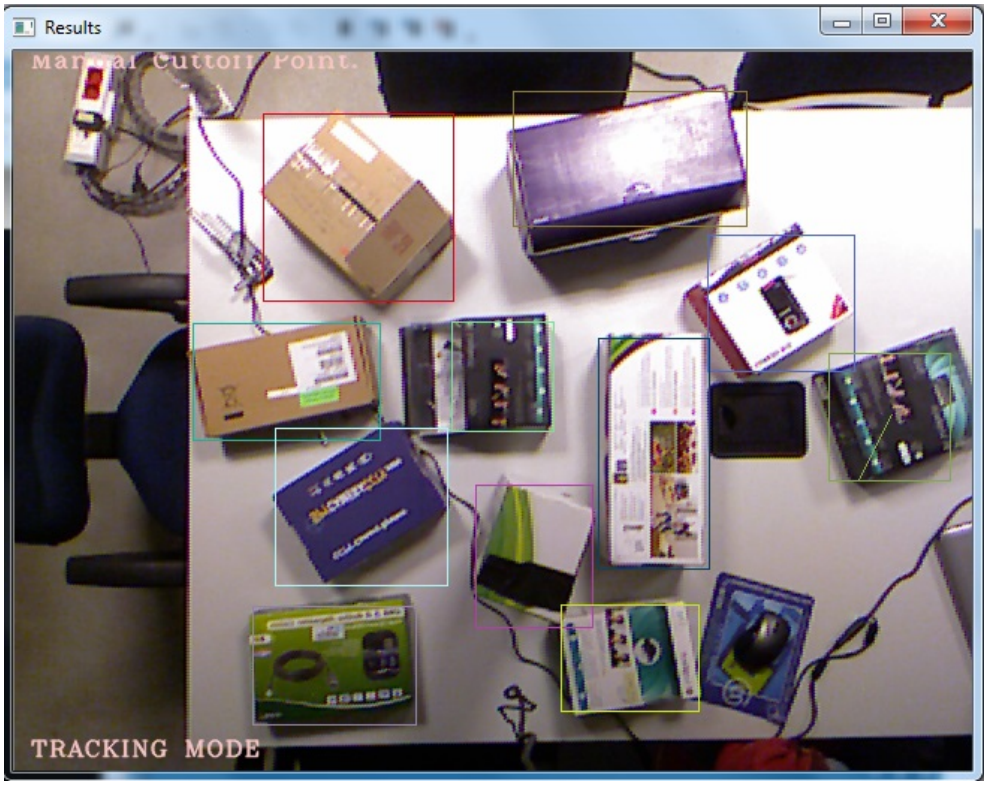


Figure 41: Tracking Sample with 11 objects

bad homography calculations during partial occlusions or drastic illumination changes. Figure 44 shows this method in practice.

The method that uses ellipse fitting to calculate orientation works in a similar manner to the first method. It can only detect rotations up to 180 degrees, but it is very fast to calculate and, when using adequate objects, can be very accurate. Objects whose height and width ratios greatly differ, like rectangle boxes, pens or screwdrivers, produce the best results. Figure 45 illustrates this method.

In order to test the quality of the results obtained from each method, 24 frames of two objects were collected. For the shape based and ellipse fitting methods, the cardboard triangle object was used. For the texture based method, the *Kinect 360* box was selected. Each frame was rotated by fifteen degrees at a time, performing a full 360 degrees rotation in the last frame. Finally, the absolute values of the differences between the true angle and calculated angle were averaged. Each test was performed ten times, using a different set of frames each time. The results can be observed in Table 1.

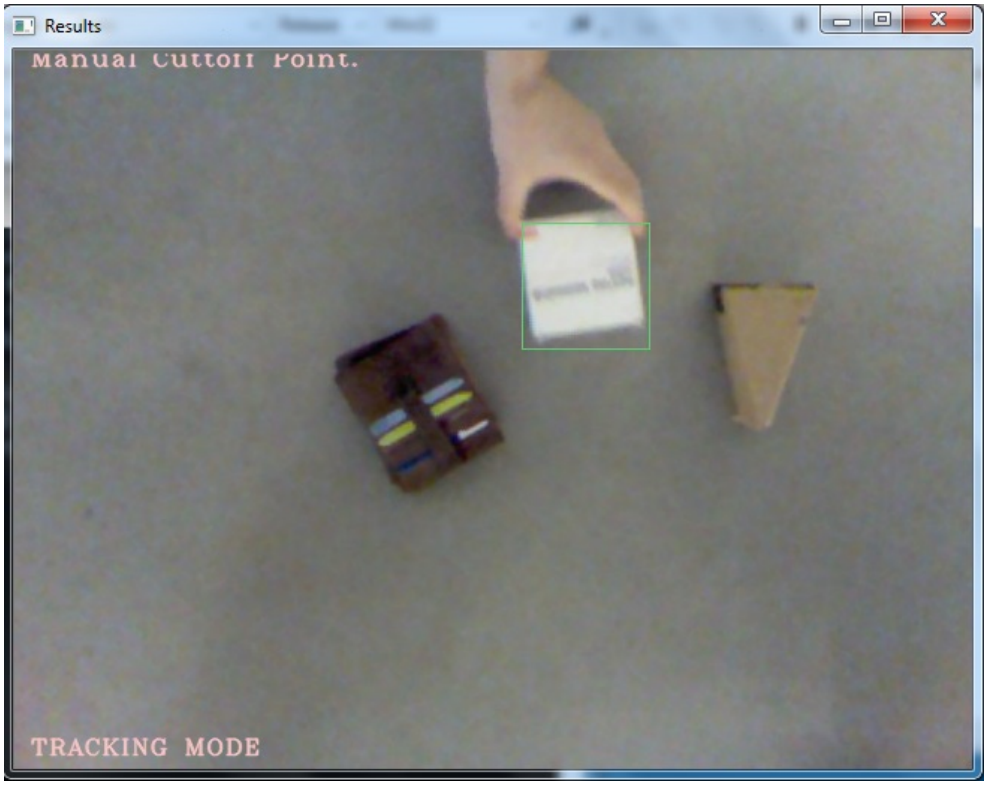


Figure 42: Tracking an object amongst two non tracked objects

The shape based method came ahead with a mean error rate of 2.20 degrees, but relies heavily on the shape of the object in order to function correctly. The Ellipse fitting method achieved the second best score, and is more forgiving on the type of objects it can work with, but can only detect rotations up to a limited angle. The texture based approach can be used for object whose orientation cannot be determined by shape, but requires that the objects possess somewhat diverse textures in order to function.

The developed methods can offer acceptable results when the goal is to understand the object's orientation, but struggle when a great deal of accuracy is required. Furthermore, while the three

Table 1: Measured error from the three alternative orientation algorithms

Orientation Results		
Method	Mean Error	Standard Deviation
Shape based	2.20^o	1.04^o
Texture based	8.24^o	14.16^o
Ellipse fitting	3.38^o	2.96^o

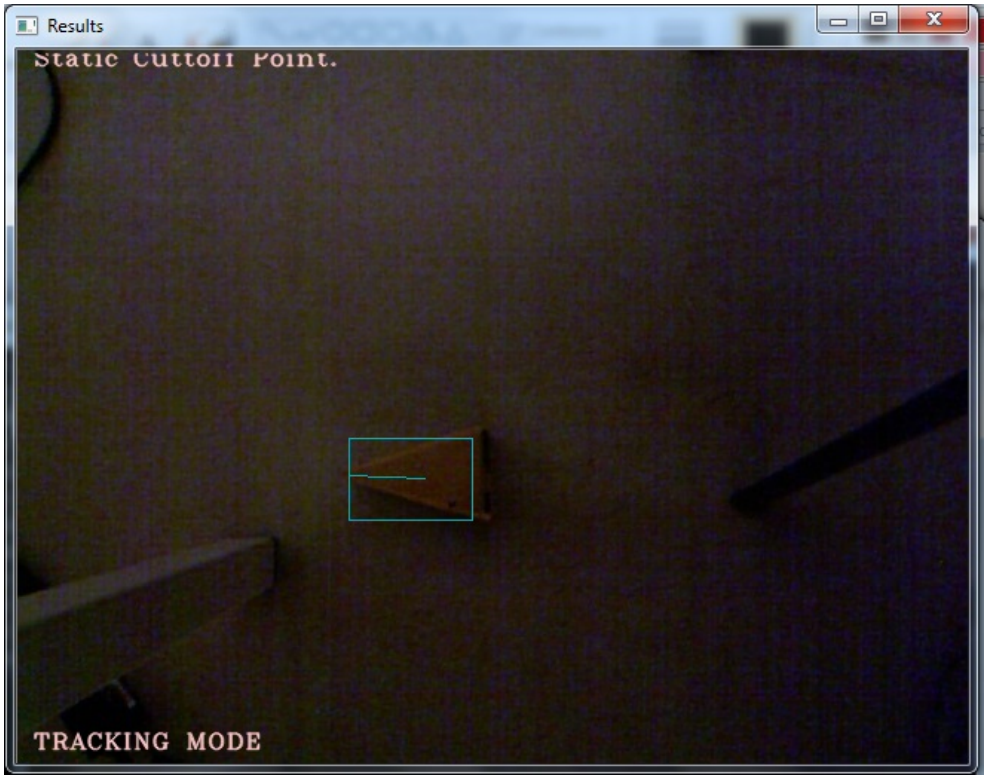


Figure 43: Tracking rotation by shape

methods work to cover a lot of object types, there are still many limitations present. The rotation of square or round texture-less objects for example, cannot be tracked. In future work, a more in-dept analysis of the problem will be required.

4.4 OBJECT TASKS

Aside from freely tracking the position and orientations of objects, the framework allows for several more specific tasks to be assigned to objects. These tasks can modify what information the system keeps track of in relation to an object, and what information to send to client applications. Currently, the available tasks are free tracking, sliders in all three axis, x,y and z, area tracking, linked tracking and dials.

Free tracking is the default task any object has, where the system will keep track of all information relative to an object. That includes all the x and y coordinates of the object, the object's height or z coordinate and its orientation. In Figure 46 can be seen an example of this task, with the x and y coordinates written on the console.

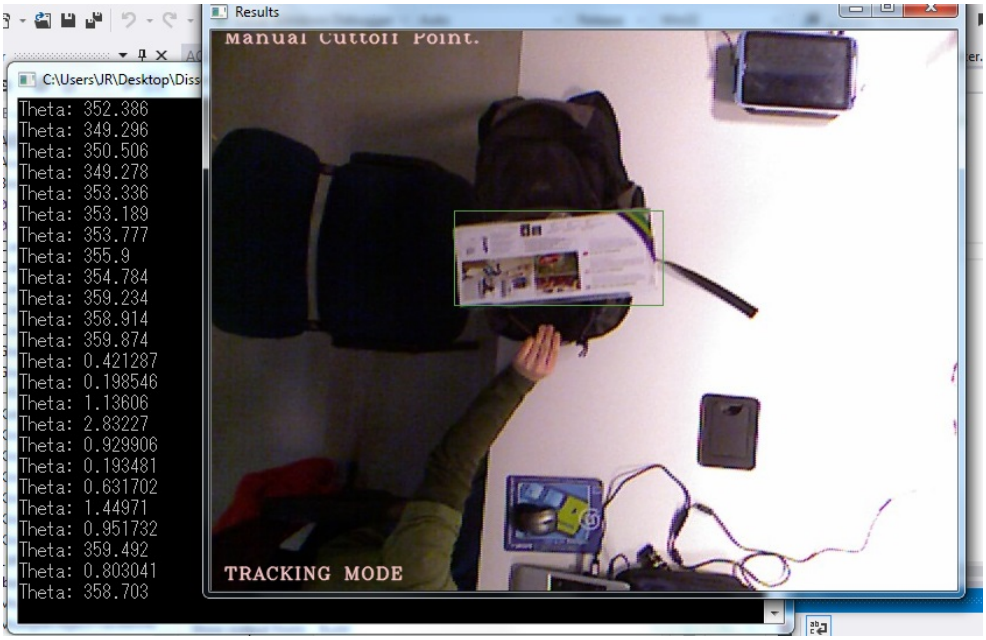


Figure 44: Tracking rotation by texture

Figure 45: Tracking rotation with ellipse fitting

A slider keeps track of an object between two positions. To create it, the user registers the object twice, one on each of the limiting positions, the minimum and the maximum. The system will determine the distance between those two points and calculate the relative position of the object accordingly. As an example, an y axis slider task has the two following points as limiters:

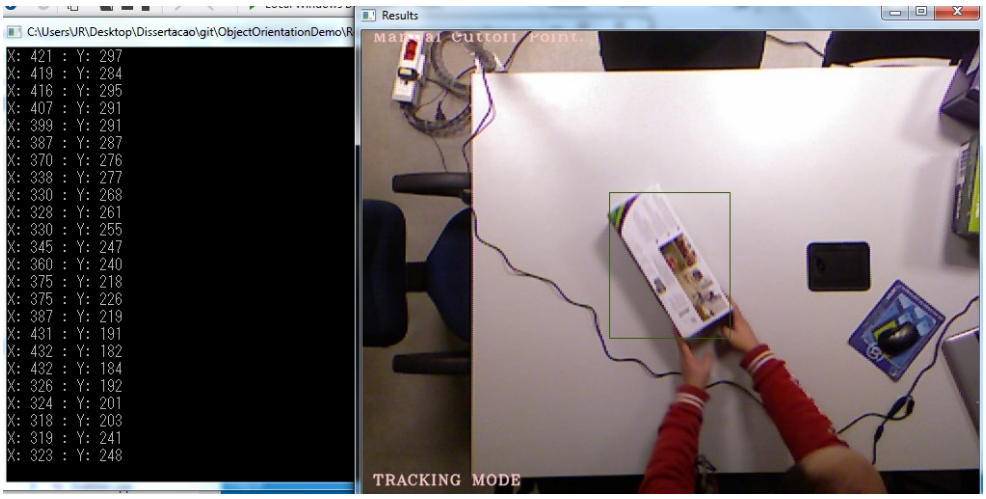


Figure 46: Free tracking task

$P_0(x_0, y_0)$ and $P_1(x_1, y_1)$. The current position of the object is $P_c(x_c, y_c)$. The system will produce a percentage value according to the following formula:

$$(|y_0 - y_c| / |y_1 - y_0|) \times 100$$

If the object is above the outside bounds, it will be ignored. Figure 47 shows a y axis slider at work.

Figure 47: Y axis slider sample

The z axis slider works in the same way, only it uses the object's height in its calculations. An example of this task at work can be seen in Figure 48.

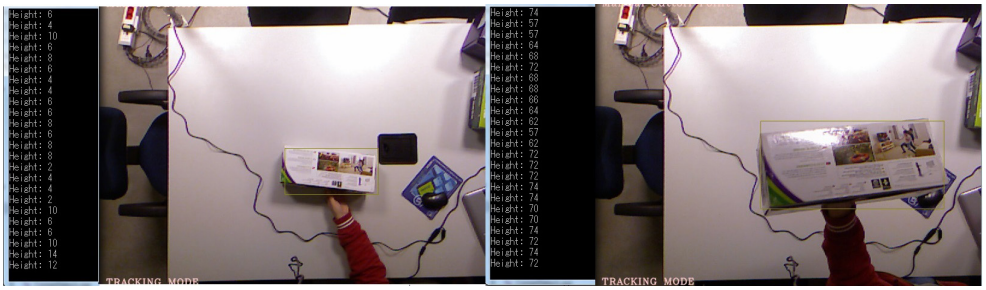


Figure 48: Tracking the object's height

Area tracking works in a similar way to free tracking, only the user can define a circular area where the object can be tracked. If the object is outside that region, any event will be ignored. To define that area, the user register the same object twice, one to define the center of the circle, and another time to define the length of the radius.

Linked tracking keeps note of the distance between two registered object. To define this task, the user register both linked objects one after the other.

The dial task will ignore an object's translation and keep note only of it's orientation. Figure 43 shows this task at work using the shape based method for orientation tracking.

4.5 EVENTS

The system uses UDP packets to send information related to events to client applications. All messages start with an id that refers to the object that produced that event. From there, the messages can either contain all the information concerning the object's position, or only the pertinent values according to the object's assigned task. The current system sends all information when dealing with free and area tracking, and only pertinent information when dealing with other tasks.

In free and area tracking tasks, the messages contain the object's id, the coordinates x,y and z of the origin point of the object, that is, where the object was first registered, the coordinates of the current position and finally, the object's orientation. Slider task messages contain only the id of the object and the calculated value of the slider. Linked tracking tasks add the origin points of both linked objects and the current distance between them, and dial tasks add only the object's current rotation relative to its original orientation.

4.6 DEMO: TANGIBLE PONG

To test the application in a practical scenario, a game of Pong was created in the Unity3D engine. Pong is a two player game in which each player controls a paddle that moves horizontally. A ball will move back a forth between players, bouncing off walls and paddles. The objective of the game is for a player to use his paddle to send the ball towards the other player's goal, scoring a point, while at the same time, protect his own goal.

The game accepts UDP packets from objects that have been registered as slider tasks. The first registered object will control the lower paddle, becoming player 1, while the second object will control the upper paddle, becoming player 2. Moving the objects across the interaction surface will cause the respective paddle too move accordingly (Figure 49).

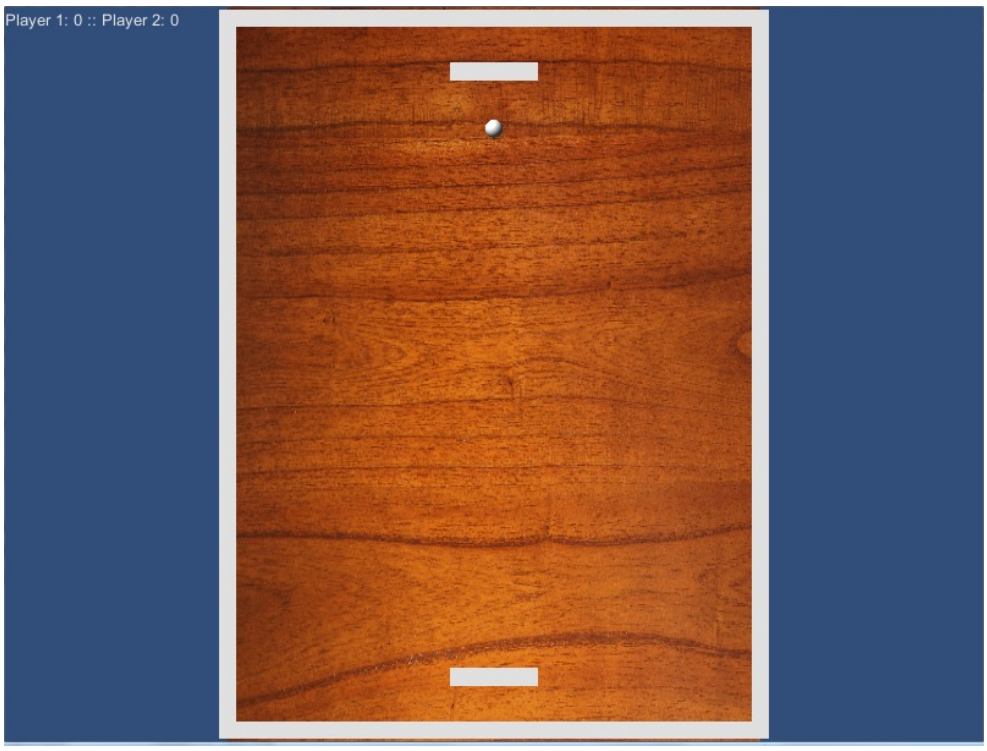


Figure 49: Tangible Pong: a demo application created to demonstrate some of the framework's capabilities.

Other options can be configured using tracked objects. A dial object can be used to rotate between different backgrounds for the board, and sliders of any type can be used to adjust the speed of the ball or the volume of the music.

As seen in Figure 50, two users can manipulate tracked objects as if they were directly moving the paddles of the game, which allows for a more immersive user experience.

Figure 50: Two users playing Tangible Pong using regular boxes as controllers.

4.7 SUMMARY

This chapter presented and analysed the operation and performance of diverse components of the implemented framework. Obtained results were presented and discussed. From the presented discussion main conclusions follow.

Segmentation of objects is fast to execute and produces good results being rather invariant to light changes. The presented implementation is limited in the fact that the camera needs to be fixed at a perpendicular angle to the surface, but benefits from the reduction of occlusions between objects, being able to see the entirety of the workspace and not needing complex models to represent objects.

The tracking can achieve an acceptable performance for an interactive interface, even on lower tier computers. Further improvements could be obtained by optimizing the code. When tracking

orientation, ellipse fitting provides very accurate results, but can only track a limited rotation angle. Texture based tracking achieves the second best score and can track a full 360^{o} rotation.

Different tasks can be assigned to objects in order to simplify the information sent to client applications. The system can keep track of the object's position on the 3D space and can also detect its 2D rotation.

An application was developed as a proof of concept demonstrator. At on hand, this demonstrator enabled to verify the capabilities of the framework in the context of a real application, on the other hand, the ability to easily integrate the framework in such applications.

5

CONCLUSION AND FUTURE WORK

TUIs are an interesting alternative to the classical UIs. They seek to bridge the gap between the physical and virtual worlds by allowing users to manipulate physical objects as substitutes for virtual data. Users can then use more natural gestures when interacting with data, granting a more immersive and enjoyable experience.

Despite the many TUIs that have been developed, they are not very widespread, and the main means of interacting with computers is still the GUI based on the WIMP paradigm. Most of the implementations studied in this dissertation are all oriented to solve very specific tasks, and require specialized equipment.

The goal of this dissertation was the design and implementation of a modular framework capable of creating a Tangible User Interface system, using a camera as the sensor and common everyday objects, such as cans or books, as controllers.

The proposed framework was designed in order to be easy to use, and to not be too intrusive to the user's actions, allowing for a smooth user experience.

The framework is able to accept a great variety of objects, and detect when users interact with them. Information generated from manipulating the objects can be clearly transmitted to other applications. The system allows for more specific tasks to be assigned to object, such as making them act as sliders or dials. This way, transmitted information can also be tailored for each task, simplifying the interpretation of data on the side of the client applications.

Integrating all the necessary requirements for this project proved to be a very challenging task, and no current method can offer robust solutions for the entire problem. As such, many computer vision algorithms to describe, detect, recognize and track objects were studied, in order to find the best approach to the solve the issue.

The implemented framework makes use of a RGB-D camera as the sensor and consists of five modules, camera, setup, detection, tracking and communication. The camera module receives color and depth data from the sensor, and the setup module applies preprocessing to the retrieved

data. The detection module detects and segments the objects from the background, the tracking module identifies and calculates their position, and the communication module transmits object data to client applications.

Depth segmentation is used to detect and separate the objects from the background. The depth of the surface can be detected automatically or manually adjusted by the user. Further processing is performed through the use of the Watershed algorithm, in order to separate connected blobs and isolate the detected objects.

Objects to be tracked are registered one by one in the system, and data relating to their shape is recorded. Color and shape descriptors were chosen for their invariance and for being fast to calculate and match. Hue and Saturation channels from the HSV color space are used to create the histogram that describes the object's color. Hu's invariant moments are used to describe the shape. A distance formula is used to calculate the quality of each match. A Kalman filter is used to generate guesses for the position of each object, and narrow the search area of the tracker. Should the Kalman prediction not be accurate, the last known position of the object will be checked. If the systems loses track of the object completely, it is possible to perform an exhaustive check to find the object.

Three different methods to detect the orientation of objects were also implemented. The first method analyses the contour of the object in order to find a recognizable point of interest. The object's orientation is defined from the line segment between that point and the center of mass of the object. The process is repeated every time the object is detected and the angle between the current and original lines are calculated to determine the object's rotation.

The second method relies on point detectors to find points of interest on the object. When the object is first registered in the system its interest points are also recorded. Afterwards, the homography matrix between the original and current set of point is calculated and its rotation component is extracted.

The final method uses least-square curve fitting to find the ellipse that contains the object, and then calculates the orientation of the ellipse. A disadvantage of this method is that it can only detect rotations up to 180^{o}.

Whenever the system detects that an object was moved, it will generate an message containing information pertinent to the object and the event. That message is then delivered through a communication protocol to any client applications connected to the system.

Overall, the final implementation of the framework manages to achieve acceptable results. It is simple to set up and use, and accepts a grand variety of object with no extra effort from the user's part. It uses only one sensor mounted above the workspace and does not require the creation of

complex object models for tracking. The object detection is invariant to light changes and can provide a clear segmentation between foreground and background elements.

The system can detect 3D translations and 2D rotations of multiple objects simultaneously, and is robust against diverse types of occlusion. This allows for multiple users to simultaneously make use of the interface, while manipulating the objects in a more natural manner. It can achieve an acceptable execution speed for an interactive system, even on lower tier computers.

Messages generated from events can be transmitted to connected client applications through the use of UDP sockets. It is also possible to assign specific tasks to each object in order to modify it's behavior and the type of events it generates.

The modular nature of the framework also allows it to be easily modified to work with different sensors, detection methods or communication protocols.

The framework presents a great potential for growth. It's ability to use general purpose objects as controllers, and serve as interface for many different types of applications are some of it's greatest strengths. The only product that offers similar functionality is *Aytole's AnyTouch* (Chapter 2, section 2.1). Their product can turn object's into tangible surfaces, offering similar possibilities to this framework. Since not much information concerning *AnyTouch* is available, a more detailed comparison of functionalities is not possible.

It is not without it's flaws, however, and the current implementation presents a drawback when dealing with objects whose only distinct features lie in the texture, sharing shape and color distribution similarities. Similar or even identical object can still be tracked, but will be fully reliant on the Kalman filter predictions. Turning off the exhaustive search, keeping a safe distance between objects and preventing the occlusion of moving objects can solve this problem. Since shape is an important descriptor, non-rigid objects are also not fully supported.

Object models could provide more solid descriptors for each object, solving the issue of using similar objects. The problem lies in building a robust model. It is a taxing task and does not guarantee complete accuracy. It also does not solve the problem of using identical objects.

Another issue lies in the detection and segmentation of objects. It requires a perpendicular angle between the camera and the interaction surface, and also suffers from limitations generated from the hardware itself, such as the inability to correctly detect transparent objects.

Changing the angle of the camera too much would require the use of object models, as the system would need to recognize objects seen from different perspectives. The top down view the camera provides allows the system to only work with one perspective of the objects, which is also the reason that only 2D rotations are supported.

Another issue tied to this limitation lies in the segmentation of the background. The current method of depth segmentation produces robust results at a very low computational cost. Changing the camera angle would invalidate this method, and would require the use of more complex and computationally expensive methods, such as plane detection.

To test and illustrate the capabilities of the framework, a proof of concept application demonstration consisting of a tangible game was developed. Tangible Pong allows the player to control the paddles of a Pong game using normal everyday objects. The game's options can also be linked to tracked objects, such as changing the board's texture or the sound volume. This application can also demonstrate a multi-user experience. Assigning a dial task to an item and using it to control the sound volume can exemplify the ability to track rotation, while a registered item with a slider task could control the speed of the ball.

5.1 FUTURE WORK

Despite the presented contributions, several issues were identified in the current implementation of the framework that would require further work, in order to enhance its robustness and provide more adaptable solutions for a wider range of TUI applications.

An important progress would lie in making the system completely independent of the sensor's type and position. Making the system work without depth data would mean that any common RGB camera could be used as a sensor. Expanding on methods to deal with different object perspectives would also greatly benefit the framework.

A study of adaptive descriptors could also offer advantages for the system, namely when dealing with non-rigid objects, or workspaces with complex light conditions.

The current methods to detect object rotation still present some limitations. It would prove useful to develop more robust methods to solve this issue. Methods that can offer greater accuracy or accept more object types are of particular interest.

Computational optimization could also be made in order to improve the system's performance. Taking advantage of parallel processing or the graphics processing unit, could greatly improve execution speed.

Expanding the number and type of tasks that can be assigned to object to objects would also be interesting.

Different means of communicating with client applications could also result attractive prospects.

Another attractive concept would be the creation of sessions, allowing users to save the current status of the system in order to continue at a latter time.

Combining data from multiple sensors could also produce interesting results. A room full of sensors could mean that every object inside could become a tangible controller, including the user himself.

Above all, TUI is a very interesting concept. This work attempted to contribute to the field, but a lot of work is still needed, as many problems still need to be solved. On the other hand, there is also a vast amounts of ground for expansion, and some of the most recent advances in technology open some very interesting possibilities. As an example, the increasing computational power of mobile devices and the creation of glasses with attached cameras, such as *Microsoft HoloLens* or *Google Glass* or similar projects, could project this framework towards the realm of mobile platforms.

REFERENCES

Ayotle and Digital Labs. AnyTouch, 2013. URL http://digitaslabsparis.com/post/28555383766/anytouch. Date Accessed: 2015-01-16.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 15–22. ACM, 2004.

Eran Ben-Joseph, Hiroshi Ishii, John Underkoffler, Ben Piper, and Luke Yeung. Urban simulation and the luminous planning table bridging the gap between the digital and the tangible. *Journal of Planning Education and Research*, 21(2):196–203, 2001.

Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. *OPTICAL ENGINEERING-NEW YORK-MARCEL DEKKER INCORPORATED-*, 34:433–433, 1992.

Stan Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 232–237. IEEE, 1998.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

Wen Chen, Yun Q Shi, and Guorong Xuan. Identifying computer graphics using hsv color model and statistical moments of characteristic functions. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1123–1126. IEEE, 2007.

Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

Matthew Cottam and Katie Wray. Sketching tangible interfaces: Creating an electronic palette for the design community. *Computer Graphics and Applications, IEEE*, 29(3):90–95, 2009.

Daniel Cremers and Christoph Schnörr. Statistical shape knowledge in variational motion segmentation. *Image and Vision Computing*, 21(1):77–86, 2003.

Sébastien Derivaux, Germain Forestier, Cédric Wemmert, and Sébastien Lefevre. Supervised image segmentation using watershed transform, fuzzy classification and evolutionary computation. *Pattern Recognition Letters*, 31(15):2364–2374, 2010.

Chen Feng, Yasuhiro Taguchi, and Vineet R Kamat. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6218–6225. IEEE, 2014.

Enrique J Fernandez-Sanchez, Javier Diaz, and Eduardo Ros. Background subtraction based on color and depth using active sensors. *Sensors*, 13(7):8895–8915, 2013.

Andrew Fitzgibbon and Robert Fisher. A Buyer ' s Guide to Conic Fitting. *British Machine Vision Conference*, pages 513–522, 1995.

George Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449. ACM Press/Addison-Wesley Publishing Co., 1995.

Xiang Gao, Terrance Boult, Frans Coetzee, and Visvanathan Ramesh. Error analysis of background adaption. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 503–510. IEEE, 2000.

Osian Haines and Andrew Calway. Recognising planes in a single image. *Pami*, 37(9):1849–1861, 2014.

José-Juan Hernández-López, Ana-Linnet Quintanilla-Olvera, José-Luis López-Ramírez, Francisco-Javier Rangel-Butanda, Mario-Alberto Ibarra-Manzano, and Dora-Luz Almanza-Ojeda. Detecting objects using color and depth segmentation with kinect sensor. *Procedia Technology*, 3:196–204, 2012.

Ming-Kuei Hu. Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187, 1962.

Hiroshi Ishii. Bottles: A transparent interface as a tribute to mark weiser. *IEICE Transactions on information and systems*, 87(6):1299–1311, 2004.

Hiroshi Ishii and Brygg Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 234–241. ACM, 1997.

Hiroshi Ishii, Craig Wisneski, Scott Brave, Andrew Dahley, Matt Gorbet, Brygg Ullmer, and Paul Yarin. ambientroom: integrating ambient media with architectural space. In *CHI 98 Conference Summary on Human Factors in Computing Systems*, pages 173–174. ACM, 1998.

Martin Kaltenbranner, Sergi Jorda, Gunter Geiger, and Marcos Alonso. The reactable*: A collaborative musical instrument. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*, pages 406–411. IEEE, 2006.

Kourosh Khoshelham. Accuracy Analysis of Kinect Depth Data. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-5/(August):133–138, 2012.

Daehwan Kim and Daijin Kim. A novel fitting algorithm using the icp and the particle filters for robust 3d human body motion tracking. In *Proceedings of the 1st ACM workshop on Vision networks for behavior analysis*, pages 69–76. ACM, 2008.

Mi Jeong Kim and Mary Lou Maher. The Impact of Tangible User Interfaces on Designers' Spatial Cognition. *Human-Computer Interaction*, 23:101–137, 2008.

Tian Li, Prabhat Putchakayala, and Mike Wilson. 3d object detection with kinect, 2012.

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

Wei-Ying Ma and BS Manjunath. A comparison of wavelet transform features for texture image annotation. In *icip*, page 2256. IEEE, 1995.

Rakesh Mehta and Karen Egiazarian. Rotated local binary pattern (rlbp)-rotation invariant texture descriptor. In *ICPRAM*, pages 497–502, 2013.

Microsoft. Microsoft PixelSense, 2012. URL http://www.microsoft.com/en-us/pixelsense/default.aspx. Date Accessed: 2015-01-16.

Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

Rahul Mishra, Mahesh K Chouhan, and Dr Dhiiraj Nitnawwre. Multiple object tracking by kernel based centroid method for improve localization. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(7), 2012.

Takayuki Nakamura. Real-time 3-d object tracking using kinect sensor. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 784–788. IEEE, 2011.

Katja Nummiaro, Esther Koller-Meier, and Luc Van Gool. An adaptive color-based particle filter. *Image and vision computing*, 21(1):99–110, 2003.

Timo Ojala and Matti Pietikäinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32:477–486, 1999.

Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

Jason Owens. Object detection using the kinect. Technical report, DTIC Document, 2012.

Youngmin Park, Vincent Lepetit, and Woontack Woo. Texture-less object tracking with online training using an rgb-d camera. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 121–126. IEEE, 2011.

Ben Piper, Carlo Ratti, and Hiroshi Ishii. Illuminating clay: a 3-d tangible interface for landscape analysis. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 355–362. ACM, 2002.

Julián Esteban Gutiérrez Posada, Elaine Hayashi, and Cecília Baranauskas. On feelings of comfort, motivation and joy that gui and tui evoke. In *Design, User Experience, and Usability. User Experience Design Practice*, pages 273–284. Springer, 2014.

Ivan Poupyrev, Desney Tan, Mark Billinghurst, Hirokazu Kato, Holger Regenbrecht, and Nobuji Tetsutani. Tiles: A mixed reality authoring interface. In *INTERACT 2001 Conference on Human Computer Interaction*, pages 334–341, 2001.

Dilip Prasad. Survey of the problem of object detection in real images. *International Journal of Image Processing (IJIP)*, 6(6):441, 2012.

Eric S. Raymond and Rob W. Landley. History: A Brief History of User Interfaces. In *The Art of Unix Usability*, chapter 2. 2004.

Donald B Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.

Jeremy Reimer. A History of the GUI, 2005. URL http://arstechnica.com/features/2005/05/gui/. Date Accessed: 2015-10-20.

Romer Rosales and Stan Sclaroff. 3d trajectory recovery for tracking multiple objects and trajectory guided recognition of actions. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

Khurram Shafique and Mubarak Shah. A noniterative greedy algorithm for multiframe point correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:51–65, 2005.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

Tom Simonite. Depth-Sensing Cameras Head to Mobile Devices, 2013. URL http://www.technologyreview.com/news/519546/depth-sensing-cameras-head-to-mobile-devices/. Date Accessed: 2015-01-31.

Luciano Spinello and Kai O Arras. People detection in rgb-d data. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3838–3843. IEEE, 2011.

Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

Christiane Ulbricht and Dieter Schmalstieg. Tangible augmented reality for computer games. In *Proceedings of the Third IASTED International Conference on Visualization, Imaging and Image Processing*, pages 950–964, 2003.

Brygg Ullmer and Hiroshi Ishii. The metadesk: models and prototypes for tangible user interfaces. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 223–232. ACM, 1997.

Brygg Ullmer and Hiroshi Ishii. Emerging frameworks for tangible user interfaces. *IBM systems journal*, 39(3.4):915–931, 2000.

Brygg Ullmer, Hiroshi Ishii, and Dylan Glas. mediablocks: physical containers, transports, and controls for online media. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 379–386. ACM, 1998.

Greg Welch and Gary Bishop. An introduction to the kalman filter. 2006. *University of North Carolina: Chapel Hill, North Carolina, US*, 2006.

Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780–785, 1997.

Alper Yilmaz, Xin Li, and Mubarak Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(11):1531–1536, 2004.

Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM computing surveys (CSUR)*, 38(4):13, 2006.

Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.