Ain Shams University
Faculty of Computers and Information Sciences
Scientific Computing Department – 2008

Team Members:

- Alaa Shaker CS
- Roaa Mohammed CS
- Mahmoud Osama SC
- Mohammed Fouad SC

Supervisors:

- Prof. Dr. Mohammed Fahmy Tolba
- Dr. Ashraf Saad Hussein
- TA. Mahmoud Hossam

[Touchwork]

A Multi-touch Applications Development Framework

Acknowledgment

At the very beginning, we would like to thank and direct our major gratitude to **Allah**, Who has granted us with His blessings all through the time of developing this project.

We would like to thank **Prof. Dr. Fahmy Tolba** for his time, his encouragement and for putting such faith in us and in our work, **Dr. Ashraf Saad** for his motivation, his guidance and his contribution to our education, and teaching assistant *Mahmoud Hossam* for his support throughout the project.

Moreover, we would also like to thank a number of people, without them we wouldn't have completed the project; *Dr. Hassan Ramadan* for his support, our dear *teaching assistants* for their constant help, *Eng. Tarek Hafez* for the housing, *Eng. Hisham Osman* and *Mr. Mohamed Mostafa* from *EMS* for the projector, *Eng. Hisham Bahgat* for the webcam, *Eng. Magdy Hosny* for the circuit design, the kind people working at *Al-Maamoun* for the electronic components, *Spiroplast* and *Mr. Salah El-Garhy*'s workshop for the acrylic and its polishing. We would also like to thank ITAC (ITIDA), as well as *Essam Salah* and *Mihan Samy* from *ITWorx*.

We would like to thank all our colleagues, friends and teachers in the department for their support. Finally, we want to express our gratefulness to our parents and families for their endless love, support, encouragement, patience and self-sacrifice.

Abstract

Human-Computer Interaction has shifted our computing experience to a new level. Nowadays, environments have turned clearly to more of the WYSIWYG concept, demanding new ways of computing; new visions, new concepts, new technologies, and new applications – a revolution.

This revolution has introduced us to the era of multi-modal input devices, where the user can directly interact with the system in powerful, intuitive scenarios. Most research in human-computer interaction field is now oriented about moving the computing experience beyond conventional input devices, as they mouse and the keyboard. As a result, multi-touch screens have taken a good share of this revolution, as seen in Apple's iPhone and Microsoft Surface Computing. Consequently, the developers' community is required to supply the users with application and environments that utilize such technologies. Here, Touchwork comes into play.

Touchwork is a development framework that is used to develop applications for multi-touch input devices. It applies computer vision through processing and handling user-input from multi-touch input devices such as FTIR multi-touch screens, and acting as a middle layer between the input device, and the user interface. It provides a wider scope of user-input through the usage of different input modes and gestures. Using Touchwork facilitates developing much more intuitive environments, providing countless, creative solutions for human-computer interaction.

In this documentation, we present how such a framework was built, its usage and applications, as well as the process of building a hardware prototype of a multi-touch screen.

Table of Contents

List of Tables and Figures

List of Tables

List of Figures

Part

I

[An Overview]

Introduction to Touchwork

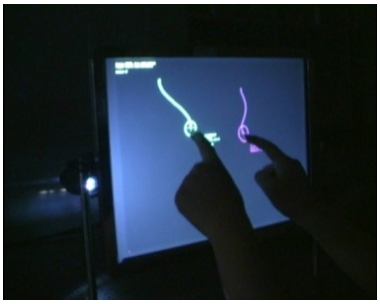
Chapter 1: Introduction

1.1. Motivation

Multi-modal input is a step further beyond traditional input devices such as keyboards, mice, sensors and microphones. Most of the traditional input devices suffer from limitations as with using a mouse that only offers a single-point visual input. Having more than one point of manipulation at the same time provides more flexibility and control such as in multi-modal tablets.

Multi-touch interaction gives the users a face-to-face chance to collaborate and work together through direct interaction using several touch points at the same time, without the use of the mouse or keyboard. Moreover, direct manipulation with the addition of using gestures can even provide more intuitive interaction (especially in educational software, CAD applications, virtual reality interfaces, 3D visualization in different fields like medical visualization, etc.).

1.2. Background



"This is really the way we should be interacting with the machines."
– Jefferson Han, FTIR Multi-touch Tabletop, TED-2006

The limitations faced due to the single-input constrains while using the mouse as a conventional input device has lead to the evolution of multi-modal input-devices, starting with pen-tablets and single-touch screens, to Wacom's advanced wireless multi-pen tablets and Microsoft Surface Computing.

Several techniques have been used to build multi-touch screens such as using magnetic-fields, sensors, infrared-reflection and more. Jefferson Han, a research scientist working out of New York University's Courant Institute, introduced a fascinating new vision of "*the way we should be interacting with machines*" at TED-2006[1]. Han brought up prototype of a multi-touch tabletop built using the FTIR (Frustrated Total Internal Reflection) technique, eliminating all the limitations previously faced, and opening a brand new scope of creativity to produce more interactive and intuitive computer applications.

This kind of interactivity could be used not only for simple ZUI (Zoom User Interface) applications, to scatter around, grow, shrink and rotate images and

[1] **TED** (Technology, Entertainment, and Design) is an invitation-only event that started in 1984, where the world's leading thinkers and doers gather to find inspiration. [http://www.ted.com/]

videos, but could be of further practical use as in educational software, CAD applications, 3D-visualization, virtual reality applications and far much more, where your limit is the sky.

Touchwork used computer vision to process the captured information and transform it into input points, in a way that will enhance the way we deal and interact with applications and environments. It will also facilitate developing custom-tailored applications based on using multi-points for input, breaking the boundaries of conventional applications and making them much more useful and interactive.

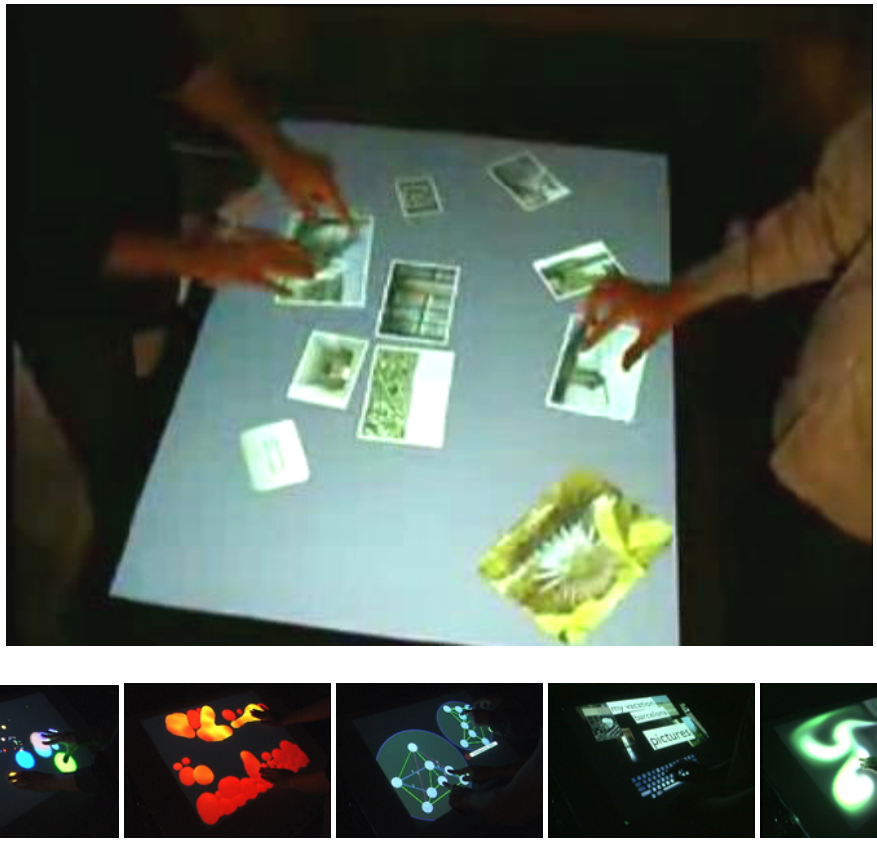


Figure 1 - Examples of multi-touch interaction

1.3. Problem Definition

With the emergence of the technology of multi-touch interfaces, a new way to develop applications was needed; software that helps interface developers to harness the power of this technology without worrying about the hardware details.

Since the technology is new, such a solution should conform to protocols that current developers are familiar with, these protocols should be backward compatible yet open to change and modification, allowing a very fast adoption rate with small changes applied to the existing interfaces. Existing solutions were either tightly bound to a certain hardware, not backward compatible, very hard to learn, or not extensible enough. **[Chapter 8]**

Hence, we decided to build Touchwork.

1.4. Objectives

- **Intuitive, interactive interface:** Introduce a new way of handling objects on the screen and manipulating computer applications.
- **Direct interaction:** Users can interact with content directly using touch and gestures, without any use of mouse and keyboard.
- **Development framework:** Provide a cross-platform development framework for developing applications for multi-touch input devices as touch screens based on FTIR.
- **GUI manipulation:** Facilitate the manipulation the Graphical User Interface (GUI) of conventional external applications and environments.
- **Processing multiple inputs:** Not only one touch point as usual touch-screens, but several touch points could be processed at the same time.
- **Tracking multiple input points:** Allows multiple users to gather on the same touch-panel, providing a collaborative, face-to-face computing experience.
- **Basic gesture recognition support:** An extensible gesture library.
- **Input device calibration.**
- **Build a simple hardware prototype.**
- **Develop a demonstration application.**

1.5. Framework Architecture

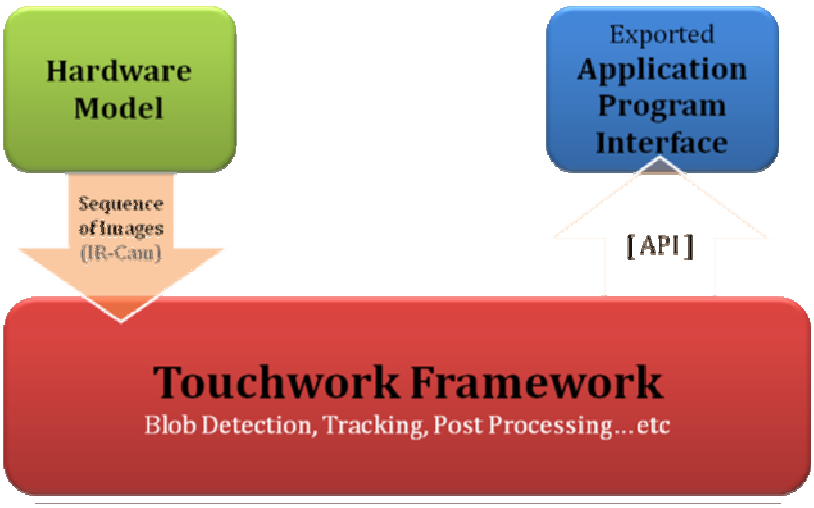


Figure 2 - General overview of the framework

The software system will be divided into six modules:

1. **Configuration Manager**
2. **Module Manager**
    a. **The Input Module** - twInput
    b. **The Detection Module** - twDetector
    c. **The Tracking Module** - twTracker
    d. **The Agent Module** - twAgent

Figure 3 shows the data flow between the framework modules, and how they interact. The modules twDetector and twTracker represent the core of the Touchwork framework.

1. **Configuration Manager:**
   Manages the persistence of user settings and preferences and helps the user in adjusting each module in order to make the framework operate efficiently on the Hardware and the user's environment.

   All the changes made by the user are saved in *Configuration Files* and are loaded whenever the framework starts, and are applied to their respective modules.

2. **Module Manager:**
   The framework is composed of independent set of modules that interact with each other, to perform certain processing on the data flowing through the framework.

   The **Module Manager** is the building block of the framework that manages all the other modules, defines the data flow between them, and defines the basic interface that a module should follow in order to interact with the framework.

   Each Module can be replaced with another module that conforms to the same communication interfaces exposed to the *Module Manager*, as long as the new module performs the same task of the old one, aside from the internal details of either of the two modules.

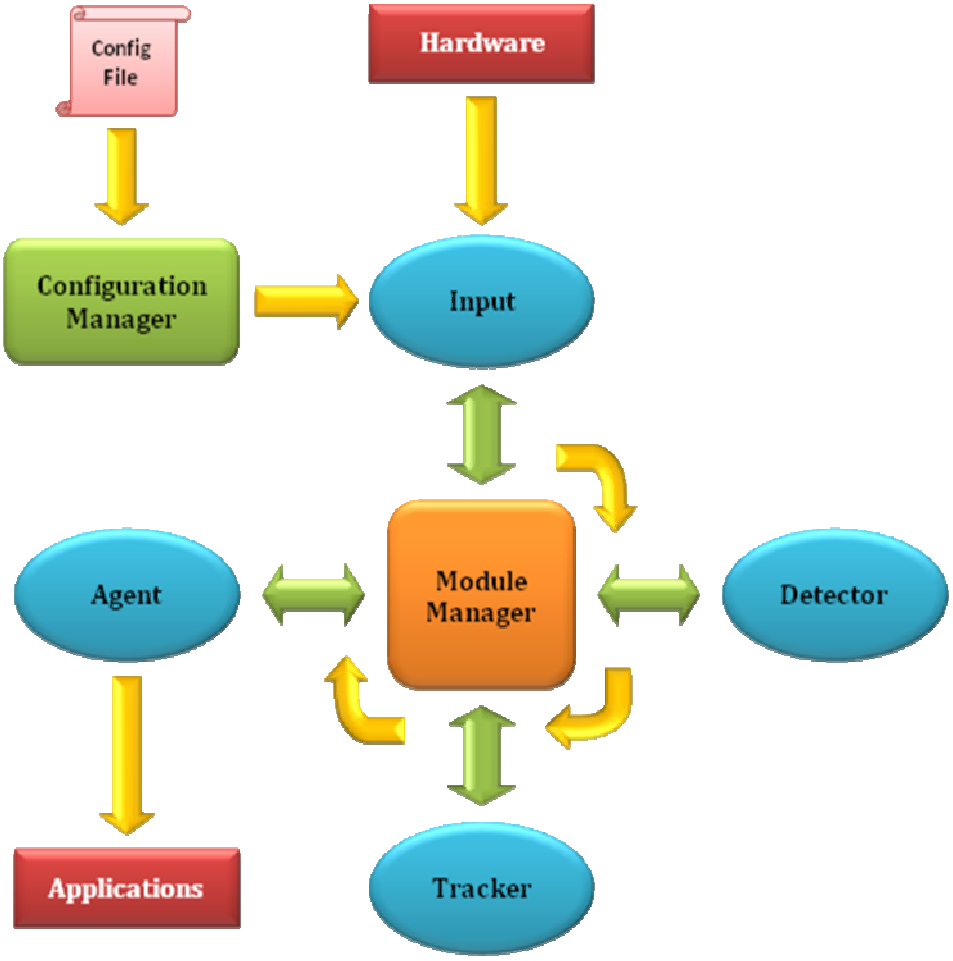


Figure 3 - Module Manager controlling data flow between the set of other modules

3. **The Input Module - twInput:**
   Captures video stream from the infrared camera as an input, and converts them into a sequence of images (frames) to be processed separately. It also handles the camera configuration and settings. Furthermore, it can apply the defined framework settings for input device calibration and settings as size of screen, tracking boundaries, etc.

   Chapter 3 contains more details about the Input Module, twInput.

4. **The Detection Module - twDetector:**
   Processes each frame in the received frame sequence by applying blob detection and image processing algorithms to identify all available blobs and prepare them for further processing. Component Labeling techniques are used for blob detection. Each frame is subjected to those tracing and image processing algorithms (as background separation), to help label and identify each blob in that frame on a color-based criteria; white blobs on black background. Computer vision techniques are applied to interpret the contact points as discrete touches or strokes, and gather the results in suitable data structures for later processing.

   Chapter 4 contains more details about the Detection Module, twDetector.

5. **The Tracking Module - twTracker:**
   Keeps track of the previously detected blobs. It will monitor the appearance, disappearance and motion of the blobs. Several tracking algorithms were tested for efficiency and were modified to reach satisfying results. This allows the tracking of multiple touch points simultaneously.

   Chapter 5 contains more details about the Tracking Module, twTracker.

6. **The Agent Module - twAgent:**
   Plays as the interface between the Touchwork framework and the above layer of developed applications, external applications, existing frameworks, and surrounding environment. The module will also provide basic event processing for applications; it will also expose the programming interfaces of the framework, acting as the Software Development Kit (SDK) for Touchwork.

   Chapter 6 contains more details about the Agent Module, twAgent.

1.6. Tools, Technologies and Libraries

Tools	• Microsoft Visual Studio .NET 2005 and 2008 • Sparx Enterprise Architect	• Tortoise SVN • Philips VLounge • WinMerge
Technologies	• C/C++, Managed C++ and C# (.NET)	
Libraries	• WinAPIs • Windows Template Library (WTL version 8.0) • OpenCV	• Active Template Library (ATL) • Microsoft DirectShow • Microsoft DirectX 9.0c

Table 1 - Tools, Technologies and Libraries

1.7. Milestones and Time Plan

Milestone 1: Prototype
A primitive working sample

Milestone 2: Framework I
Detection (Mature stage) – Tracking (Basic stage) – Agent (Basic stage)

Milestone 3: Framework II
Tracking (Mature stage) – Agent (Mature stage) – Gesture (Basic stage)

Milestone 4: Framework III + Demo Application I
Gesture (Mature stage) – Primitive Demo Application

Milestone 5: Framework IV + Demo Application II
Framework (Testing stage) – Demo Application (Mature stage)

Milestone 6: Release
Final Testing – Documentation Finalization

Milestones execution is shown in figure 4, the time plan for the project.

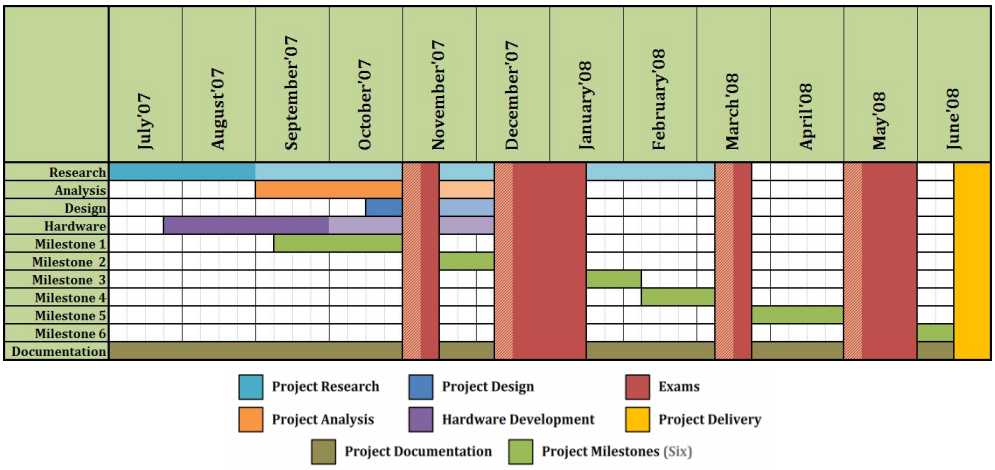


Figure 4 - Project Development Time Plan

1.8. Documentation Organization

This documentation is organized as follows:

- **[Chapter 1] Introduction:**
  Discusses the background, motivation, problem definition and objectives of the project, as well as the overall system description, milestones, time plan, tools, technologies, and libraries used in the development process.
- **[Chapter 2] The Hardware:**
  As the implementation of our project requires the presence of a hardware unit, a multi-touch screen, this chapter discusses the different hardware models and setups for building such screens, compares their differences, and shows the building process.
- **[Chapters 3, 4, 5, and 6] The Input Module, The Detector, The Tracker and The Agent:**
  Provides a more detailed description of the four modules shortly explained in Section 1.5. The chapter discusses the modules definition, role in the system, problems and tasks, possible solutions and alternatives, and the choice and details of its implementation.
- **[Chapter 7] Output:**
  In this part of document, we present our deliverables, test plans, results, conclusions, and more details about the output of this project.
- **[Chapter 8] Demos:**
  This chapter provides details about the demos implemented to demonstrate the Touchwork framework, as NASA World Wind.
- **[Chapter 9] Similar Work:**
  Other people are working on similar projects. A few are listed in this chapter; Microsoft Surface, Perceptive Pixel, Reactable, TouchLib and OpenTouch, AudioTouch and Tangible Table.
- **[Chapter 10] Future Work:**
  Our vision of the future development of Touchwork.
- **[Appendices A-H]:**
  UML specifications, analysis and design diagrams, tutorials, code implementations, team members, references and index.

Part

II

[Building the Hardware]

The Hardware Prototype

Chapter 2: The Hardware

2.1. Overview

We all use both the mouse and the keyboard in our daily interaction with computers. Both of them are typical examples of input devices. **Input devices** are hardware equipment used to provide a computer (or any other information processing system) with data or control signals. They play as an intermediate interface between the computer and the external world. In general, input devices are sensor that monitor, scan and accept information from that external world, whether the information is under the direct control of a user or not.

There are other conventional input devices other than those two, perhaps not as common, but provide different degrees of freedom. The most well known input devices are:

- **Mouse:**
  Is a small, hand-held pointing device that is pushed over a horizontal surface allowing the user to input spatial data to the computer, by moving it around and pressing one or more buttons. Enhancements include extra function buttons and wheels to perform various system-dependant operations.
- **Keyboard:**
  Is a peripheral that models a typewriter keyboard by providing an arrangement of buttons, or keys with characters engraved or printed on them. Once a user presses a key or more, a corresponding symbol is produced or a pre-defined action is fired.
- **Camera:**
  Is a device used to capture still photographs or a sequence of moving images – a video.
- **Microphone:**
  Is a sensor used to convert sound into electrical signals.
- **Scanner:**
  Is a device that optically scans two-dimensional objects as images, printed text, handwriting, etc., and converts it to a digital image. There are also 3D scanners that analyze real-world objects to construct digital, three dimensional models.

Nowadays, command line interfaces have been incredibly replaced with rich, graphical user interfaces. Here comes the important role of pointing input devices, such as the mouse, in our daily lives. However, the great mouse seems to suffer from some disadvantages:

- **Single point input:**
  The mouse can only offers an input of a *single* point at a time – spatial coordinates, P(x, y).
- **Indirect interaction:**
  The user moves the mouse around, which in turn moves a cursor around. This

results in the presence of an intermediate layer causing indirect interaction between the user and the system, making it not that easy to manipulate.

- **One user at a time:**
  A mouse can be controlled by only one user at a time. If there are several users, then they will have to exchange turns.
- **Gesture limitations:**
  Several software packages have tried to implement mouse gestures, but still they suffered many limitations due to the hardware capabilities.

As a result to those disadvantages, the evolution of the mouse began.

Evolution of Pointing Devices:

Nowadays, the WYSIWYG environment has pushed the evolution of pointing devices, as the mouse, to a different level. Computer mice now have more advanced functionality than before. The Digitized Tablet and Touchpad appeared later with wireless mice and styluses. Then, touch screens were developed, leading us to multi-touch surfaces and multi-modal devices.

So, what are **multi-touch devices**? They are devices that implement a human-computer interaction technique that allows the user to compute without conventional input devices (such as the keyboard and mouse) using multiple touch points simultaneously, opposing the standard touch screens that recognize only one touch point. (Examples can be multi-touch screens, tablets, walls, surfaces, etc.)

Multi-modal interaction devices provide the user with more than one mode to interact with the system, whether it was a visual modality (display, keyboard, mouse, touch, stylus, etc.) or voice modality (speech recognition for input, speech synthesis and recorded audio for output). Multiple modalities increase the usability of the system. So, by applying a form of human-computer interaction, the drawbacks or the weaknesses of one modality are offset by the strengths of another.

Figure 5 - Human-Computer Interaction Evolution

Advantages of Multi-touch Input Surfaces:

1. **Multiple input points:**
   The surface provides the ability to process visual manipulations through several touch points simultaneously.
2. **Direct interaction:**
   There isn't an intermediate layer that separates the user from the system anymore. Now, the user can directly interact with the system without any use of the mouse or keyboard.
3. **Better use of gestures:**
   Multi-touch surfaces facilitate much more intuitive interaction through the use of complex gestures.
4. **Multiple users working simultaneously:**
   It's no more a single user's world. The elimination of the single input device enabled more than one user to work at the same time.
5. **Face-to-face collaboration:**
   Surfaces, touch-walls, multi-touch screens, tablets, and more devices enable users to use, work together and experience the same system in a much more collaborative and productive environment.

2.2. Hardware Models

For the purpose of building our framework, we had to build our own hardware. To build the hardware prototype, we had to review several options and techniques, as Frustrated Total Internal Reflection, Diffused Illumination, Diffused Surface Illumination, TFT-LCD with Embedded Sensor, and Capacitance Screens.

2.2.1. Frustrated Total Internal Reflection (FTIR)

When a light ray crosses a boundary between two mediums of two different refractive indices[2], part of the light ray is reflected while the other part is refracted at the boundary surface. However, if the angle of incidence[3] of the light ray is larger than the critical angle[4] with respect to the normal to the surface, the light ray will be totally reflected back. This optical phenomenon is called **total internal reflection.**

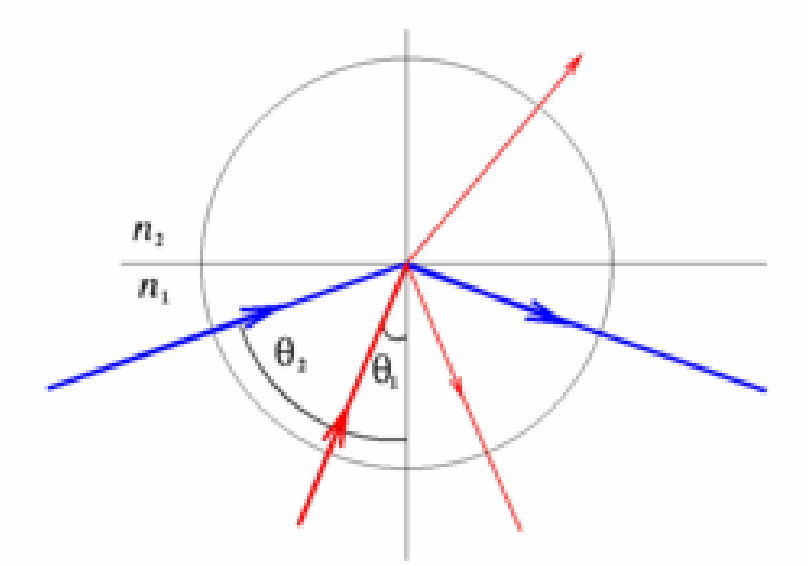


Figure 6 - Light Ray Refraction
Blue Ray: Entirely reflected ray
Red Ray: Partially reflected and refracted ray

In **Frustrated Total Internal Reflection** (FTIR), if a transparent, low refractive index material is sandwiched between two prisms of another material, this will allow the light ray to "tunnel" through from one prism to the next (very similar to quantum tunneling) while at the same time altering the direction of the incoming ray.

[2] **Refractive Index**: Is a measure for how much the speed of light (or other waves such as sound waves) is reduced inside the medium. The refractive index of a medium is defined as the ratio of the phase velocity of a wave phenomenon (such as light or sound) in a reference medium to the phase velocity in the medium itself.

[3] **Angle of Incidence:** Is the angle between a ray incident on a surface and the line perpendicular to the surface at the point of incidence, called the normal.

[4] **Critical Angle:** Is the angle of incidence above which the total internal reflection occurs.

In other words, imagine you have an acrylic sheet with a thickness of 8-10mm. If an infrared LED aligned normal to the side of the sheet (as shown in figure 7), all the emitted light rays that pass *into* the acrylic sheet that fall onto the inner surfaces of it with angles of incidence greater than the critical angle of that medium, will remain *trapped* inside the sheet through continuous reflection. Once a user touches the acrylic sheet, providing a different medium at the reflection boundary, the light will be scattered outside the sheet.

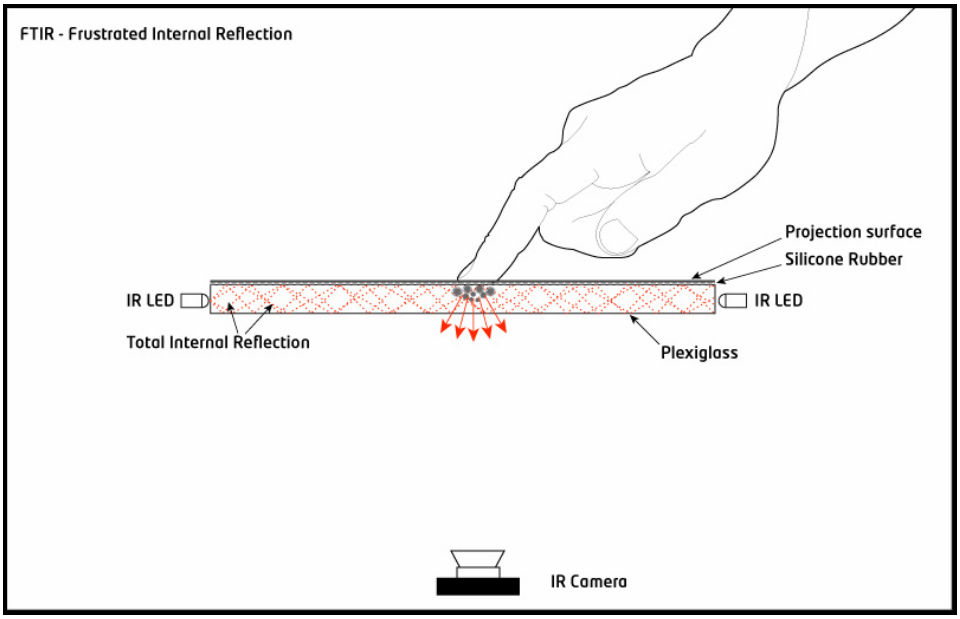


Figure 7 - Frustrated Total Internal Reflection (FTIR) Setup

When light encounters a medium with a lower index of refraction (e.g., going from glass to air), its refraction depends on the angle at which it hits the border. Beyond a certain critical angle, light is not refracted, but instead reflects entirely within the material (just as in fiber optics). When the medium is touched by another material (such as a finger-tip), the reflection is frustrated, causing the light to scatter and escape out of the medium.

2.2.2. Diffused Illumination (DI)

This model is based on **diffused reflection,** which is the reflection of light from an uneven or granular surface such that an incident ray is seemingly reflected at a number of angles.

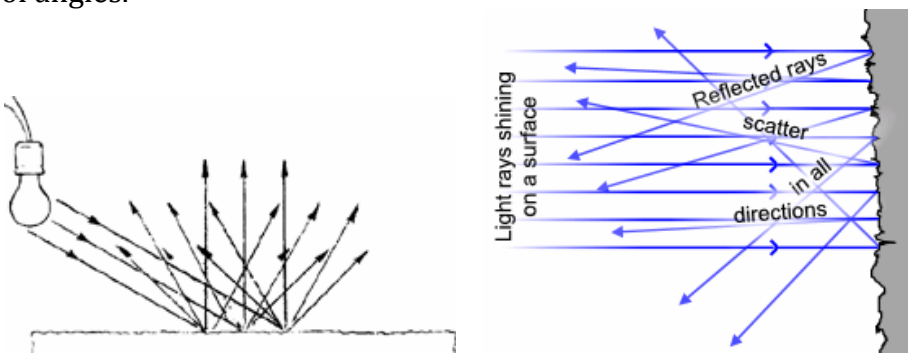


Figure 8 - Diffused Reflection

Incident rays reflect in a number of angles

Diffused Reflection is the complement to **specular reflection**. If a surface is completely nonspecular, the reflected light will be evenly spread over the hemisphere surrounding the surface (2×π steradians [51]).

In a **Diffused Illumination** (DI) setup, infrared light is shined at the screen from either below or above the surface. Infrared light is mounted underneath a glass or acrylic sheet with a diffuser. When an object touches the surface it reflects more light than the diffuser or objects in the background. The extra light is sensed by a camera. A diffuser can be tracing paper or a thin sanded glass sheet. Refer to compliant surfaces, *Building the Hardware*. **[Section 2.4]**

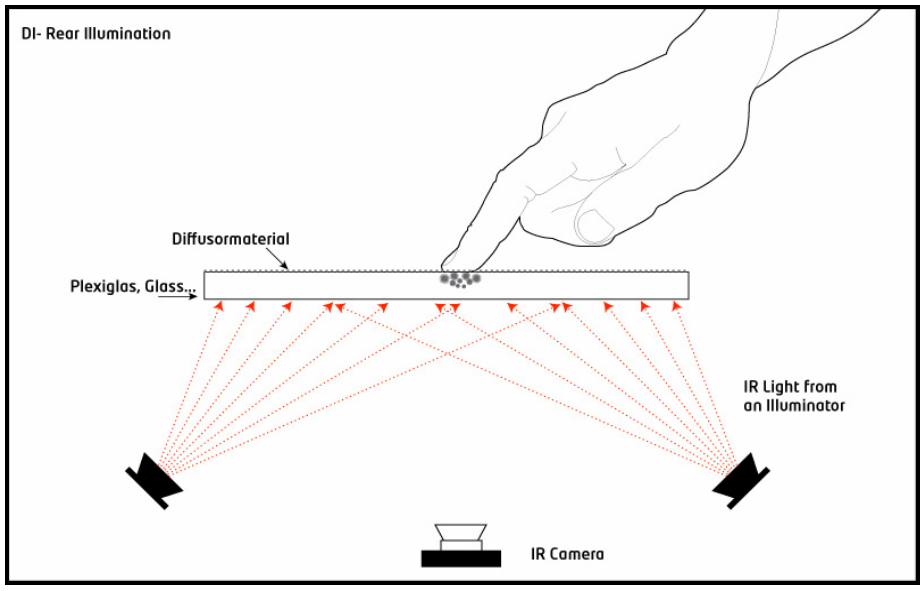


Figure 9 - Diffused Illumination (DI) Setup

[5] **Steradian (sr)**: A unit of measure equal to the solid angle subtended at the center of a sphere by an area on the surface of the sphere that is equal to the radius squared. It is used to describe two-dimensional angular spans in three-dimensional space, analogous to the way in which the radian describes angles in a plane. The name is derived from the Greek stereos for "solid" and the Latin radius for "ray, beam".

Depending on the diffuser, this method can also detect hover movements and objects placed on the surface. The diffuser lets light through, but not all of it. Therefore, the light that hits the diffuser is around 50% of the total light. When a finger touches the diffuser, the light can't pass through the finger and therefore 100% of the light hits the finger which then lights it up for the camera to see.

The main Problem with most DI setups is getting infrared light to spread evenly across the screen surface. Normally, we can use two or four infrared illuminators as an infrared source. However, it takes lots of effort to get the infrared even while dealing with possible reflections from the illuminators. Also, in DI setups, the resulting image requires lots of image processing to be suitable for detection.

2.2.3. Diffused Surface Illumination (DSI)

The **Diffused Surface Illumination** (DSI) model introduces a minor modification to the DI model and a variation of the FTIR method. It is also called "Diffused Screen Illumination".

Using a standard FTIR setup with infrared LEDs, this model just uses a special acrylic instead of using illuminators as an infrared source, which does not frustrate the infrared light from the edges and distributes it evenly across the surface. This acrylic uses small particles that are inside the material, acting like thousands of small mirrors. So when you shine infrared light into the edges of this material, the light gets redirected and spread to the surface of the acrylic. Also this acrylic sheet is thinner than that used in the FTIR Model.

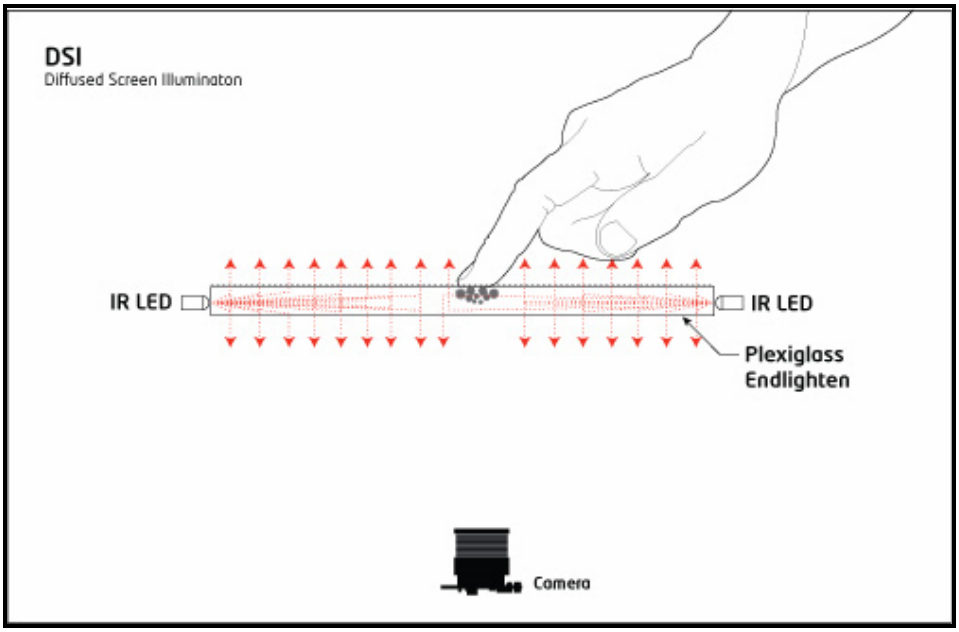


Figure 10 - Diffused Surface Illumination (DSI) Setup

2.2.4. TFT-LCD with Embedded Sensor

In this model, a **TFT-LCD** is presented with an embedded array of **a-Si photo** sensors, for use as a high performance touch-enabled LCD. In the shadow mode the optical touch screen operates with finger or stylus input under ambient lighting conditions from 50 lux[6] to 50 klux.

TFT-LCD is a variant of Liquid Crystal Display (LCD) which uses Thin Film Transistor (TFT) technology to improve image quality. TFT-LCD is one type of *active matrix* LCD, though it is usually synonymous with LCD. It is used in televisions, flat panel displays and projectors. **Active matrix addressing** is an addressing scheme used in video displays. Given a **m** × **n** matrix, the number of connectors needed to address the display is **m** x **n**. (Wikipedia)

Each pixel is attached to a switch-device, which actively maintains the pixel state while other pixels are being addressed, which also prevents crosstalk from inadvertently changing the state of an unaddressed pixel. The most common switching devices are TFT. This model is based on either the cheaper non-crystalline thin-film silicon **a-Si** semiconductor material. **[Stiftung-88]**

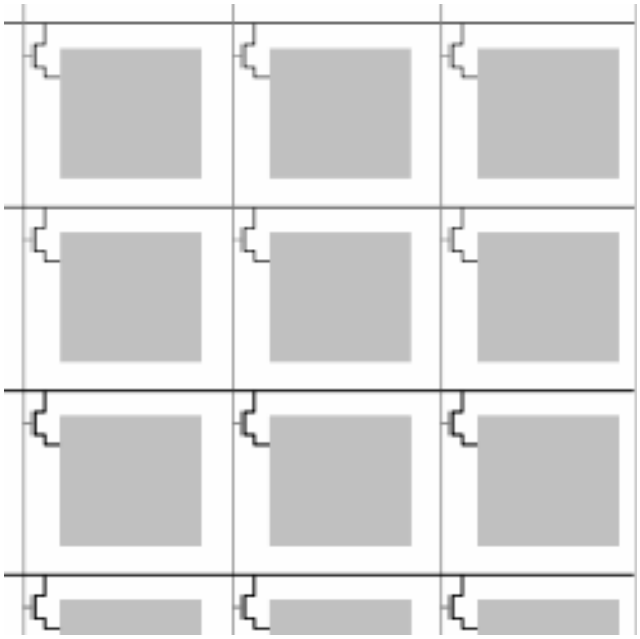


Figure 11 - Pixel layout in an active matrix

This model adds an integrated array of optical sensors, which, after image processing, provides the XY location of a shadow cast by a stylus or finger touch. **A-Si** TFTs exhibit a high photocurrent when exposed to visible light, therefore, shielded from the backlight by the gate metal and from ambient light by the black matrix on the color plate, to take advantage of the photosensitivity of a-Si TFTs and integrate an optical sensor array into the LCD to be used for touch detection. **[Boer-03], [Matuszczyk]**

[6] **Lux:** (symbolized lx) is the unit of illuminance in the International System of Units (SI). It is defined in terms of lumens per meter squared (lm/m^2).

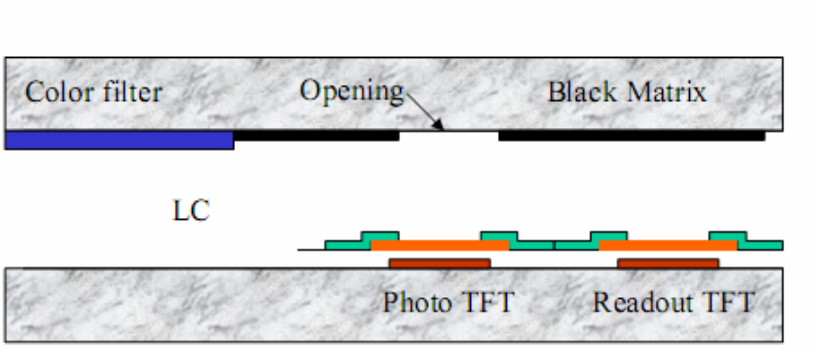


Figure 12 - Cross section of readout and photo a-Si TFT with an opening in the black matrix.

When the display is touched, the ambient light is locally blocked at one or a few of the photo sensors (figure 12). The surrounding photo sensors are exposed to the ambient light through the opening in the black matrix. Their storage capacitors will be partially or completely discharged. Each time a row of photo sensor circuits is read out, a current flows through the readout lines that is proportional to the integrated light exposure on the phototransistor during the preceding frame period.

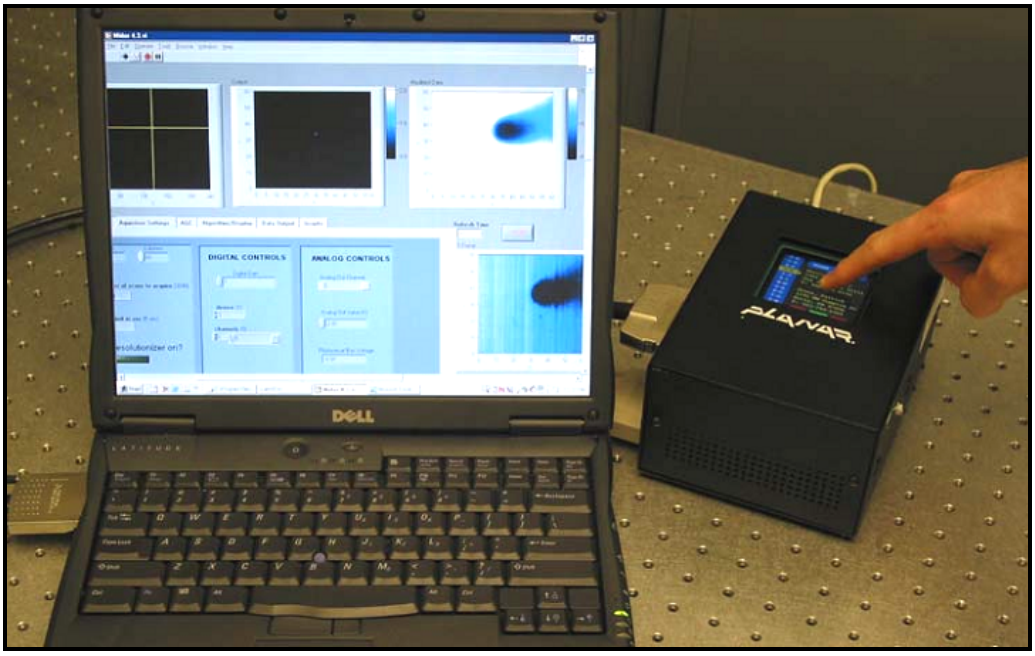


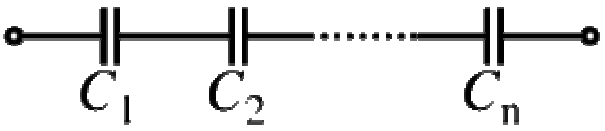
Figure 13 - Photograph of prototype TFT LCD with Integrated Optical Touch Screen (right).

The LabVIEW plots on the notebook the image of a finger touch and a cross-hair indicating location of touch. **[Boer-03]**

2.2.5. Capacitance Screens

A **capacitor** is an electrical/electronic device that can store energy in the electric field between a pair of conductors (called “plates"). The process of storing energy in the capacitor is known as **charging**, and involves electric charges of equal magnitude, but opposite polarity, building up on each plate. Most capacitors are designed to maintain a fixed physical structure. However, various factors can change the structure of the capacitor; the resulting change in capacitance can be used to sense those factors.

The current through capacitors in series stays the same, but the voltage across each capacitor can be different. The sum of the potential differences (voltage) is equal to the total voltage. Their total capacitance is given by:



$$\frac{1}{C_{eq}} = \frac{1}{C_1} + \frac{1}{C_2} + \cdots + \frac{1}{C_n}$$

Reference: [Huelsman]

So, if a finger changes the connectivity between capacitor plates, the capacitance changes, and using the total capacitance, we can detect the position of the finger by calculating the change in capacitance-connectivity.

In the ***matrix approach***, a series of conductors are arranged in an array of parallel lines in two layers, separated by an insulator and crossing each other at right angles to form a grid. A high frequency signal is applied sequentially between pairs in this two-dimensional grid array. The current that passes between the nodes is proportional to the capacitance. When a virtual ground, such as a finger, is placed over one of the intersections between the conductive layers, some of the electrical field is shunted to this ground point, resulting in a change in the apparent capacitance at that location. **[Gerpheide]**

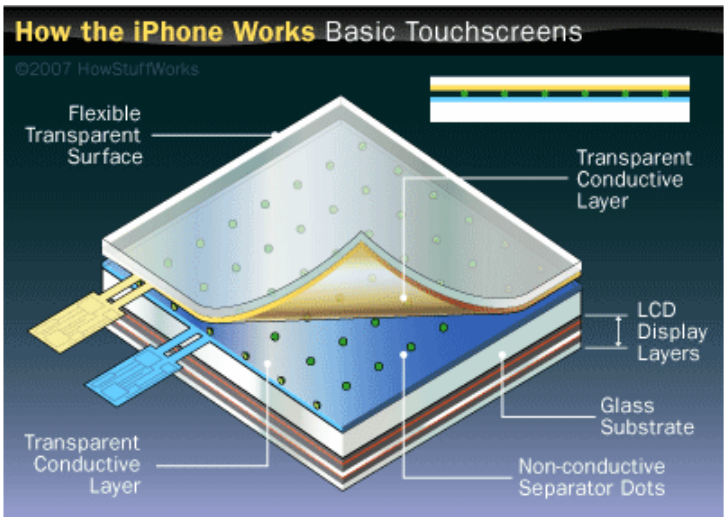


Figure 14 - Basic Capacitance Touch-screens

iPhone form Apple Inc. uses a similar approach. iPhone uses a new arrangement of existing technology. Its touch-sensitive screen includes a layer of capacitive material, just like many other touch-screens.

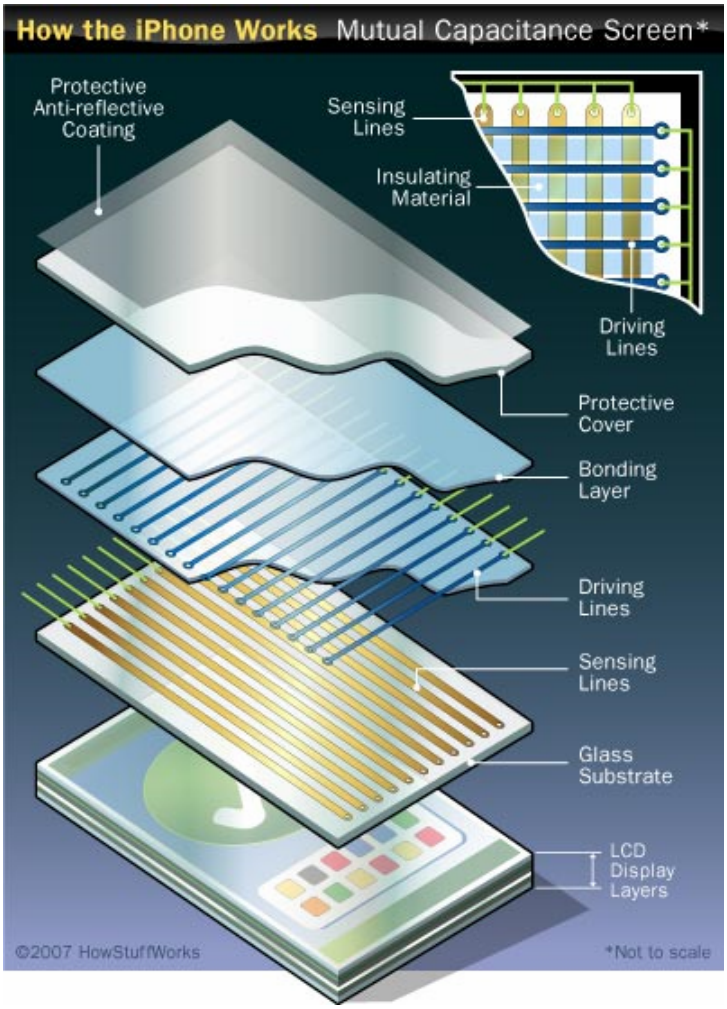


Figure 15 - Mutual Capacitance Touch-screens

Contains a grid of sensing lines and driving lines to determine where the user is touching

However, the iPhone's capacitors are arranged according to a coordinate system. Its circuitry can sense changes at each point along the grid. In other words, every point on the grid generates its own signal when touched and relays that signal to the iPhone's processor. This allows the phone to determine the location and movement of simultaneous touches in multiple locations. Because of its reliance on this capacitive material, the iPhone works only if you touch it with your fingertip -- it will not work if with a stylus or if the user wears non-conductive gloves.

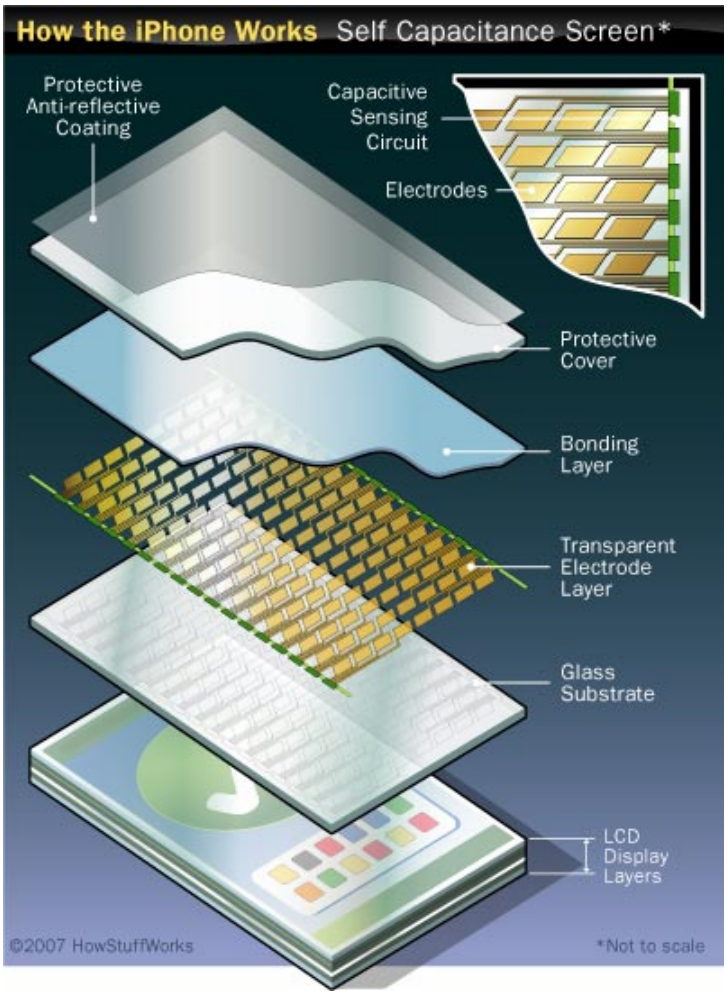


Figure 16 - Self-Capacitance Touch-Screens

Contains sensing circuits and electrodes to determine where a user is touching

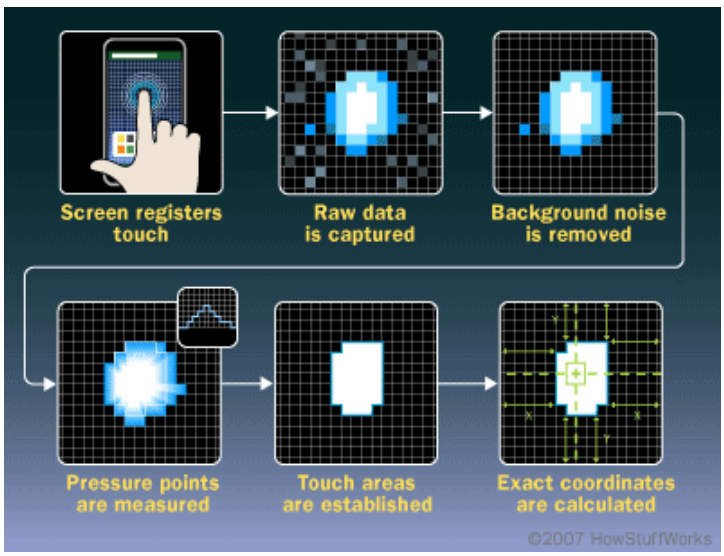


Figure 17 - iPhone Coordinates Analysis

2.3. Hardware Models Comparison

This section details the differences between the hardware models discussed in the previous section, along with advantages and disadvantages of each, the feasibility of building it in Egypt on large scale, and its ability to fit our requirements.

Touchwork Hardware Prototype Requirements:

1. Medium sized, between 20-100 cm in width and height.
2. Low electricity consumption.
3. Low cost.
4. Easy setup.
5. Portable, can work in different light environments without modification.
6. Easy to maintain, fix and modify.

2.3.1. Frustrated Total Internal Reflection (FTIR)

Advantages:

a. Provides a crisp clear image for processing.
b. Detects thin objects, allowing using a stylus.
c. Does not require changes in configuration when the light environment changes so it is calibrated only once.
d. No need to build a confining box during testing phases.
e. We can always add DI and have an FTIR/DI setup, with the addition of an infrared light source under the table.

Disadvantages:

a. Setup needs a led frame which has to be custom made for different acrylic dimensions (soldering needed, unless you use a LED holder such as PC-Boards or Breadboards).
b. Only allows touching with fingers, and support other objects to a very limited extent.
c. Requires a compliant surface, so dragging operations can be easily made. **[Section 2.2.1]**

Projects:

Audio Touch, Perceptive Pixel, TouchLib

Results:

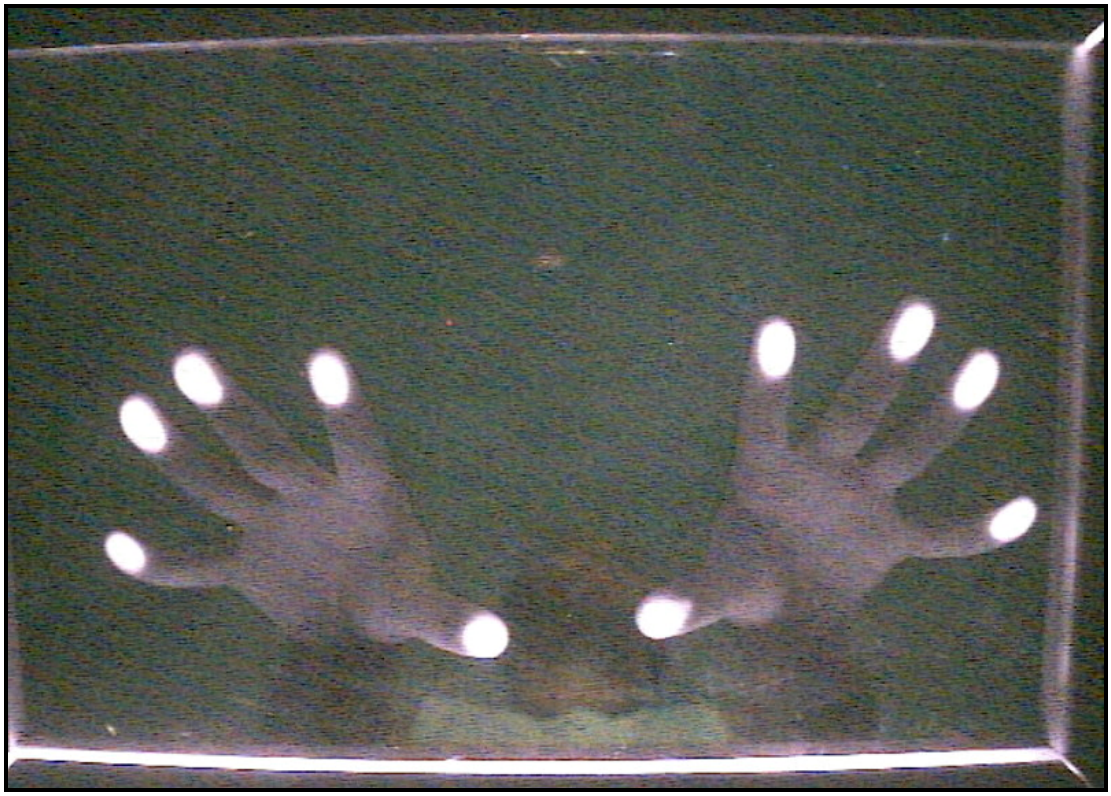


Figure 18 - Initial snapshot captured using a FTIR setup

Figure 19 - FTIR snapshot before and after image processing

2.3.2. Diffused Illumination (DI)

Advantages:

a. No need for a compliant surface, just a diffuser or a projection surface on top.
b. No need for an acrylic sheet, any other transparent materials like glass could be used instead.
c. No frame required since this model does not require LEDs placed at the sides, allowing different housing setups.
d. No soldering required, because you can buy ready-made infrared illuminators (very expensive though).
e. Simple setup.
f. Tracks different objects as fingers and fiduciary markers[7].

Disadvantages:

a. Blobs are not as clear and sharp as of the FTIR model.
b. Difficult to get it working right, lots of effort must be done to make sure that the diffused light is distributed evenly on the surface.
c. Infrared diffusers are relatively expensive.
d. Requires calibration on different places according to the surrounding lighting in the environment, also daylight and room light changes can affect the setup requiring recalibration.
e. Must be confined in a closed box, before you can test it.
f. Cannot detect thin objects or styluses.

Projects:

Microsoft Surface, ReacTIVision, CityWall

Figure 20 - CityWall at the center of Helsinki, Finland

[7] **Fiducials:** (or fiduciary markers) are objects used in the field of view of an imaging system which appears in the image produced. Markers can simplify computerized image processing applications such as motion capture, by providing an easy-to-track feature in images which follows the movements of the marked subject.

Results:

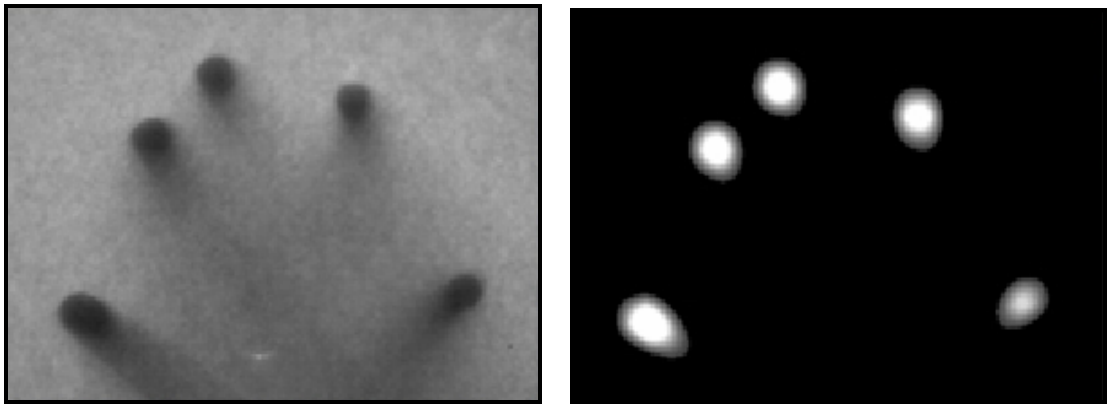


Figure 21 - DI snapshot before and after image processing

2.3.3. Diffused Surface Illumination (DSI)

Advantages:

a. No problems getting the light even, unlike pure DI setups.
b. Easy setup.
c. FTIR setup can be easily converted to DI.
d. No need to build a case for the setup, as in DI setups.
e. Possible Fiducial tracking.

Disadvantages:

a. Blobs are not as clear and sharp as of the FTIR model.
b. Less contrast compared to normal DI setups, because the Plexiglass used also redirects the infrared towards the camera.
c. More problems with ambient infrared because of the low contrast.
d. Possible size restrictions as Plexiglass if very soft, very thin, and bendable, so it can easily break.

Projects:

Only private research projects at the moment writing this documentation

Results:

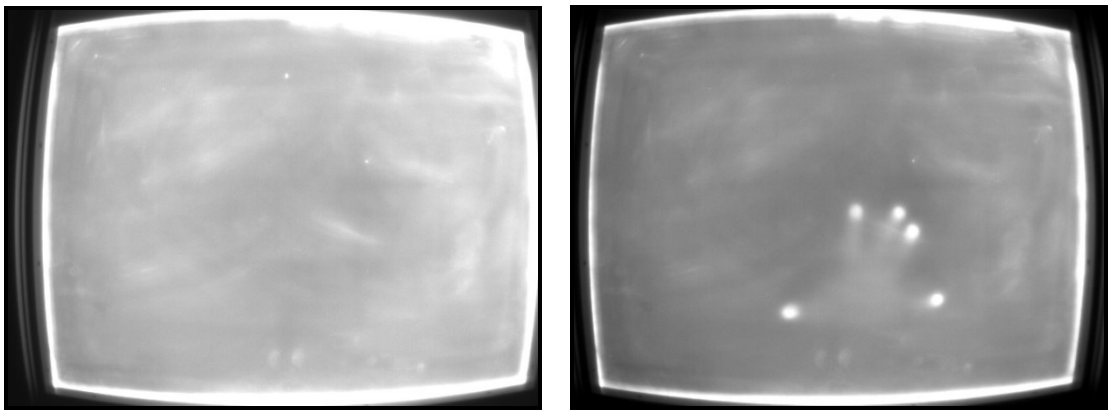


Figure 22 - DSI snapshots with and without touching the surface

2.3.4. Electronic Models
(TFT-LCD with Embedded Sensor and Capacitance Screens)

Advantages:

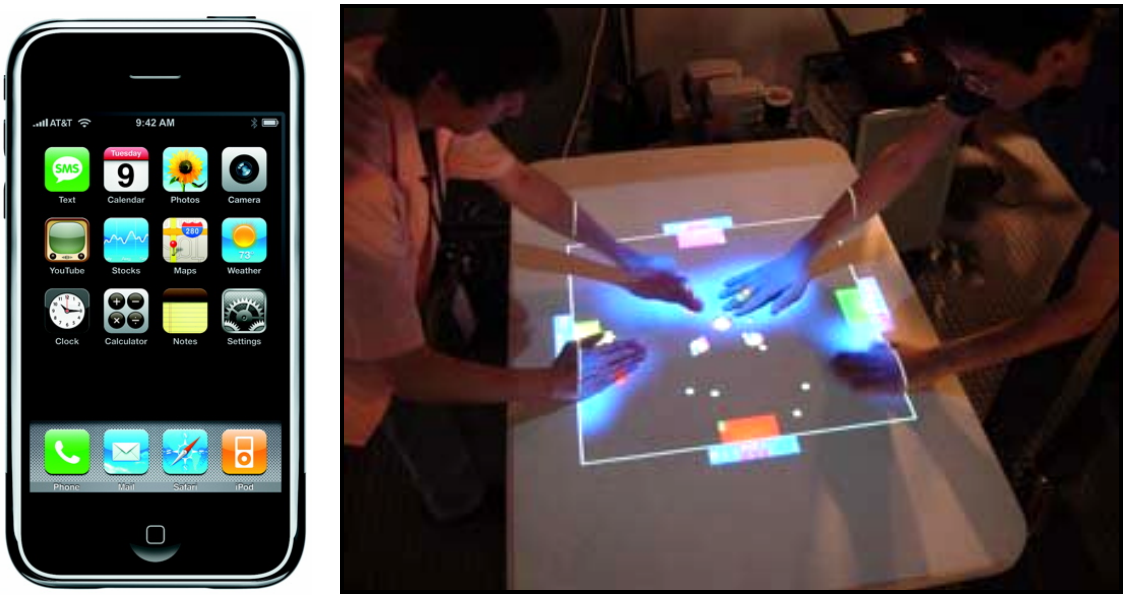
a. Results in a very thin, lightweight, portable screen.
b. Can operate with surrounding light.
c. No calibration required.

Disadvantages:

a. The electronic circuit layout for large screen will get very complicated and expensive.
b. There are no TFT displays manufactures in Egypt that can help us create one (also cannot be created using material from the Egyptian market).
c. Best fit for handheld devices and PDAs not for large prototypes.
d. A broad electronic background is required.

Projects:

iPhone (Apple Inc.), Synaptic Touchpad, SmartSkin **[Rekimoto]**



(a) **(b)**

Figure 23 - (a) iPhone (b) SmartSkin

2.3.5. Summary

In conclusion, both the FTIR and DI/DSI setups seemed to be the most appealing and the most compatible for our purpose. Table 2 compares the main properties and features in both models. The better properties are marked in red.

We decided to go for the **FTIR model** to build our hardware prototype. FTIR is considered the most appropriate for being cheap, requiring much less engineering effort, its robustness and its clear results. Also, it will be easy to add DI support to our prototype in the future.

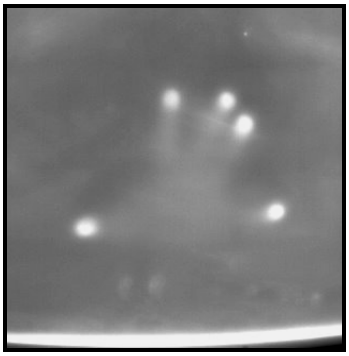
		FTIR Model	DI/DSI Model
Setup	**Ease of Setup**	Simple, except for the frame	Simple, except for the diffusers placement
	Calibration	Once	Whenever the environment changes
	Affected by Surrounding Light	No	Yes, requires recalibration and might even fail to work
	Sheet Material	Thick, clear acrylic	Any semi-transparent material
	Sheet Requires Polishing	Yes	No
	Compliant Surface	Must exist	Not required
	Light Source	Infrared LEDs Array – available	Infrared Diffuser or Ambient Light (e.g. sunlight, room-light) – not always available
	Setup Size	Depends on the quality of the infrared LEDs and the acrylic sheet	Unlimited, as long as light is distributed evenly on the surface
	Confining Table	Open or closed housing	Must be closed housing
Object Support	**Supports Fiducials**	No	Yes
	Supports Thin Stylus	Yes	No
	Supports Weaving	No	Yes
Results	**Resulting Image**	Crisp, sharp	Weak
	Image Processing Required	Simple, basic image processing	Lots of filters required
	Noise	None	Wrist and the palm shadows can appear in the input
	Samples		

Table 2 - Comparison between FTIR and DI/DSI Models

Better properties are marked in red

2.4. Building the Hardware

After deciding to work with the **FTIR solution**, the process of building the hardware prototype started. In this section, this process is described in details. Figure 24 below shows a basic FTIR setup:

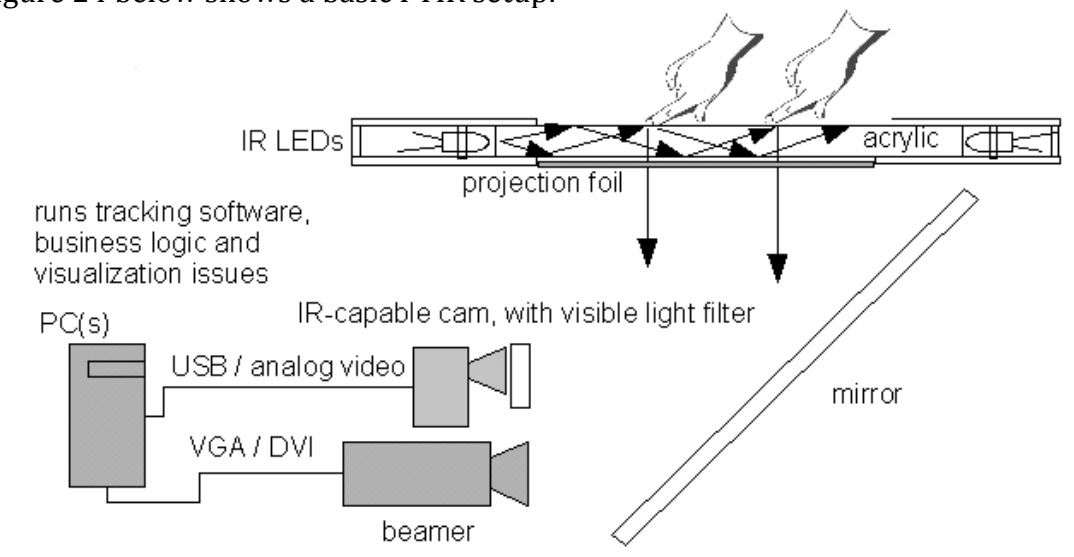
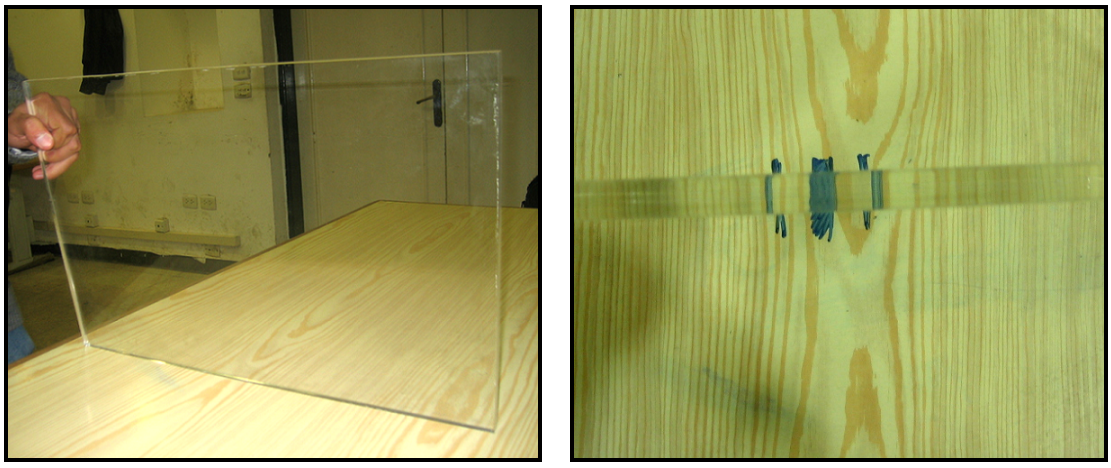


Figure 24 - Basic FTIR Setup

The setup is basically composed of:

1. **Acrylic Sheet:**
   The acrylic sheet[8] is a transparent sheet similar to glass or clear plastic. It plays the role of the touched screen. The thickness of the sheet ranges from 8-10mm and the sides have to be polished so it can be used (you can see in Figure 25(b) how the wood is completely visible through the sheet). If the sheet isn't properly polished, infrared light rays won't pass through it.



(a) **(b)**

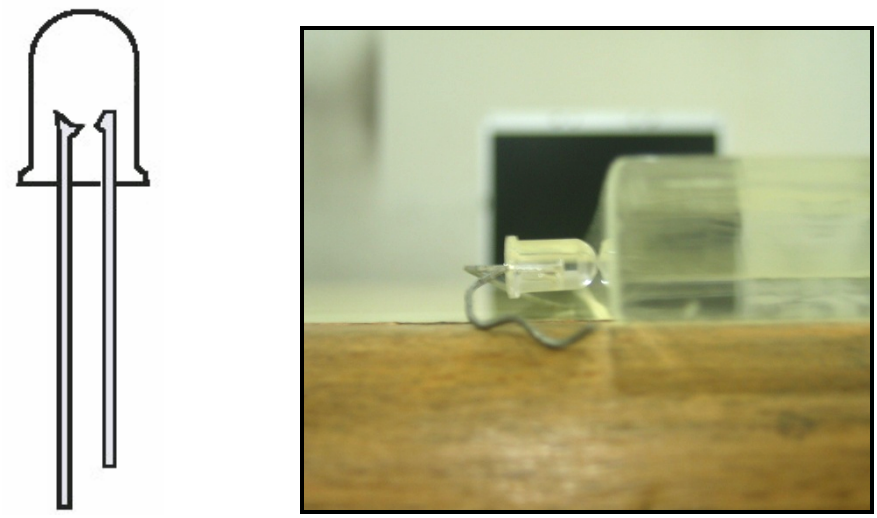
Figure 25 - Acrylic Sheet

(a) Acrylic Sheet of 10mm thickness **(b)** Acrylic sheet after getting the sides polished

[8] **Acrylic:** Chemically, it is the synthetic polymer of methyl methacrylate; poly(methyl methacrylate) (PMMA) or poly(methyl 2-methylpropenoate). It is commonly called acrylic glass or simply acrylic and sold by the trade names like Plexiglass.

Drilling holes for the infrared LEDs sounds like a good suggestion, as it will give a cleaner-looking and more compact surface except for the fact that it is very difficult to avoid frosting the holes as you drill. To solve that, an acrylic solvent could be poured into the holes to reform the acrylic around the LEDs.

2. **Infrared LEDs:**
   Light-emitting diodes (LEDs) are semiconductor diodes that emit incoherent narrow-spectrum light when electrically biased in the forward direction of the p-n junction. Simply, a LED is usually a small area light source, whose color depends on the composition and condition of the semiconducting material used, and can be infrared, visible, or ultraviolet. For our purpose and as previously explained in the FTIR section **[Section 2.2.1]**, we will by using around 150 infrared LEDs of 1.5 volts and 5 milliamps (5mm). Wiring the LEDs will be explained shortly.



(a) **(b)**

Figure 26 - Infrared LED

(a) A light-emitting diode (LED).
(b) An infrared LED setup similar to Figure 2.

3. **LED-Array Holder:**
   The infrared LEDs array has to be placed in some sort of a "holder". There are different solutions to that problem:
   a. **Metal (or plastic) frame:** One solution is building a metal or plastic frame. Aluminum L-profiles could be used for this purpose. The goal is building a metal frame with holes drilled on the sides to hold the infrared LEDs (as shown in Figure 27). The solution was fine in the time we were starting the project, but reaching the end of it we needed a much more professional solution.

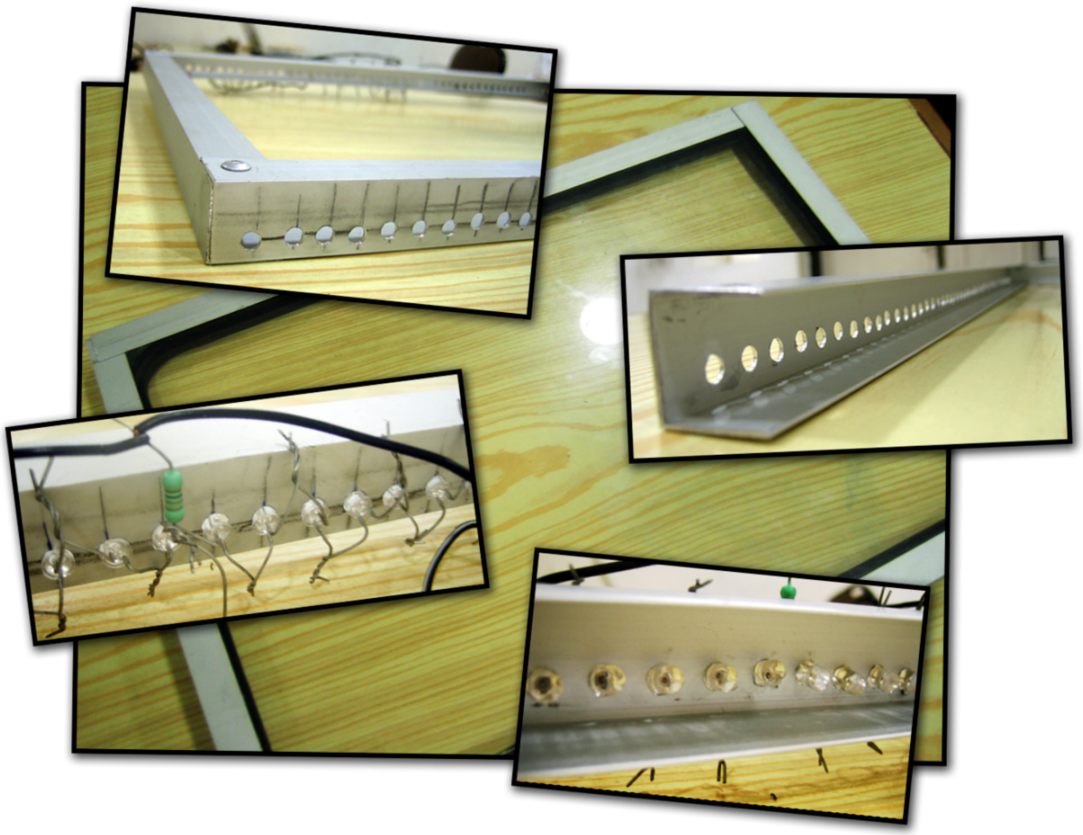


Figure 27 - LED-Array Holder: Metal Frame

b. **Circuit Boards:** Another solution was building circuit boards with LEDs fixed to them. This is a great solution until a LED goes bad, and you start facing huge difficulty replacing it.
c. **PC-Boards:** The best solution if you are connecting your LEDs in series – let's say connect every eight LEDs in series, then each bundle of eight in parallel. (In the series connection, LEDs could be far apart, which is something we couldn't afford due to the poor quality of the LEDs).

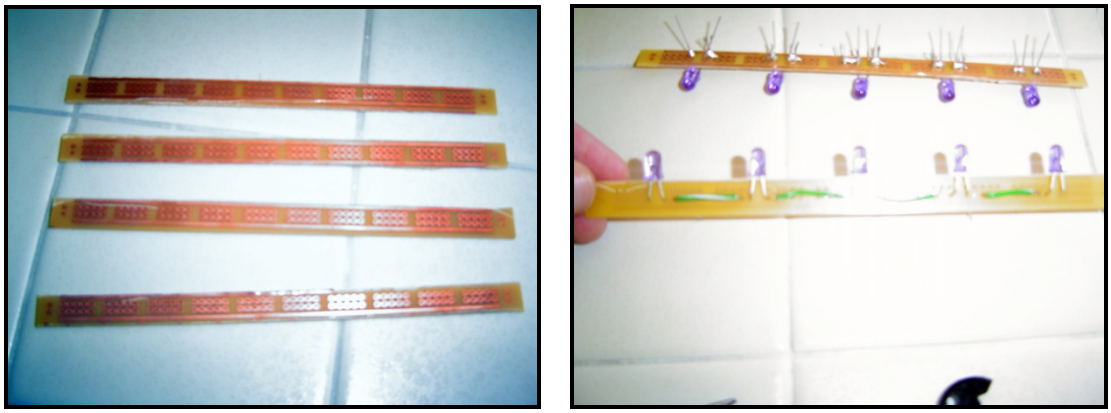


Figure 28 - Infrared LEDs Wiring on PC-Boards

d. **Breadboards:** The solution we settled for. Easy, just cut of the legs off each LED. It allows easy replacement of bad LEDs and circuit maintenance. Also, it made it easy to place the LEDs very close to each other, providing better lighting in our case of the poor quality of the LEDs. The LEDs are connected

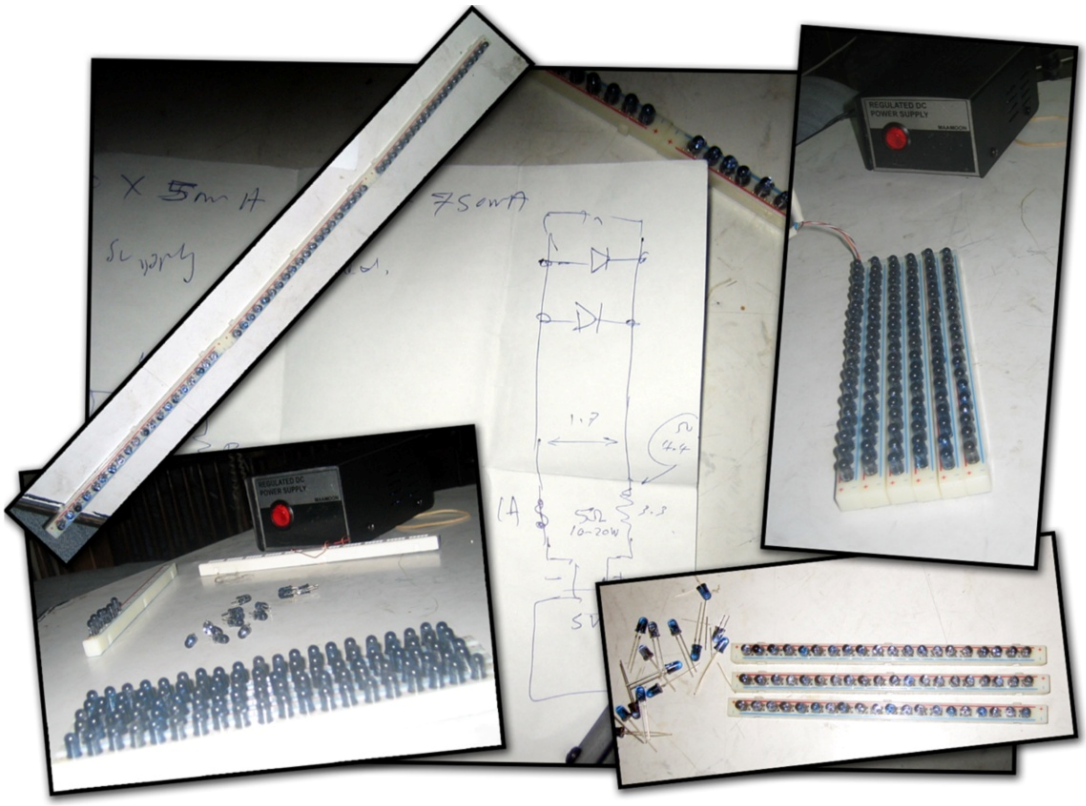


Figure 29 - Infrared LEDs Wiring on Breadboards

in parallel as shown in figure 29. Moreover, breadboards could be grouped together side-by-side creating an infrared light source for DI purposes.

4. **Resistances:**
   We used a resistance of 5 ohms and 7 watts in the circuit connection.

5. **Power Supply:**
   You need a power supply unit to provide a direct current to power up the LEDs.

   There are several options:
   a. **PC Power Supply:** The usual power supply used in most desktop computers. There are adapters on the power supply to connect it to drives; each with four colored wires (usually Red=5 volts, Black=Ground, Black=Ground, and Yellow=12 volts).
   b. **USB Power:** The USB ports caries 5 volts power. To access that, you'll have to cut the end off of the cable. You will also find four colored wires (Black=Ground, Red=5 volts, Random Color1=Data, and Random Color2=Data).
   c. **Wall Adapter/Transformer:** A device that converts wall AC power (Alternative Current) to DC power (Direct Current).

   We used the regulated, direct-current power supply (of 5 volts and 2 milliamps) shown in figure 30(c).

(a)

(b)

(c)

Figure 30 - Different alternatives for powering up the LEDs-array

(a) PC Power Supply **(b)** USB Power **(c)** Wall Adapter/Transformer.

6. Housing (Confining Table):

You will need a housing to contain the LED-Array and the acrylic sheet together along with the rest of the components.

Figure 31 - The Wooden Housing

We decided to build a wooden housing for that purpose. Besides, it will provide better lighting conditions for the camera (capturing) and the projector (visual aid). The housing is composed of a three main parts:

a. **Body:** Is the box-like shape that will contain the camera, mirror, and projector.
b. **Tray:** Is the upper face of the box. It is more of a tray rather than a face; shaped such that the acrylic sheet and the LED-Array holder fit in easily.
c. **Lid:** A part similar to the tray but inverted up-side down, such that it could close the whole set up from above.

7. **Webcam:**
It is recommended to get a camera with a high frame rate (30 fps or higher). Cameras have built-in, infrared filters that block infrared light. This filter is easily popped-up in cheap cameras and a very few high-end ones. In general, cameras with CCD[9] sensors should perform better than those with a CMOS [10]chip.

Figure 32 - Philips SPC900NC Webcam

One good high-end camera we used is Philips SPC900NC (figure 32), which has the following specifications:

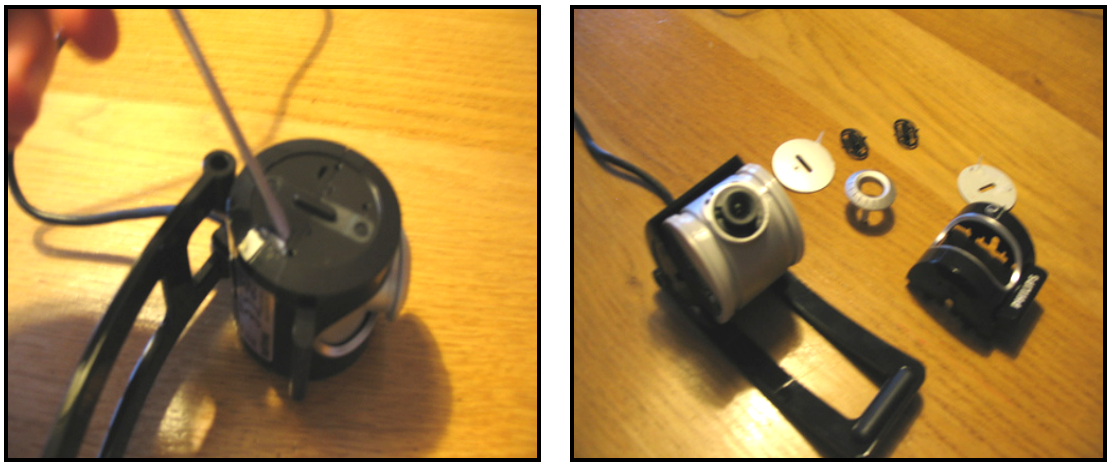
[9] **CCD:** Charged-coupled device, is an analog shift register, enabling analog signals (electric charges) to be transported through successive stages (capacitors) controlled by a clock signal. Here, it refers to the way that the image signal is read out from the chip inside the camera, from arrays of photoelectric light sensors.

[10] **CMOS:** Complementary metal–oxide–semiconductor is a major class of integrated circuits. CMOS technology is also used for a wide variety of analog circuits such as image sensors, data converters, and highly integrated transceivers for many types of communication.

Sensor	CCD
Sensor Resolution	VGA
Video Resolution	VGA
Snapshot Resolution	VGA
Interpolated Snapshot Resolution	1.3 Megapixels
Maximum Frame Rate	90 fps
Lens	F2.2, D55°
White Balance	2600 – 7600 k
Minimum Luminance	< 1 lux
Color Depth	24 bit

Table 3 - Philips SPC900NC Specifications

The next step is to hack the camera. Figures 33(a) and 8(b) show how the Philips SPC900NC could be hacked to remove the infrared blocking filter. The piece that holds the focus lens holds that filter on the other end (figure 33(c)). A simpler solution was to replace that piece with another one from an old camera that doesn't contain the infrared blocking filter (the piece will contain only the focus lens).



(a) **(b)**

(c)

Figure 33 - Hacking the Philips SPC900NC Camera

(a) & (b) Breaking the camera apart.
(c) Lens piece (LTR): Infrared-blocking filter, Lens piece, Focus Lens.

One last thing is that a special infrared pass filter is better added to the lens to prevent visible light from being detected. As you see in figure 24, the projected light from the projector will lighten up the sheet from the bottom, preventing

the camera from seeing any finger blobs. An infrared pass filter will make sure that *only* infrared light passes (reflected on your fingers) while blocking any other visible light (as the projected screen for instance). Infrared pass filters aren't that available and are expensive. A good, cheap replacement could be two layers of an exposed and processed film negative or a magnetic layer from floppy disk – we went for the first alternative.

8. **Projector:**
A projector is needed in order to provide the user with a visual feedback on the surface, some sort of visual aid to their interaction. There are two main display types: LCD and DLP. Liquid Cristal Display (LCD) has very sharp images with strong colors, yet the colors tend to fade after a few years. Digital Light Processing (DLP) have a very good contrast ration and are very small in physical size, but they normally have a slower response rate. A few more options should be considered as the brightness, the throw distance, and the aspect ratio.

Figure 34 - Epson EMP-X5 projector

We used an Epson EMP-X5 projector (figure 34) that has the following specifications:

Aspect Ratio (Native)	4:3
Brightness (ANSI Lumens)	2200 ANSI Lumens (Normal) 1750 ANSI Lumens (Eco Mode)
Contrast Ratio	400:1
Display Type	0.63" Polysilicon TFT LCD x 3
Resolution (Native / Max)	XGA (1024 x 768)
Projection Lens	F = 1.6, f = 18.4 ~ 22.12mm
Number of Colors	16.7 Million Colors
Lamp Type	170W UHE
Lamp Life	3000 hrs (High Brightness) 4000 hrs (Low Brightness)
Projection Distance	2.8ft. ~ 39.8ft. (0.85m ~ 12.13m)
Projection Mode	Front, Rear, Ceiling
Optical Zoom	1.6:1
Vertical Keystone Correction	+/- 30 Degrees

Table 4 - Epson EMP-X5 Projector Specifications

9. Compliant Surface:

A compliant surface is an overlay placed over the acrylic sheet, which couples with it under pressure to set the FTIR effect, and decouples once the pressure is released.

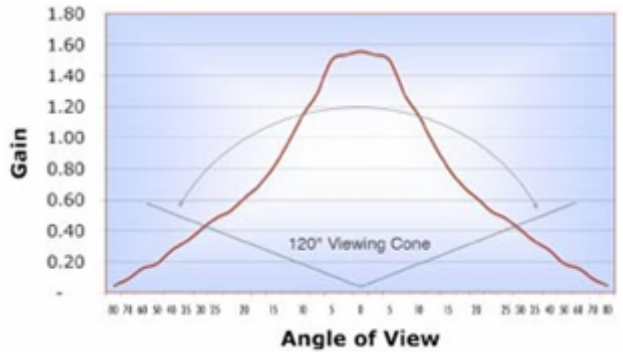
The compliant surface has several benefits:

1. Better for dragging movements, making the motion smoother when dragging you finger around in comparison to the acrylic.
2. Blocks light pollution (background light from the surrounding environment).
3. Provides consistent results, as the effectiveness of the bare acrylic touch depends on how sweaty or greasy yours hands are, and how much pressure you apply to the surface.
4. Protects the acrylic sheet against scratches and keeps it clean from stains and greasy marks.
5. Some compliant surfaces can act as projection screens.
6. Some compliant surfaces can act as diffusers, but the blobs that are detected by the camera get more blurry (less sharp).

Examples of compliant surfaces are silicon sheets, milk paper, caulk paper, silk, and Rosco sheets (will be described shortly). According to the most experimenters, silicon sheets give the best results, but it wasn't available for us. Instead, we decided to manufacture our own silicon sheet. Pouring silicon over the acrylic sheet is a difficult process as silicon is a thick material that isn't easily spread. A solvent, as toluene, could be used to reduce its thickness to the extent of turning it into a liquid that could be sprayed. Also, the thickness of silicon creates air bubbles during the spreading process, leading to noise particles in the detection process.

Figure 35 - Rosco Projection Sheet

The gray Rosco sheets seem to be the next best solution. They provide true-to-life image color and clarity with higher transmission and a brighter image. We ordered the sheet and cut a piece with a suitable size for our setup. Here are the Rosco gray sheets specifications:



Width	55", 86", 110"
Thickness	7 mil
Composition	100% PVC
Weight	1 Pound per sq. yd.
Cold Crack	-12° C
Viewing Cone	120°
Applications	Front Projection Rear Projection

Table 5 - Rosco Gray Sheets Specifications

The problem is that the blob intensity is less when using those sheets – you have to press even harder. A suggestion was to add a silicon layer between the Rosco and the acrylic surface.

10. **Mirror:**
    To bend the projector beam providing longer projection distance. The mirror also helps in projectors that do not have the rear projection function by inverting the image.
11. **Computer:**
    The webcam is connected to a computer in order to run the software and operate the framework or the application.

Important Note:

In the FTIR setup, the projection lens reflects a light spot on the lower side of the acrylic sheet; this light spot becomes in the field of view of the webcam. If the infrared pass filter doesn't block that light spot, it will be detected as a huge blob. In this case, both the webcam and the projector should be repositioned in a way such that the camera doesn't see that light spot, which is sometimes troublesome.

Part III

[Touchwork Internals]

The Software Modules: Input, Detection, Tracking, and Agent

Chapter 3: The Input Module

3.1. Overview

The Input Module, **twInput**, receives its input from a video camera of a prerecorded video file, decodes it, and transforms it to a sequence of images in a format understandable by the Detection Module **[Chapter 4].**

Module Information:

Input Video Stream
(from a camera device, still-image capturing device or a video file)

Output Sequence of images, each image must be represented by the format detailed in table 6.

Format	RGB
Compression	Uncompressed
Bit Count	24
Bits Per Color	8
Width	Undefined
Height	Undefined

Table 6 - Output of the Input Module

The whole sequence (stream) is presented to the framework with a frame rate of 30-60 frames per second.

Configurations of the capture device are saved by the ***Configuration Manager*** **[Section 1.5]** to be used whenever the framework is loaded. This configuration information are retrieved by the input module and reapplied on the capture device after it is activated.

Configurations retained by the Input Module:

1. **Frame Rate**
   Default: 30 fps (frame per second)
2. **Capture Resolution**
   Default: 320x240.
   This denotes the preferred size for the images received from the device.
3. **Color Space**
   Default: RGB-24.
   This value defines the preferred format of images received from the device.
4. **Compression**
   Default: None.
   Defines the preferred compression level of images received from the device.

3.2. Requirements

1. **Abstraction:**
   Provide an abstraction layer to the framework so that the video source used can be any capture device. Moreover, the module must hide all platform or library dependent specifications form the rest of framework in order to ease portability and enable the framework to be cross-platform with minor modifications.
2. **Emulation:**
   Provide an emulation layer in the case of absence of a capture device, so it should read a prerecorded video file or a network stream.
3. **Speed:**
   The abstraction and emulation layers must be fast enough in order to increase the time allowed to framework to process the user input.
4. **Responsiveness:**
   In case the framework takes more time than the allocated to process a certain input frame and other frames are present already from the capture device, the input module should skip the intermediate frames and send the most recent one to the framework, based on a timestamps associated with every frame.
5. **Extensibility:**
   The module must be extensible in order to fulfill any future requirements of the framework and also to work with different capture and video libraries.
6. **Configurable:**
   Due to the diversity of video capture devices and working environments (surrounding lights, camera quality) the module should expose configuration methods for these devices.

3.3. Possible Solutions

3.3.1. Level of Interaction with Windows APIs

1. **Using a video library that interacts with Windows APIs**

   Examples: Video Wrapper, DSVideoLib

   Advantages:
   a. Easy to use
   b. Fast to prototype
   c. Portable.

   Disadvantages:
   a. Not flexible enough
   b. Has no emulation layer, and also this layer cannot be created above them
   c. Will dictate its own design on the whole framework unless another layer is built above it (a wrapper).

2. **Using Windows APIs directly:**

   Examples: VFW (Video for Windows), Active Movie, Microsoft DirectShow

   Advantages:
   a. Flexible and Powerful
   b. Customization is possible

   Disadvantages:
   a. Complicated, has a steep learning curve
   b. Not portable

3.3.2. Available Windows APIs

1. **Video For Windows (VFW)**
   VFW is a multimedia framework developed by Microsoft that allowed Microsoft Windows to play digital video. It is also referred to as Video Compression Manager (VCM).

   Features:
   a. Easy to program.
   b. Uses the old standard VxD (Virtual X Driver) model that was replaced with WDM (Windows Driver Model) in 2000.
   c. Does not offer enough power and not flexible enough.

2. **Microsoft DirectShow (DS)**
   A multimedia framework and API produced by Microsoft performs various operations with media files or streams. It is the replacement for Microsoft's earlier Video for Windows technology. Based on the Microsoft Windows Component Object Model (COM) framework, DirectShow provides a filter-based framework that can render or record media files on demand. The DirectShow development tools and documentation were originally distributed

as part of the DirectX SDK, however, they are currently distributed as part of the Windows SDK (formerly known as the Platform SDK).

Features:

a. Provides a common interface for media across many of Microsoft's programming languages.
b. Supports the latest driver models (e.g. modern camera devices).
c. Powerful and flexible.
d. Extensible.

3.4. Proposed Solution

For the sake of flexibility, extensibility and power, we developed the input module depending on **DirectShow** APIs.

3.5. Architecture

The module is composed of 3 interacting components figure 36:

1. **Camera Driver:**
   Handles the interaction with the camera according to its internal interface, there are two types of drivers VxD (Virtual X Driver) which is old and supported by devices manufactured before the year 2000, and there are WDM (Windows Driver Model) drivers supported by modern devices manufactured lately.
2. **Microsoft DirectShow:**
   Captures the Video form the Camera Driver or video files, handles configuration.
3. **twInput:**
   a. Delivers a sequence of images to the framework.
   b. Abstracts the video source (which can be a camera or a video file).

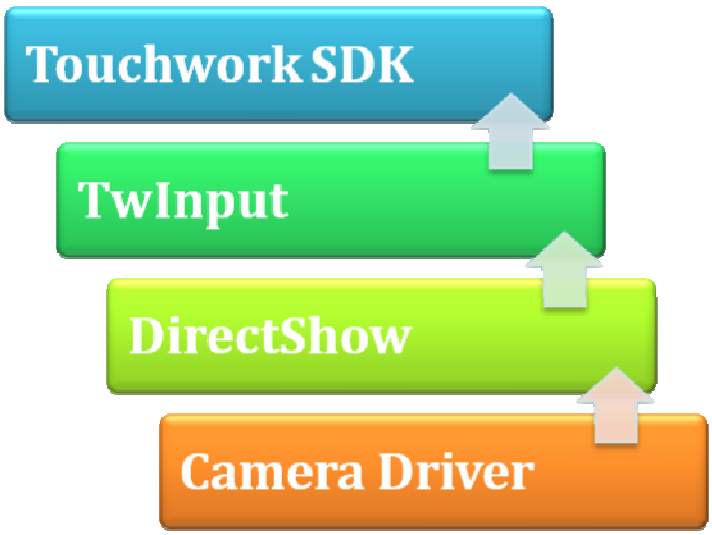


Figure 36 - Input Module Architecture

DirectShow divides a complex multimedia task (e.g. video playback) into a sequence of fundamental processing steps known as filters. Each filter – which represents a single stage in the processing of the data – has several input and/or output pins that may be used to connect the filter to other filters. The generic nature of this connection mechanism enables filters to be connected in various ways so as to implement different complex functions. To implement a specific complex task, a developer must first build a filter graph by creating instances of the required filters, and then connecting the filters together.

Application performs any task by connecting chains of filters together, so that the output from one filter becomes the input for another. A set of connected filters is called a *filter graph*. For example, the following diagram shows a filter graph for playing an AVI file.

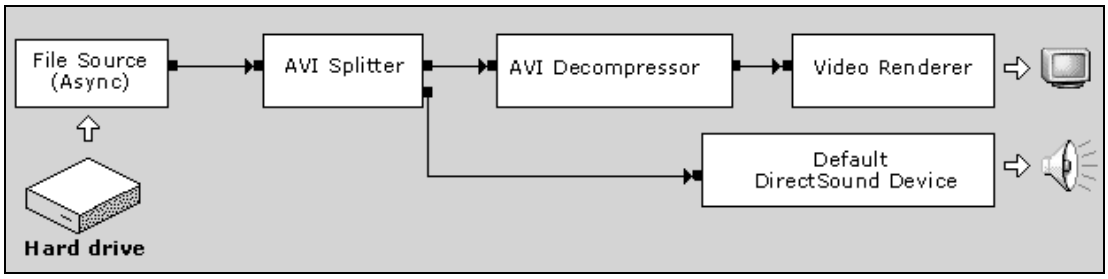


Figure 37 - Filter graph for playing an AVI file

The File Source filter reads the AVI file from the hard disk. The AVI Splitter filter parses the file into two streams, a compressed video stream and an audio stream. The AVI Decompressor filter decodes the video frames. The Video Renderer filter draws the frames to the display, using DirectDraw or GDI. The Default DirectSound Device filter plays the audio stream, using DirectSound.
A **Media Sample** is sent along the graph from the Source to the Sink. Every filter transforms the Media Sample to a format that is understood by the next filter in the graph. The filters are controlled by a high-level component called the Filter Graph Manager. The application makes high-level API calls such as "Run" (to move data through the graph) or "Stop" (to stop the flow of data). For more control over the stream operations, we can access the filters directly through COM interfaces. The Filter Graph Manager also passes event notifications to the application. **[PSDK]**

There are three types of filters in the graph: (Figure 38)

1. **Source Filter:**
   Read the input from a device or a file and send it downstream along the graph.
2. **Transitional Filter:**
   Convert the data from the Source Filters to a format that is understood by the Sink Filters.
3. **Sink Filter:**
   Receive the data from the graph, display it to the user and releases the resources allocated to the data, then waits for new data from the source graph.

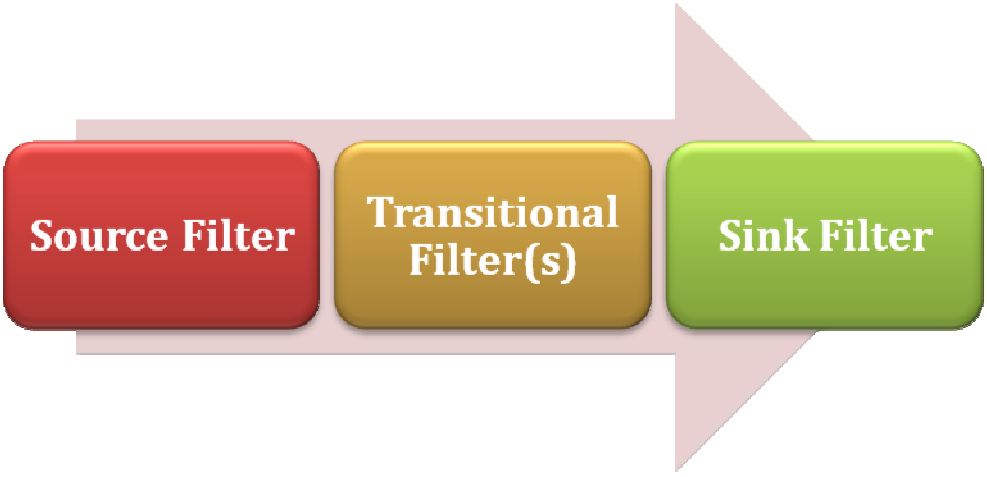


Figure 38 - A generic filter graph

Filters used to build the Source Filters:

1. **WDM Video Capture Filter:**
   The WDM Video Capture filter controls analog capture devices that use Windows Driver Model (WDM) drivers.(figure 39)

Figure 39 - Input and Output pins for the Capture Filter

Filters used to build the Transitional Filters:

2. **Color Space Converter (CSC):**
   Converts YUV images captured by the Source filter to RGB images. (figure 40)

Figure 40 - Input and Output pins for the CSC Filter

3. **Sample Grabber:**
   Grabs a copy of the image, sends it to the Detection Module (and then the framework), then passes the copy to the next filter in the graph. (figure 41)

Figure 41 - Input and Output pins for the Sample Grabber Filter

4. **AVI Compressor:**
   Encodes the RGB images received from the sample grabber into an AVI (Audio Video Interleave) stream and then presents it to the video Renderer filter. (figure 42)

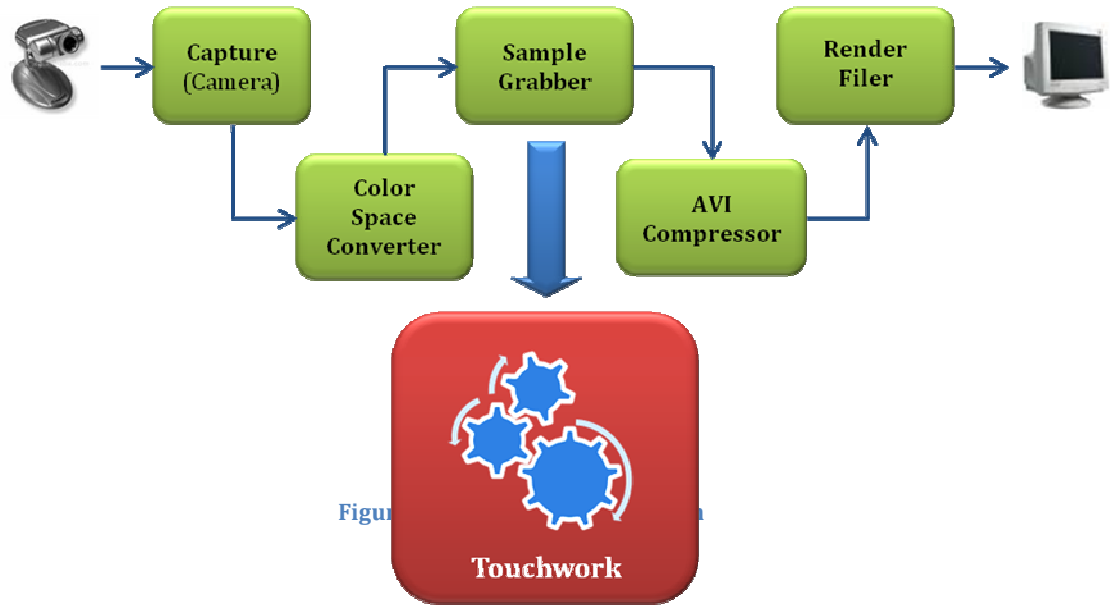
Figure 42 - Input and Output pins for the AVI Compressor Filter

Filters used to build the Sink Filters:

5. **Video Renderer:**
   For debug purposes, this filter renders the final images after being processed by the framework. (figure 43)

Figure 43 - Input and Output pins for the Renderer Filter

And the final filter sequence graph becomes:



Figu

Chapter 4: The Detector

4.1. Overview

As a consequence of the previous module, the input video captured by the camera is ready to be handled and processed later by the framework. Thus, there was a need for a detection algorithm to detect process and identify the positions of the blobs - the user's fingers on the multi-touch interface.

The Detection Module, **twDetector**, is responsible for processing each frame in the frame sequence received from the input module by applying image processing and blob detection algorithms to identify all available blobs and prepare them for further processing. An optimized technique must be used for blob detection where each frame will be subjected to those tracing and image processing algorithms (as background separation) to help label and identify each blob in that frame on a color-based criteria; white blobs on black background.

Module Information:

Input A captured image.
(the module is triggered each frame to process a single image)

Output List of blobs in frame.

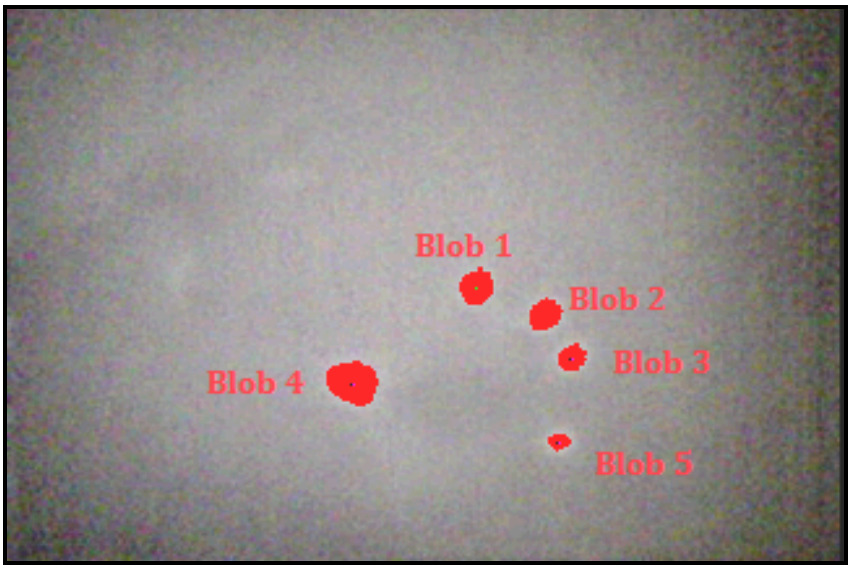


Figure 45 - Detection of five finger touches

4.2. Requirements

Any processing – either the preprocessing or the detection algorithm – implemented in that module should have some criteria:

1. **Optimized algorithms:**
   As previously explained, the detection algorithm will be applied per each frame in the input frame sequence, thus there is a need for an optimized algorithm that runs with low complexity and consumes low memory space in order to avoid any processing delay of the framework.
2. **Generic:**
   Independence of the illumination and blobs shapes should be considered in the algorithms selected and their implementation.
3. **Handle noisy images:**
   The algorithms used should be independent on the image quality processed and should handle any noise and can distinguish between blobs and noise.

4.3. Construction

We face the need to detect and classify objects (blobs) in each frame. Any image is formed out of objects that are formed out of components that in turn are made of connected pixels. By extracting the objects from their backgrounds, it becomes easier to identify them which accordingly will help in identifying the blobs in each frame in the input sequence. Thus, a detection algorithm is needed to extract and identify the blobs; also some preprocessing techniques are applied for noise reduction and optimization.

Accordingly, this module was divided basically in to two phases:

1. **Preprocessing Phase:**
   Filtering functions are implemented to be responsible for noise reduction and preparing the image for applying the detection algorithm, preprocessing filters will be discussed in details in the next section.
2. **Detection Phase:**
   Different methods are used for detection, yet contour tracing has proved to be the most effective method for detecting and identifying two-dimensional objects in any given image. Accordingly a labeling contour tracing algorithm will be used to detect the blobs in each frame, many contour tracing algorithms were introduced which will be discussed in further details in the section 5.3.

4.4. Preprocessing Phase

The captured image doesn't always make it easy for the detection algorithm to detect the blobs. One of the reasons could be the environment where the hardware unit is placed; the lighting in the room, any possible reflections caused by an object in the ceiling for instance, or any other light sources that could direct light into the scope of the camera. Another reason could be the camera itself; some cameras provide grainy images. The noise presented in the capturing phase could result in defining incorrect/unreal blobs.

To solve this problem, we had to apply some image processing operations to facilitate the easiness of the detection process. In this preprocessing phase, a sequence of "filters" was applied to the captured image received from the Input Module, before applying the detection algorithm. The phases are described as follows:

1. **Monochrome:** The Input Module outputs an RGB-image. The image is converted into grayscale; reducing the memory allocated for each frame and the number of instructions to be executed later on (to one-third).
2. **Noise Removal:** After that, the image is subjected to an erosion process followed by a dilation process with the same structuring element, in order to remove any possible noise or grains. A Gaussian filter is also applied in between for better noise removal.
3. **Thresholding:** The image is finally cut-off at some threshold value, leaving a solid black background (background separation) and obvious, clear blobs, ready for detection.

At first, implementing the filters (at least the second and the third ones) ended up with pulling down the performance dramatically - each frame could use around two seconds at least to be processed. Instead, we decided to use the OpenCV[11] library for that purpose. OpenCV is optimized for Intel processors, so it applies no processing overhead to our mechanisms. Hence, the performance is back to normal and you won't feel any slowing down due to the preprocessing phase. Now, the image is ready for the detection algorithm.

[11] **OpenCV:** (Open Computer Vision Library) is an open source computer vision library developed by Intel. It provides basic optimized (and hardware accelerated) C-style routines for many basic image processing algorithms and operations as capturing video from a camera, loading image files, playing AVI movies, and more . It is cross platform and free for commercial and research use under a BSD license.

4.5. Detection Phase

4.5.1. Different Algorithms

In this section, five important methods for component labeling are reviewed. One of them is the first proposed method, and the other four use varied strategies in attempt to improve the first. They all attempt to re-label component pixels according to the eight neighboring pixels around a given pixel and consequently detect and identify the components.

All of the five methods perform two passes over a binary image. The first method proposed by Rosenfeld and Pfaltz **[Rosenfeld]** where at each black pixel P, its four neighboring points (left, upper left, top, and upper right) are examined. If none of these neighbors is unlabeled, P is assigned a new label. Otherwise, those labels carried by neighbors of P are said to be equivalent. In this case, the label of P is replaced by the minimal equivalent label.

Haralick **[Haralick]** designed another method to remove the extra storage required for the pair of arrays proposed in the first method. Before starting to scan the image, each black pixel is given a unique label. In the first pass, each labeled pixel is assigned the minimum label among its four neighboring pixels. The second pass is like the first pass; the only difference is that the scanning process is done in a bottom up fashion. The process goes on iteratively until no more labels change. Unlike the first method, the memory used in this technique is small, yet the processing time may increase much with complex images.

The Lumia, Shapiro, and Zuniga **[Lumia]** method compromises between the two previous methods; in the first top-down pass, similar to the first methods, labels are assigned to black pixels. At the end of each scan line, however, the labels on this line are changed to their minimal equivalent labels. The second pass begins from the bottom and works similarly as the top-down pass. It can be proved that all components obtain a unique label after these two passes.

Fiorio and Gustedt **[Fiorio]** method used a different algorithm that runs in linear time, their method consists of two passes where in the first pass, equivalent labels are represented as a tree; while in the second pass, relabeling is done. The operation used in the union-find technique merges two trees into a single tree when a node in one of the two trees had an 8-connectivity relationship to a node in the other tree.

In Shima, Murakami, Koga, Yashiro, and Fujisawa **[Shima]** technique, a searching step and a propagation step are executed in an iterative way. In the searching step, the image is encountered until an unlabeled pixel is found and is assigned a new label. In the propagation step, the label of each pixel is propagated to other pixels bellow or above the scan line.

In the Component-Labeling Algorithm using Contour Tracing Technique by Fu Chang, Chun-Jen Chen, and Chi-Jen Lu **[Fu-Chang]**, an image is scanned from up to bottom and from left to right to detect the pixels of the components and follow that

by a contour tracing technique to label the pixels of the contour of that component. This algorithm runs in linear time and proved to be the most optimized algorithm of the previous mentioned algorithms. Therefore we chose that algorithm to implement it for the detection phase in the Detection Module (details of implementation will be discussed in the next section).

4.5.2. Component-Labeling Algorithm using Contour Tracing Technique

As a polygon can be determined by set of vertices, a component can be determined with the set of points lying on its contour. Basically, the algorithm scans a binary image from top to bottom and from left to right searching for the blobs in the received frames.

The searching process is divided mainly into four steps (as illustrated in figure 46):

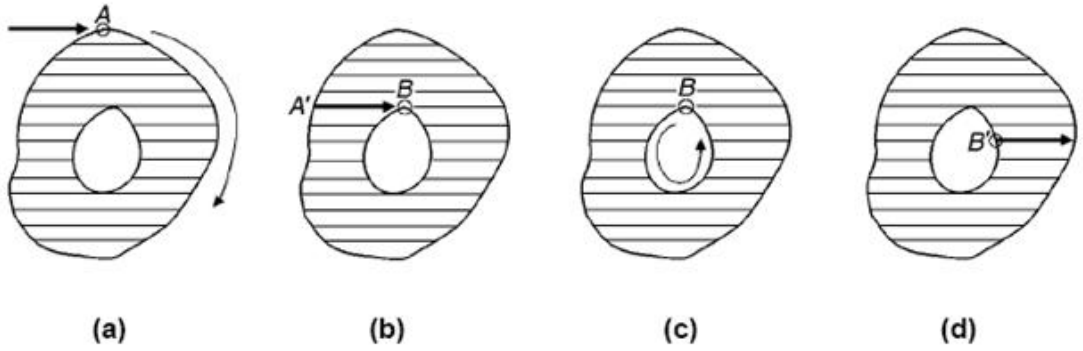


Figure 46 - The basic four steps of the searching process

- When an external contour point, say **A**, is encountered the first time, we make a complete trace of the contour until we return to **A**. We assign a label to **A** and to all points of that contour.
- When a labeled external contour point **A'** is encountered, we follow the scan line to find all subsequent black pixels and assign them the same label as **A'**.
- When an internal counter point, let's say **B**, is encountered the first time, we assign **B** the same label as the external contour of the same component. We then trace the internal contour containing **B** and also assign to all contour points the same label as **B**.
- When a labeled internal contour point, say **B'**, is encountered, we follow the scan line to find all subsequent black pixels and assign them the same label as **B'**.

When implementing the above procedures first we add a dummy row of white pixels at the top (in our case, the background of the received image from the hardware is black, accordingly a dummy row of black pixels is added instead) and associate the image with an array of the same size carrying the labels per each pixel (initially all pixels are unlabeled). Later on, a scanning procedure is followed searching for white pixels

We start initially with label **C** (where **C** = 1). Following the four steps described above, we will be able to assign a label for each pixel; the value of **C** is incremented when a new unlabeled external contour is encountered. Each time the value of **C** is incremented, this indicates that a new blob is found, a new blob is then created, given a new ID and added to the list of blobs (which is later on handled by the Tracking). **[Chapter 5]**

Let **P** be the current point that is being scanned and processed by our algorithm:

Step 1:

If **P** is unlabeled white pixel and the pixel above it is a white pixel, **P** must be on an external contour of a newly encountered component. So, we assign label **C** to **P** and follow that by a contour tracing mechanism (discussed later in details) to find and label the contour pixels with the same label **C**, then increase the value of **C** by 1.

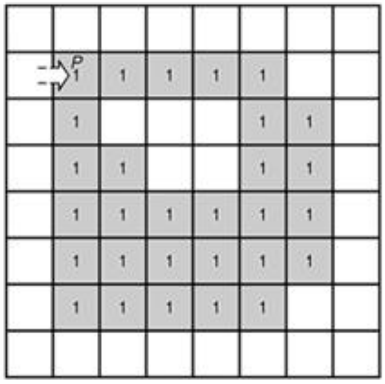


Figure 47 - Detection - Step 1
P is a point on the external contour

Step 2:

If the pixel below **P** is an unmarked white pixel ('unmarked' means that this pixel wasn't encountered before during any previous scanning process, more explanation of 'unmarked' will be provided later), **P** must be on a newly encountered internal contour. There are two possibilities. First, **P** is already labeled. In this case, **P** is also an external contour pixel. Second, **P** is unlabeled. In this case, the preceding point **N** on the scan line (the left neighbor of **P**) must be labeled. We then assign **P** the same label as **N**. In either case, we proceed to execute contour tracing to find the internal contour containing **P**, and assign the same label to all the contour pixels.

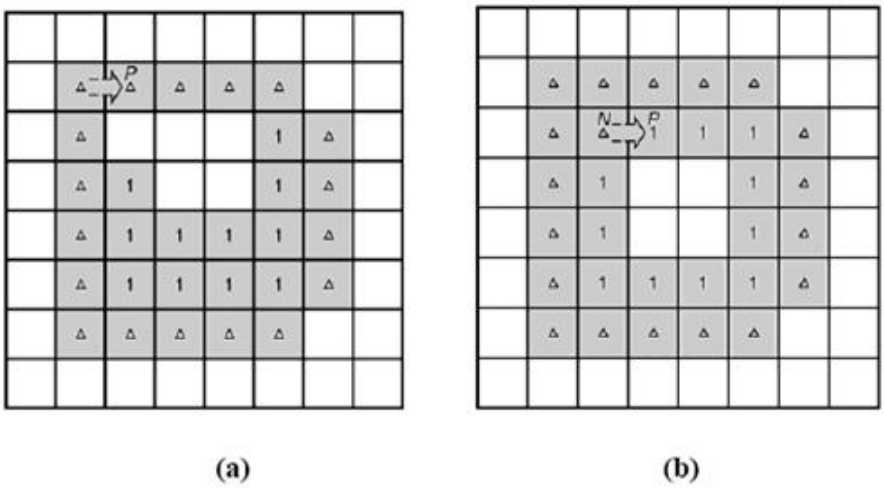


Figure 48 - Detection - Step 2

(a) **P** is also on the external contour (b) **P** is on the internal contour

Step 3:

If **P** is a point that isn't dealt with in Step 1 or Step 2 (i.e. **P** is not a contour point, **P** is a component pixel), then the left neighbor **N** of **P** must be a labeled pixel. We assign **P** the same label as **N**.

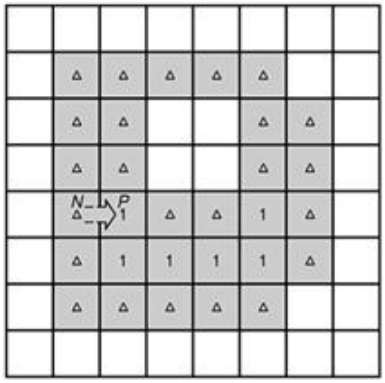


Figure 49 - Detection - Step 3

P is not a contour point

Notes:

In order to avoid executing counter tracing at the point **Q** (a point that is already labeled), we mark surrounding white pixels of a component with a negative integer. Therefore, when **Q** is encountered during scanning process, the pixels bellow **Q** are no longer unmarked white pixels and therefore no contour tracing will be processed. On the other hand, the pixels bellow internal contour pixel **P** are still unmarked white pixels as the internal contour has now been traced yet, so when pixel **P** is encountered, a contour tracing technique proceeds.

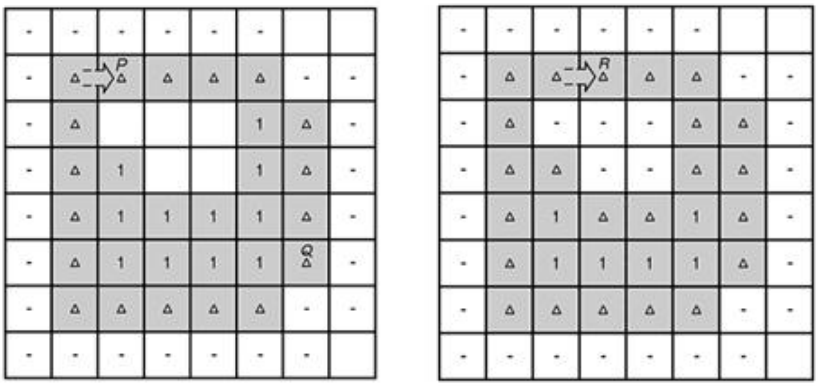


Figure 50 - Marking white pixels

(a) Unmarked white pixels **(b)** White pixels surrounding the contour

1 : Unlabeled black pixels
Δ : Labeled black pixels
– : Marked white pixels

By marking surrounding white pixels, we also ensure that each internal contour is traced only once. As illustrated in figure 50, when an internal contour is traced, white pixels are also marked so as to prevent further tracing, so neighbor pixels bellow pixel **R** are marked with negative values avoiding re-tracing (we need to trace the internal contour at a point only when the white pixel below that point is unmarked).

The operation of marking surrounding white pixels with a negative integer is included in the procedure Tracer, which is called during scanning by the contour tracing procedure explained in details bellow.

Contour Tracing

Contour tracing procedure is essential in finding either the internal or external contour of the pixel we are currently investigating.

Assume we have a given point **S** (as shown in figure 51). First, we execute the tracing procedure to check whether **S** is an isolated point of not. If **S** has been found as an isolated point, contour tracing is stopped, otherwise, the Tracer will output the contour point following **S** (let this point be **T**). We then continue to execute the Tracer until both the following conditions are satisfied:

1. Tracer outputs **S.**
2. The next contour point outputted by the Tracer is **T**.

As shown in figure 51, when **S** is the starting point and **T** is the next contour point, the path traced by Tracer is STUTSVWVS.

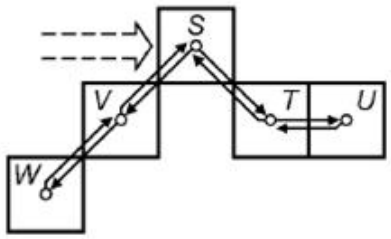


Figure 51 - Tracing a contour of a stripe-shaped component

The Tracer

At any given contour point **P**, the tracer should investigate the eight neighboring pixels searching for the contour point following **P**. Assuming each neighbor pixel has an index, the search process proceeds in a clockwise direction from the initial position that is determined in a special way.

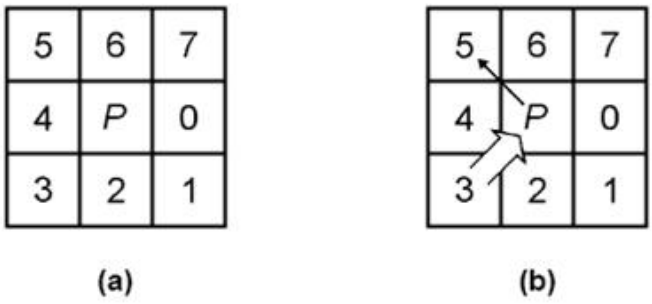


Figure 52 - Searching process in the Tracer

(a) The neighboring points of **P** are indexed from 0 to 7**(b)** If the previous contour point lies at 3, the next search direction is set to be 5

The initial search position varies according to the following cases:

1. **P is a starting pixel of an external contour:**
   In this case, the Initial position in this case is set to 7 (upper right) as the pixel above **P** is known to be a white pixel and the next pixel to it in the clockwise direction that should be considered is that pixel at position 7.
2. **P is a starting pixel of an internal contour:**
   The initially position is set to 3 (lower left) knowing that the pixel bellow **P** is

known to be a while pixel and the next pixel to **P** in the clockwise direction that should be investigated is that found at position 3.

Generally, when **P** is not the starting pixel of a contour, irrespective of whether the contour is external or internal, its initial search position is set to **d + 2 (mod 8)**, where **d** is the position of the previous contour point.

After determining the initial position, searching procedure is followed to determine the following neighboring black pixel. If no black pixel is found in the whole circle, **P** is identified to be an isolated point.

Here is the pseudo code for the function responsible for the contour tracing and detection:

```
Blob_List DETECTBLOBS(Video_Frame_Image)
      //Initialize labels to 0
      //Add dummy row of Black pixels
      //Start with C = 0

      //Scan image
      for(y : 0 → Height)
        for(x : 0 → Width)
           if (current pixel is a blob)
              if(The current pixel is unlabeled & lies on external
                  //Follow Contour Tracing Algo
                  //Initial start position = 7 (start = 7)
                  //Add currrent point to the Contour list of points

                  while(there are still more blobs in Contour list)
                      //Get the first pixel point to invisigate
                      //whether it belongs to component or not
                      Point p = Contour[count];

                      if(If p is a blob & unlabeled)
                          //Label it
                          //Follow Tracer method to determine the
                          //intial search position & then start
                          //searching

                          for(j = 0; j < 8; j++)
                              int k = (start + j)%8;
                              if(unlabelled pixel)
                                   //Add it to the Contour points
                              else if(unmarked white pixel)
                                   //mark it as white with -1}
                      else if(The current pixle is unlabeled &
                               not on external contour)
                           //Mark it as the pixel preceding it

                   C++; //Increment C
                   if(C > 1)
                       //A new blob is found
                       //Add it to the list
                       blobList.insert(current);
              else if(pixel is an unmarked white pixel)
                       //Mark it as the pixel preceding it
      RETURN blobList;
```

4.5.3. Modifications and Additions

Some modifications were made during the implementation of the algorithm in order to twist it to fit our problem. As the image captured by the input camera for the infrared light (as explained in the previous chapters) has a black background and white components such that the positions of the fingers (blobs) appear as white, we had to reverse the implementation of the algorithm to suite our problem and can detect the blobs successfully.

On the other hand, some additional information about the blobs were needed for further processing in the next modules, thus some additions were made to the algorithm to calculate some values like:

- **Blob Center:** The geometric center of each blob is calculated during the algorithm run time by summing the total pixels positions (x and y) forming the component (blob) and then later dividing them by the total number of pixels forming that blob.
- **Distance from Origin:** The distance of each blob from the origin is calculated and accordingly blobs are inserted in order in the list of blobs according to their distance, which will be used later on in the Tracker Module for tracking the blobs.

4.5.4. Algorithm Analysis

By analyzing the performance of the component labeling contour tracing algorithm, it is deduced that it is the most efficient algorithm to use.

Advantages of the Component Labeling Contour Tracing Algorithm:

1. **Linearity:**
   It requires only one pass over the image. Contour points are visited more than once due to the contour tracing procedure, but no more than a constant number of times, therefore the algorithm runs in linear time.
2. **No re-labeling mechanism:**
   Once a labeling index is assigned to a pixel, its value is unchanged.
3. **All contours:**
   As a result of the algorithm, we obtain all contours and sequential orders of contour pixels.
4. **Fastest:**
   Experimental results show that this algorithm is faster than traditional component-labeling algorithms (as illustrated bellow).

The founders of that algorithm have carried out different analysis experiments on the algorithm in comparison with other algorithms and techniques. The results of those experiments were published in paper titled "*A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique*" written by Fu Chang, Chun-Jen Chen, and Chi-Jen: **[Fu-Chang]**

Contour Tracing Algorithm is compared with the five other component-labeling methods; six types of test images: legacy documents, headlines, textual contents, half-tone pictures, newspaper and photographs. The test environment is an Intel Pentium III 1GHz personal computer with 384MB SDRAM. Each document type has four sets of images. Each set, in turn, consists of four images whose sizes correspond to four paper sizes: A3, A4, A5, and A6.

We have to make certain test images by cutting relevant objects from various sources and `pasting them onto a blank canvas of a specified size. The reason we need to make a collage out of small images is because we are not able to obtain a document that consists of only a single type of specified objects (e.g., headlines). Some test images, however, can be directly segmented from their sources (e.g., photographs, newspaper, and legacy documents) without being made into collages. All six algorithms being compared are listed in the bottom of table 7. The comparison results are listed in table below. The performances of all the algorithms are shown in table 7. In this table, the size of the test image is plotted along the horizontal axis, and the average processing time of each method is plotted along the vertical axis.

Document Type	Image Size (M pixels)	#CC	A	B	C	D	E	F
			Average Processing Time (sec)					
Legacy Documents	2.16	741	0.50	0.41	0.70	0.10	0.07	0.06
	4.33	1668	1.95	1.02	1.21	0.19	0.15	0.13
	8.69	3708	7.03	3.27	4.08	0.38	0.30	0.27
	17.39	6570	29.95	6.45	8.08	0.74	0.55	0.49
Headlines	2.16	439	0.70	0.79	0.76	0.12	0.10	0.07
	4.33	916	2.35	1.79	1.56	0.24	0.19	0.14
	8.69	1577	7.49	3.91	3.31	0.47	0.37	0.30
	17.39	3145	39.20	9.20	6.11	0.89	0.71	0.58
Textual Content	2.16	1808	1.78	1.02	0.76	0.12	0.12	0.08
	4.33	3509	6.45	2.50	1.53	0.23	0.24	0.15
	8.69	6825	25.29	6.30	3.10	0.47	0.45	0.31
	17.39	13157	370.71	15.31	7.37	0.92	0.92	0.66
Halftone Pictures	2.16	14823	3.18	3.86	2.77	0.18	0.19	0.09
	4.33	28793	14.51	10.09	5.08	0.37	0.42	0.19
	8.69	52087	59.57	25.76	13.05	0.71	0.75	0.36
	17.39	131628	773.93	100.83	28.13	1.41	1.68	0.76
Newspaper	2.16	1408	1.65	1.08	0.83	0.12	0.11	0.07
	4.33	4828	6.61	3.51	4.20	0.24	0.24	0.15
	8.69	12024	25.99	7.94	5.05	0.51	0.52	0.32
	17.39	16680	287.31	17.24	21.10	0.98	0.93	0.67
Photographs	2.16	4196	2.18	4.49	2.02	0.16	0.15	0.09
	4.33	2018	1.72	2.88	2.54	0.23	0.18	0.11
	8.69	3206	3.22	3.13	5.78	0.26	0.20	0.13
	17.39	1416	2.41	2.75	2.96	0.35	0.25	0.15

#CC: Number of Connected Components

A: Rosenfeld et al. **B:** Haralick **C:** Lumia et al.
D: Fiorio et al. **E:** Shima et al. **F:** Component-Labeling Algorithm

Table 7 - Performance of the six methods compared

4.6. Conclusion

By applying the previous preprocessing and detection algorithms, blobs are detected and identified. Where each blob is given a unique ID, in addition to computing other values such blob center, position of blob relative to the origin and some other values which will be used and discussed more in the next module, the Tracking Module.

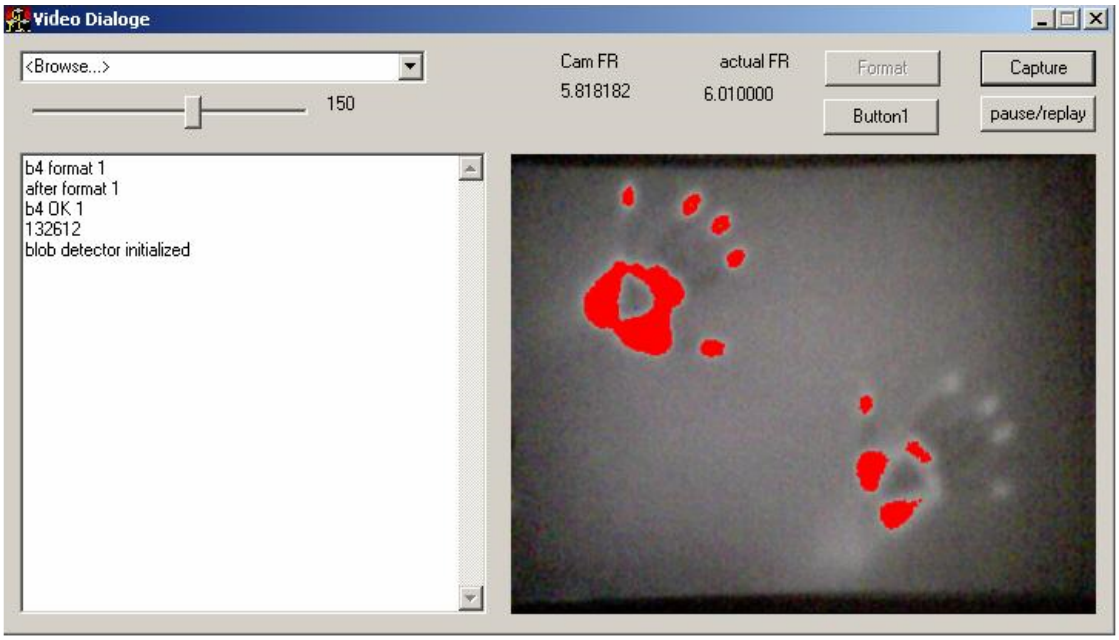


Figure 53 - Screenshot of the detection process in Touchwork

Chapter 5: The Tracker

5.1. Overview

After the Detection Module has done its job, we have information about each frame regarding how many blobs are available, their sizes, positions, and so forth. The next frame different information is produced, raising a new problem: When could we know if the user has added a new finger to the touch-surface, moved it around, or even removed it? The Tracking Module, **twTracker**, finds answers to those questions.

Module Information:

Input List of blobs in frame

Output List of current blobs (new and updated)
List of deleted blobs

5.2. Problem

Each frame passing by the Detection Module produces a list of the blobs detected in that frame. The task is to track down three main events, and update a static list containing the blobs currently available.

The events are defined as:

1. When a new blob appears (user puts down a new finger).
2. When an existing blob moves (user moves his finger around).
3. When an existing blob disappears (user removes his finger).

So, the problem appears when the Tracking module receives a list of detected blobs; the Tracking Module should identify the new blobs, the blobs that changed their locations and label them, as well as the blobs that disappeared. The Tracking module should maintain a static list of the current blobs in the system, as well as a list of the deleted ones – useful in later processing.

5.3. Requirements

1. **Robustness:**
   It must be able to track "significant" fingers even they become close to each other or exchange positions.
2. **Efficiency:**
   It must be fast enough so that the user does not feel any lag in the overall framework response.
3. **Modularity:**
   The tracking should be isolated from the detection, it should not require implicit detection internally for its algorithm to work; instead it should build its results based on data provided from the framework and resulting form the framework's detection module.
4. **Informative**:
   It should provide detailed information about the tracked fingers and changes accorded to their position.

5.4. Tracking Techniques

5.4.1. Image-Based Techniques

There are many ways to track blobs. The most popular are image based techniques as CamShift algorithm.

1. **CamShift:**
   CamShift stands for "Continuously Adaptive Mean Shift." This is the basis for the face-tracking algorithm in famous computer vision libraries (e.g. OpenCV). It combines the basic Mean Shift algorithm with an adaptive region-sizing step. The kernel is a simple step function applied to a skin-probability map. The skin probability of each image pixel is based on color using a method called *histogram back projection*. Color is represented as Hue from the HSV color model. **[Bradski]**

   Since the kernel is a step function, each iteration, the mean shift is simply the average x and y of skin-probability contributions within the current region. This is determined by dividing the first moments of the region by its zeroth moment at each iteration and shifting the region to the probability centroid.

   Mean Shift:
   The essence of this tracker is region matching. The match criterion is similarity based on a color histogram. Between frames, the region being tracked can change location, appearance (somewhat), and size. To examine possibilities quickly, the Mean Shift algorithm is used as a way to converge from an initial guess for location and scale to the best match based on the color-histogram similarity. **[Hewitt]**

   Advantages:
   a. Appears on initial testing to be moderately accurate. It may be possible to improve accuracy by using a different color representation.
   b. Capable of processing colorful images.
   c. Can track Fiducials and other shapes.

   Disadvantages:
   a. Relatively slow for tracking simple blobs and non complex objects.
   b. When similar blobs get very near, the algorithm can mix them up.

2. **Feature-based Tracking:**
   Tracks the motion of individual features, from frame to frame. The features are local, textured patches with gradients in more than one direction. Features that change appearance too rapidly during tracking are discarded. **[Shi]**

   Advantages:
   a. Capable of processing colorful images.
   b. Explicitly detects unstable features that are likely to be boundary features arising from accidental intersections.

   Disadvantages:
   a. Does not account for the effects of motion blurring.
   b. May get confused if there's a large inter-frame displacement.

In our solution, we preferred non-image based solutions. Convert the image to monochrome of grayscale, will result in not needing the power of tracking colorful images and degenerating performance. Besides, our approach is based on FTIR which will not use Fiducials or any complex objects, so we only have to track normal blobs resulting from finger tips touching the screen surface.

5.4.2. Object-based Techniques

There is no standard algorithm for tracking objects that is not based on image processing. It basically defines objects in a two-dimensional space (as blobs in the camera-frame space), meaning that objects are now defined in coordinates. The simple solution (matching each blob with all the other blobs getting the nearest) is subject to modification and tradeoffs of memory, complexity and speed.

1. **Brute Force Minimum Distance:**
   A simple matching done by simply getting the minimum distance between each two blobs in two subsequent frames.

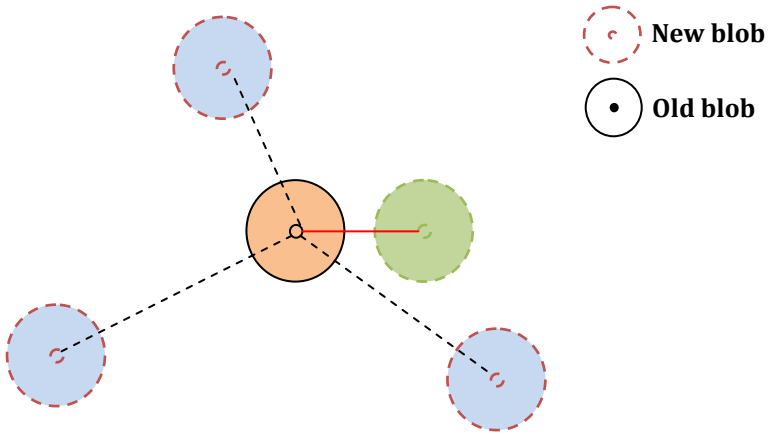


Figure 54 - Simple matching using Brute Force Minimum Distance

Advantages:

a. Very simple to implement.
b. In place (constant memory requirements).

Disadvantages:

a. Not robust, if two blobs interchange it will not detect the difference.
b. Time consuming – best running-time is $\Omega(n^2)$.

2. **Grid Matching:**
   Introduces a modification to the simple matching done the previous method, by only searching the nearby positions and omitting the other ones out of range.

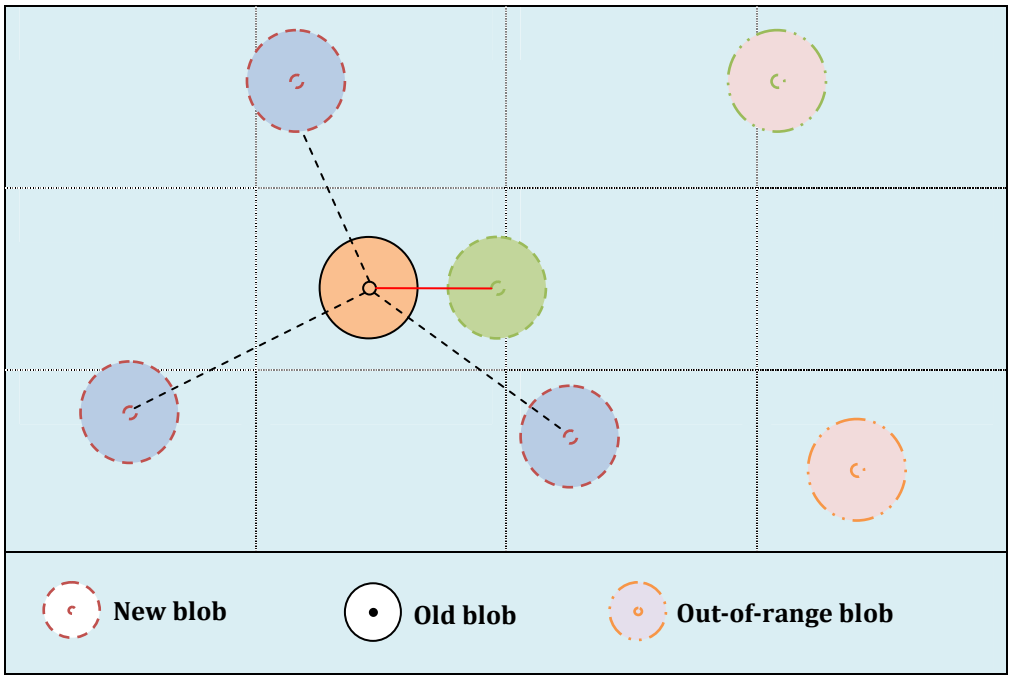


Figure 55 - Simple matching using Mean Square Distance

Although it seems very promising as it compares only blobs in the surrounding range, it can be very problematic in when it comes to addressing blobs. Also, assigning blobs to respective cells will be very time consuming. When a blob position changes, the need to remove it from a cell into another can waste valuable processing time. Moreover, the data structure representing this diagram will be very complicated, so usually another data structure will have to be created for simple access to blob data.

So, we decided to device a new algorithm that combines speed, less instructions, and a simple data structure.

5.5. The Tracking Algorithm

The tracking algorithm is a simple match-and-mark algorithm. Each blob has the following attributes:

m_id Carries the blob ID.

m_center Holds the x- and y-coordinates of the center of mass of the blob.

m_dist Represents the actual distance between the origin (0, 0) and m_center, the center of the blob (x, y).

m_checked A flag used in the tracking algorithm.

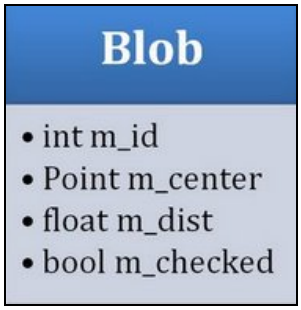


Figure 56 - Some variables in class Blob

The tracker is based on comparing the two lists of blobs each frame to find the closest two blobs within a constant distance range - a threshold - and update the list of current blobs. To ease the comparing, or the "searching" process, we suggested keeping the blobs in a sorted list, where the sorting would be based on the value of **m_dist**, the absolute distance between the origin and the center of the blob. It's somehow like putting all blobs on one straight line, where the first is the closest to the origin and the farthest is the farthest from it (the opposite corner).

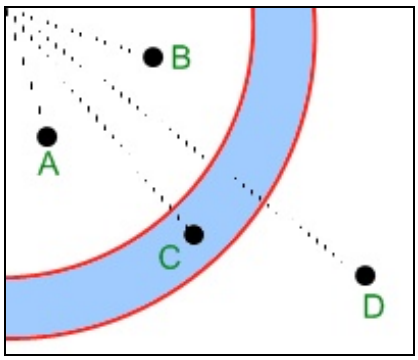


Figure 57 - Diagram showing positions of four detected blobs

Figure 57 shows a diagram of four sample blobs: **A**, **B**, **C** and **D**. The dotted line represents the distance between each blob and the origin (the value **m_dist**). Notice that blob **C** is wrapped in a thick belt. This belt-like range represents the threshold along that imaginary straight line (simply by adding and subtracting the threshold value from the **m_dist** variable.) Any blob that is outside or inside that belt (having **m_dist** value greater than that of blob **C** ± the threshold) will be excluded from the search process; increasing performance and reducing extra checking and comparisons (i.e. once blob **D** is checked and found outside the range of blob **C**, there is no point in resuming the search process). Notice also how blobs **A** and **B** could share very near values of **m_dist** as shown; which causes a problem that if blob **A** moved, both new **A** and **B** will match with the old **A**.

The static list of blobs, **currentBlobs**, is where we keep the updated positions of the available blobs only. Each frame, the module receives a list of blobs present in the frame. The list is compared against the current list to find the closest blob within the threshold, based on the **m_dist** variable as mentioned. If two new blobs match with an old one, the closest of them to the older one is picked. Matched blobs are marked using the flag **m_checked**.

Hence, the three main events are handled as follows:

1. **New blob appears:** It will be added to the **currentBlobs** list**.**
2. **Existing blob moves:** It will be updated and marked in **currentBlobs** list.
3. **Existing blob disappears:** The blob won't be marked at the end of the comparison process, so it will be removed from the **currentBlobs** list.

5.6. Implementation and Analysis

In the implementation of this algorithm, the class has one method that is invoked each frame, **Update-Blobs**. It starts by comparing each newly found blob to all old ones, trying to find the nearest candidate. If the old blob isn't in the belt-like range of the new blob, it is skipped - reducing massive performance overhead. After the search process is done, if the new blob has a match, then the old blob is updated to have the location of that new blob, and is marked by setting the **m_checked** to true. If not, this will indicate that the new blob was a truly new one, not a new location of an old blob. In this case, it will be inserted in the **currentBlobs** list (in increasing order of **m_dist**, to assist in the search process - explained shortly). Finally, all unchecked old blobs are removed from the **currentBlobs** list and added to a **deletedBlobs** list, that is used later to identify the events when the user removes a finger.

Here is a pseudo code for the function:

```
UPDATE-BLOBS(newBlobs)
      Initialize Variables
      Set m_checked in all currentBlobs to false

      // Match each new blob against current "old" blobs
      foreach Blob nb in newBlobs
          Calculate minBoundary & maxBoundary for search range

          foreach Blob ob in currentBlobs
              if ob is out of range
                   Break // and stop the search process...
              Else if ob is within range
                       Calculate the Mean Square Distance (MSD)
                           between centers of old and new blobs

                       if the MSD is less than threshold
                           Mark matchFound
                           Keep reference to the nearest old blob

                           if matchFound // this means that an old
                               // blob has changed its position to
                               // that of the new blob ...
                               Update blob with its new position
                           else
                               Insert new blob in currentBlobs
                               // sorted increasingly

      // Delete disappeared "old" blobs from currentBlobs list,
      // and put them in deletedBlobs list...
      foreach Blob ob in newBlobs
          if ob has no match // m_checked = false?
               Add ob to deletedBlobs
                  Remove ob from currentBlobs
```

Figure 58 - Pseudo code of the tracking algorithm

The point behind maintaining the **currentBlobs** list sorted is to reduce the running time of the algorithm. During the search process, all old blobs *behind* the belt-like range (with **m_dist** less than **minBoundary**) are skipped, until we are *inside* the range where the real comparison begins. Whether a match was found or not, once the old blobs are now outside the range (with **m_dist** larger than **maxBoundary**), then there is no point of resuming the search process. So, two simple checking instructions would reduce the running-time.

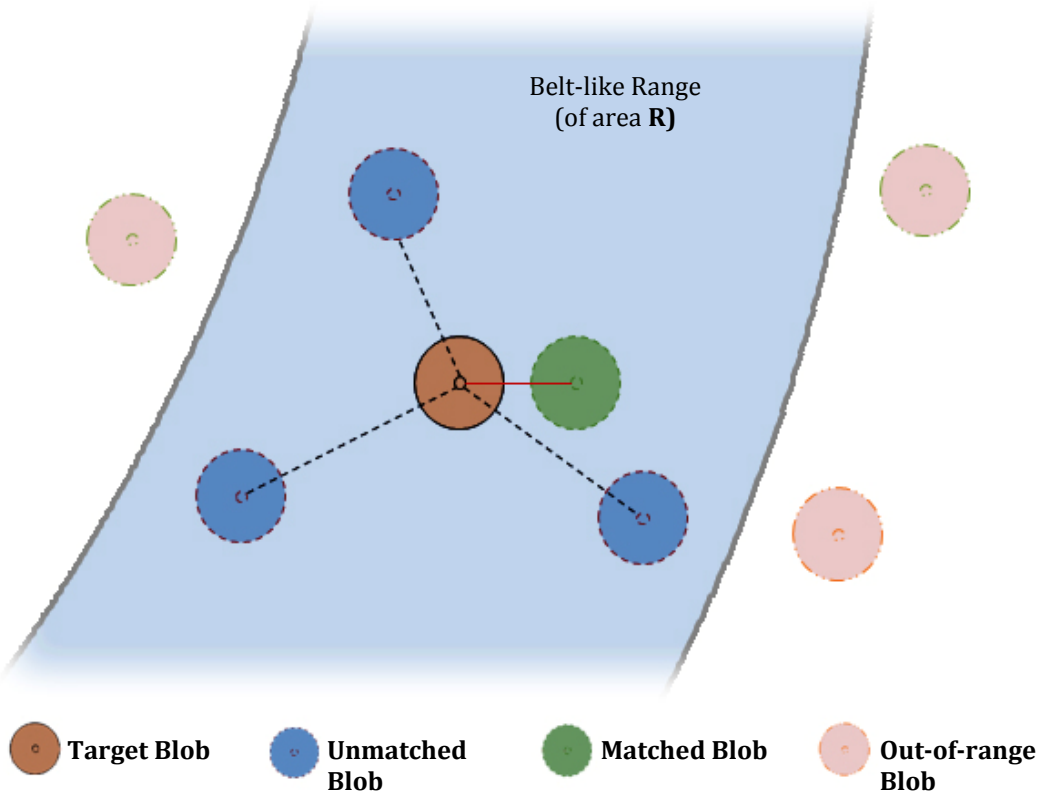


Figure 59 - Tracking Algorithm

It is also noticed that for every frame, there is a fixed number of blobs **n**, where for each blob a fixed area (the belt-like range) **R** is scanned to find a matching blobs. Taking in consideration that this fixed area **R** will have a maximum number of contained blobs **b**, this will result in a fixed number of comparisons to find a match. Hence, a constant number of instructions will be executed per new blob to find its match, leaving us with a best-case running-time of Ω (**n**). However, if the threshold (width of the belt-like range) increase, it will eventually contain all **m** old blobs, for a worst-case running-time of O (**n***m**). Therefore, the average running-time of the algorithm will be θ (**n**). Usually, the number of blobs will range from two to three blobs per frame, and won't exceed ten blobs on average.

Chapter 6: The Agent

6.1. Overview

The Agent Module, **twAgent**, is the link between the applications and the Touchwork framework. The Agent analyses the tracking information (the output of the Tracking Module) of the user's fingers and then generates the proper events as a response then prepares them to be dispatched to the application. It exposes standard interfaces for the application to use in interacting with the framework. The Agent allows for two way interaction so that the application can receive information from the framework as well as notifying it with farther needs. In case the application is not interested in input events form touch screens that were not build to support them, the Agent provides a set of default actions and executes them in behalf of that application.

The module is composed of three main components:

1. **Analyzer:** Analyses the input from the Tracking Module **[Chapter 5]** and produces the appropriate events.
2. **Dispatcher:** Notifies the applications of the new input events.
3. **Processor:** Executes the default actions in response to the input events.

Different layouts and architectures of these components will be discussed in **[Sections 6.3 and 6.4]**.

Module Information:

Input Touch positions – blob tracking data

Output Messages and notifications about touch information (sent to applications)

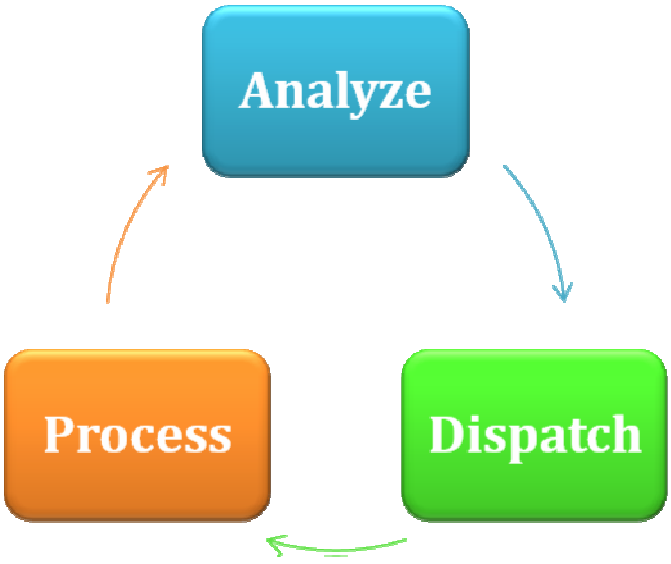


Figure 60 - The Agent Module Cycle

This cycle is executed every frame on the tracked data

6.2. Requirements

1. **Transparent to developers:**
   Developers should not notice the existence of the Agent unless they want to interact with Touchwork.
2. **No security privileges required:**
   The program should not be required to obtain any security privileges in order to interact with the framework. So, network interaction is not very suited for the Agent, although other solutions **[Chapter 9]** like TUIO use UDP packets to interact between their agents and their capture modules, which may require extra privileges to be granted to the application in question, and the user might refuse to grant the application these rights which renders the application disconnected with the framework and unable to receive any input.
3. **Extensible:**
   Extensions to the information exchanged between the framework and the application can be added easily, the framework should provide means of doing so.
4. **Overridable:**
   The Agent must be built in a way that allows other applications to override the default event processing.
5. **Global:**
   Provides means for applications to intercept the input sent to other applications.
6. **Sparse data exchange:**
   The amount of data sent to an application must be those only required by the application to interact, to minimize communication overhead, and any farther information required can be asked for on demand.
7. **Easy to use for developers:**
   The work required by the application developers should be as little as possible the Agent should provide a mean for developer to harness the power of Touchwork without having to understand the communication details.
8. **Flexible and Powerful:**
   For power developers who want to interact with Touchwork at a low-level, the Agent must provide ways to do so.

6.3. Possible Solutions

6.3.1. Communication with the Framework:

Possible solution for the communication between the Touchwork framework and Applications:

1. **UDP packets (a network stream):**
   The applications can listen at a UDP port on which it can receive tracking information from Touchwork.

   Advantages:
   d. Supports more than one machine (a machine can run applications while another one can process the input from the camera)
   e. Supports virtualization. A guest machine can host applications, while the host can run the framework, which is good for testing applications during development. The two machines can interact through UDP easily as network interaction is supported by most virtualization platforms (e.g. Microsoft Virtual PC, Xen, VMWare or VirtualBox).
   f. The different machines in this system can run different platforms. They are not required to be homogenous since the support for UDP is standard in every popular platform (e.g. Windows, Mac or Linux).
   g. Support for multicast is built-in by default due to reliance on UDP, which can ease the communication management on the Framework.

   Disadvantages:
   a. Involves communication redundancies as every interested application will receive input information of the whole system, even if its windows are hidden, minimized or not affected by the input.
   b. Places an extra burden on every application, as it will be required to filter the input information then process only the input data that directly affects it (input data at the positions obscured by the application's windows or controls). This very processing will be repeated by every application in question unless a common broker application is created which will be duplicating the Windows Messages subsystem.
   c. Not intuitive from the application development perspective.
   d. Introduces a different input paradigm from that used generally in popular platforms (for mouse and keyboard input).
   e. Visually oriented communication is hard to implement. In other words, the UDP packets destination is hard to be associated with a certain window.

2. **TUIO**
   This protocol definition is an attempt to provide a general and versatile communication interface between tangible table-top controller interfaces and underlying application layers. It was designed to meet the needs of table-top interactive multi-touch surfaces

   The protocol's flexible design offers methods for selecting which information will be sent. This flexibility is provided without affecting existing interfaces, or requiring re-implementation to maintain compatibility.

Implementation Details:
The TUIO protocol defines two main classes of messages: Set Messages and Alive Messages. *Set Messages* are used to communicate information about an object's state such as position, orientation, and other recognized states. *Alive Messages* indicate the current set of objects present on the surface using a list of unique session IDs.

In addition to Set and Alive Messages, fseq Messages (frame sequence) are defined to uniquely tag each update step with a unique frame sequence ID. To summarize:

- Object parameters are sent after state change using a Set Message.
- An Alive Message is sent on object removal.
- The client deduces object addition and removal from Set and Alive Messages.
- Fseq Messages associate a unique frame ID with a set of Set and Alive Messages.

Advantages:

Since it relies on UDP as a transportation layer, it has the same advantages as UDP packets.

Disadvantages:

a. It misses some parameters which might be useful in current environments, such as finger identification and pressure values.
b. To avoid possible errors evolving out of packet loss, no explicit add or remove messages are included in the TUIO protocol. The receiver deduces object lifetimes by examining the difference between sequential alive messages.

3. **Named Pipes:**
A pipe is a stream between two applications. One application can create the pipe, associating it with a certain name. The other application can just obtain the pipe form the operating system using that name. Pipes are very similar to UDP streams; having very similar advantages and disadvantages.

4. **Windows Messages:**
Whenever an application creates window, a message queue is associated with this window, in which the applications receives input messages related to that window. Other applications can send messages to that application by targeting that window as the destination of their messages.

Advantages:

a. Well known to application developers, and commonly used.
b. Window oriented and optimized to handle input data exchange. So, Touchwork will not have to find the application, it will just send the message to the window at the input position.
c. Supported by every application that has a GUI front.

Disadvantages:

a. Not cross platform.
b. Does not support virtualization by default.
a. Extra layers should be added to support extra machines to process the input.

6.3.2. Basic processing:

Possible solutions for places to execute the basic processing:

1. **The framework side**:
   The default behavior of the windows in response to the touch events can be executed by the framework.

   Advantages:
   a. Easy to implement.

   Disadvantages:
   a. The application might be overriding the default event response. The framework must be aware of the needs of every application that might even be changing during the application's lifetime.
   b. Not intuitive to write the logic for application behavior in the framework, while it is easier to think of a window behavior inside that windows own context.

2. **The application side:**
   The default behavior can be placed inside the application, inside a DLL that is linked to the applications when needed.

   Advantages:
   a. Intuitive, as the code runs inside the applications context.
   b. Powerful, allows for maximum access to the application's resources.

   Disadvantages:
   Hard to implement, either using Windows Hooks or using DLL Injection.

6.4. Implementation

6.4.1. The Analyzer

The Tracking Module defines a set of blobs each frame. The analyzer component raises an event for every blob after the tracker defines its state as one of the following states:

a. **Newly Added Blobs:**
Blobs that have just been added to the set are marked as new (no matches found for it in the old set). The analyzer associates a *Touch Down* event with each.

b. **Deleted Blobs:**
Blobs that disappeared from the new set, (found in the old sets with no matches in the new set), are marked as deleted. The analyzer associates a *Touch Up* event with each.

c. **Moving Blob:**
Blobs that moved, (found in both the old and new sets but with slight variation in position **[Section 5.4]**, are marked as moving, the analyzer associates a *TouchMove* event with each.

d. **Still Blobs:**
Blobs that did not change their state (or position) during transition between frames are marked as still; the analyzer does not associate any events with any of them.

The following code demonstrates this operation:

```
foreach(blob in currentBlobs)
{
	if (blob->isMoved)
	{
		// move
		NotifyWindowUnderBlob(WM_TOUCH_MOVE);
	}
	else if( blob->isNEW )
	{
		// new -> TOUCHDOWN
		NotifyWindowUnderBlob(WM_TOUCH_DOWN);
	}
}

foreach(blob in deletedBlobs)
{
	// delete -> TOUCHUP
	NotifyWindowUnderBlob(WM_TOUCH_UP);
}
```

Figure 61 - Code snippet from the Analyzer

6.4.2. The Dispatcher

This component uses Windows Messages as the dispatching mechanism of messages to the interested applications.[12] The Agent knows the position of every blob, its ID (unique number that separates it from other blobs) and the amount of pressure imposed by the finger producing that blob.

Steps:

1. Map the blob position from the image space to the screen space (transformation step) using this simple transformation:

```
xScreen = xCamera * xScreenToCamera;
yScreen = yCamera * yScreenToCamera;
```

   Providing the Screen to Camera ratio in pixels, where xScreen and yScreen are the x- and y-coordinates on the screen, xCamera and yCamera are the x- and y-coordinates in the image and xScreenToCamera and yScreenToCamera are the scaling factors (screen to camera rations) in the x- and y- direction respectively.

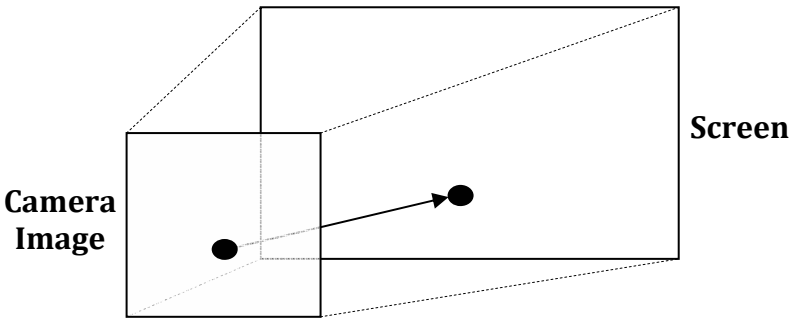


Figure 62 - Mapping points from image space to screen space

2. Find the window under the mapped point.
3. Map that point from screen coordinates to window coordinates.
4. Encode the blob characteristics (mapped position, ID, pressure) into windows message parameters.
5. Send the message asynchronously to the window. The message is sent without waiting for response from the receiver window in case the window spends a long time processing the input, the framework will be idle waiting for response instead of processing farther input which affects the overall responsiveness of the framework and other application. So Touchwork does not wait for any response from the receiving window.

[12] Details about the messages defined by Touchwork SDK can be found in the section 6.5.1.

We can post a message to a message queue by using the *PostMessage* function. **PostMessage** places a message at the end of a thread's message queue and returns immediately, without waiting for the thread to process the message. The parameters of the function include a window handle, a message identifier, and two message parameters. The system copies these parameters to an **MSG** structure, fills the **time** and **pt** members of the structure, and places the structure in the message queue. **[MSDN-Message Queues]**

This is a code snippet demonstrates this process:

```
void TwAgent::NotifyWindowUnderBlob(Blob& blob, UINT messageType)
{
	// get the blob position
	POINT pt = blob.center.getPOINT();

	// map it to the screen space
	CameraToScreen(pt);

	// locate the window under it
	HWND hwnd = WindowFromPoint(pt);

	if(hwnd)	// if a window is found
	{
		CWindow win(hwnd);

		// map screen space to window client space
		ScreenToClient(win, &pt);

		// encode input info
		WPARAM wparam = MAKEWPARAM(blob.id, blob.pointscount);
		LPARAM lparam = MAKELPARAM(pt.x, pt.y);

		// notify the screen of the new message
		PostMessage(win, messageType, wparam, lparam);
	}
}
```

Figure 63 - Code snippet for posting windows messages with blob states

A window receives the message at the end of its message queue. The message queue stores all the input messages sent to the window, waiting for the window to dequeue them and execute the appropriate response.
Events are processed in a FIFO manner (First In First Out) **[MSDN-Message Queues]**. When the window is ready, it pops a message from the beginning of the queue, calls the appropriate event handler depending on the event type, which processes the event. Then the window pops another message and so on. **[MSDN-Windowing]**.

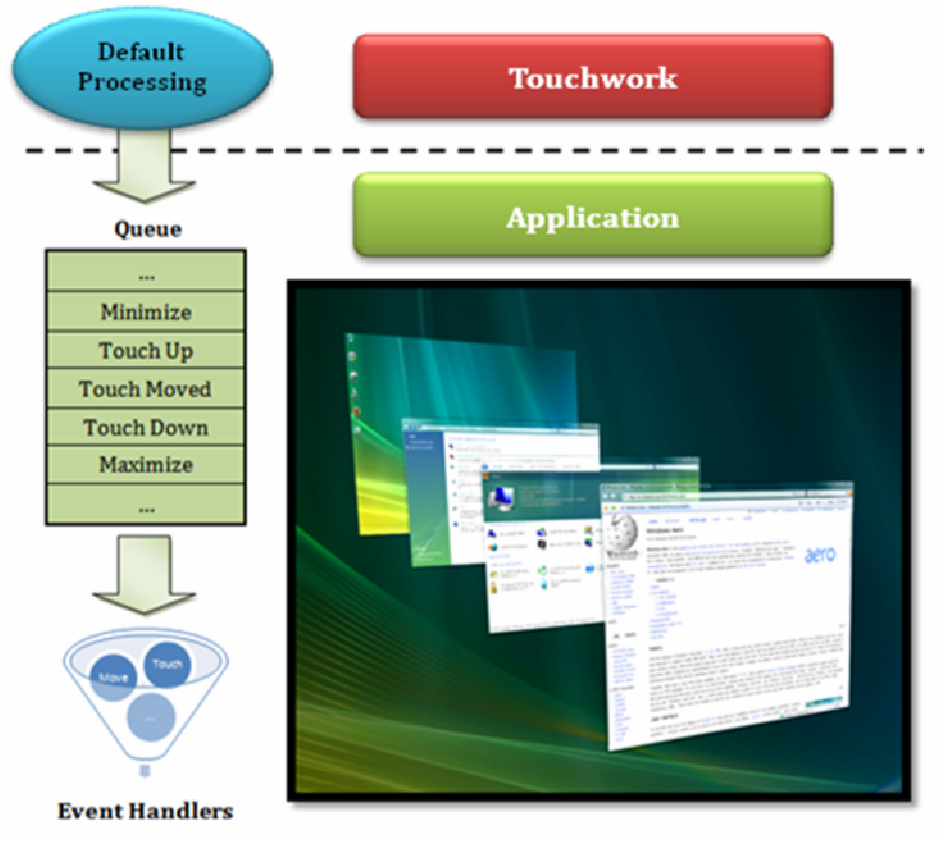


Figure 64 - Touchwork Dispatching Mechanisms

6.4.3. The Processor

This component is responsible for making the windows respond to touch events even if the developer did not add Touchwork support or handled touch events during the development.

The main idea is to handle the events sent to the window from Touchwork executing basic effects on the window such as:

1. **Taping**:
   When the user taps on a button with his finger. The (TouchDown/TouchUp) event pair is translated into (MouseDown/MouseUp) pair, which will make windows animate the button as if it was clicked by the mouse executing its click handler.
2. **Moving**:
   When the user touches a control and moves his finger, if the element does not respond to touch events, the events (TouchDown/TouchMove/TouchUp) are translated into (MouseDown/MouseMove/MouseUp) which causes the control to follow the finger as if it was a mouse.
3. **Resizing**:
   When the user touches the title bar of a window and any other part of the window, and executes a zoom gesture, these messages are translated to a move and resize event.

These actions can be overridden by the application in order to present a custom handling for the Touch Messages. The code behind this logic is compiled into a DLL (Dynamic-Link Library) which is injected into each application to run inside its context. The DLL is injected using a Global Windows Hook.

A *global hook* monitors messages for all threads in the same desktop as the calling thread. The global hook procedure can be called in the context of any application in the same desktop as the calling thread, so the procedure must be in a separate DLL module. If the application installs a hook procedure for a thread of a different application, the procedure must be in a DLL. **[MSDN-Hooks]**

That will cause the code to be injected inside every application that owns a window and will enable that code to receive every message sent to the window and to control that window accordingly.

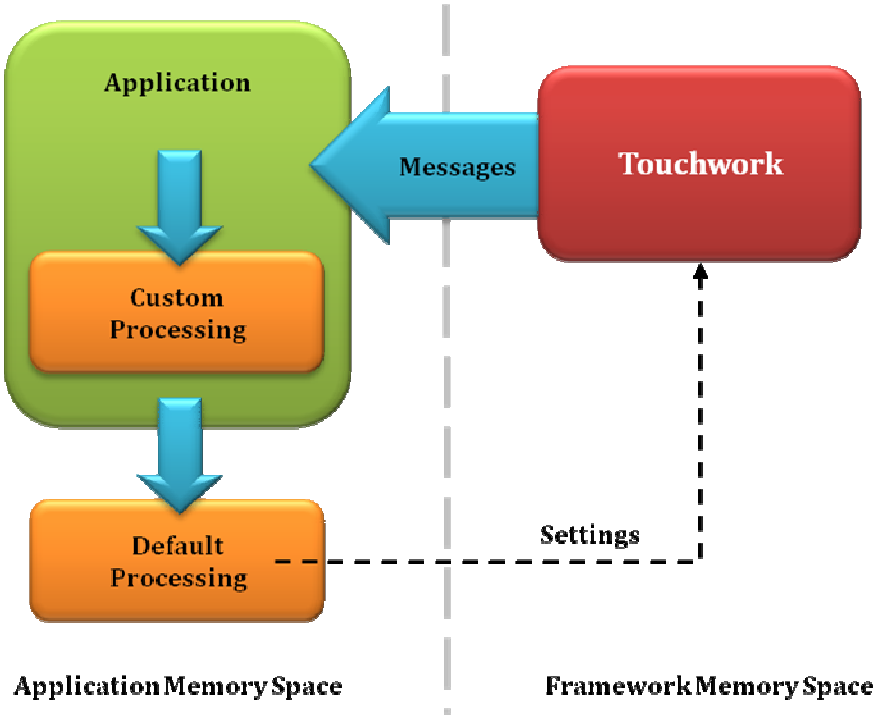


Figure 65 - The processing component placement inside the application layout

6.5. Specifications

Windows messages are the mean of communication preferred for reasons detailed in Section 6.3.1.

6.5.1. Types of Exchanged Information:

1. **Notifications:**
   The framework notifies the application of new actions performed by the user.

Type	Message	Description
Touch Down	WM_TOUCH_DOWN	A new finger touches the screen in a position obscured by a window owned by the application receiving the message.
Touch Move	WM_TOUCH_MOVE	A finger moves while still maintaining its contact with the screen's surface.
Touch Up	WM_TOUCH_UP	A finger loses contact with the surface.
Tap	WM_TOUCH_TAP	A finger taps on the screen at a position inside the window.

Table 8 - Types of notification massages sent by Touchwork

Details about the content of each message are in the next section.

2. **Queries:**
   The application can request for information from the framework.

Type	Message	Description
Touches Query	TW _TOUCHES	Requests a list of all the fingers having contact with the application.
Version Query	TW _VERSION	Requests an integer representing the version number of the framework so that the application can judge the compatibility level.
Support Query	TW_SUPPORT	Requests a Boolean represents the existence of support for a certain feature in the current version of Touchwork.

Table 9 - Types of notification messages sent by the applications

6.5.2. Touchwork Messages

1. **TouchDown Notification:**
   The WM_TOUCH_DOWN message is posted when the user touches the client area of a window. The message is posted to the window beneath the finger. A window receives this message through its *WindowProc* function.

   Syntax:

```
WM_TOUCH_DOWN
      WPARAM wParam
      LPARAM lParam
```

   Parameters:
   - **wParam:**
     The low-order word specifies the unique identifier of the touch, which is never repeated for this message. The high-order word specifies the pressure value of the touch.
   - **lParam:**
     The low-order word specifies the x-coordinate of the finger. The high-order word specifies the y-coordinate of the finger. Both coordinates are relative to the upper-left corner of the client area.

   Remarks:
   Although the framework processes input in a frame-by-frame basis, it will never send WM_TOUCHDOWN message to the same application window twice in two subsequent frames, in other words, the framework will notify the application of that touch only once, during the frame in which the touch was first presented. If the application did not store that information and wants to receive any information regarding the position of that touch it should query the framework for the all fingers position.

2. **TouchMove Notification:**
   The WM_TOUCH_MOVE message is posted when the user moves his finger inside the client area of a window. The message is posted to the window beneath the finger. A window receives this message through its *WindowProc* function.

   Syntax:

```
WM_TOUCH_MOVE
      WPARAM wParam
      LPARAM lParam
```

   Parameters:
   - **wParam:**
     The low-order word specifies the ID (unique identifier) of the touch. This ID is the same as that of received in the TouchDown notification. The high-order word specifies the pressure value of the touch.
   - **lParam:**
     The low-order word specifies the x-coordinate of the finger. The high-order word specifies the y-coordinate of the finger. Both coordinates are relative to the upper-left corner of the client area.

Remarks:
Due to the image-based detection techniques, the finger will not be considered as moving unless it executes a large enough displacement from the original position with which it started the touch. Touchwork will keep sending WM_TOUCHMOVE message to the application as long as the finger moves, i.e., until it stops moving.

3. **TouchUp Notification:**
The WM_TOUCH_UP message is posted when the user lefts his finger from the client area of a window. The message is posted to the window beneath the finger before it was removed. A window receives this message through its *WindowProc* function.

Syntax:

```
WM_TOUCH_UP
      WPARAM wParam
      LPARAM lParam
```

Parameters:

- **wParam:**
  The low-order word specifies the ID (unique identifier) of the touch. This ID is the same as that of received in the TouchDown notification. The high-order word specifies the pressure value of the touch.
- **lParam:**
  The low-order word specifies the x-coordinate of the finger. The high-order word specifies the y-coordinate of the finger. Both coordinates are relative to the upper-left corner of the client area.

Remarks:
The ID sent in this message will never appear in the list of touches that the framework returns on demand.

6.6. The Simulator

During development, the developer might need to test his application's interaction with Touchwork, in case that the hardware (the touch screen) is not available, he can use this simulator that sends touch input messages to the application to test its response. The user can simulate finger touches on the surface, move the fingers, or remove them.

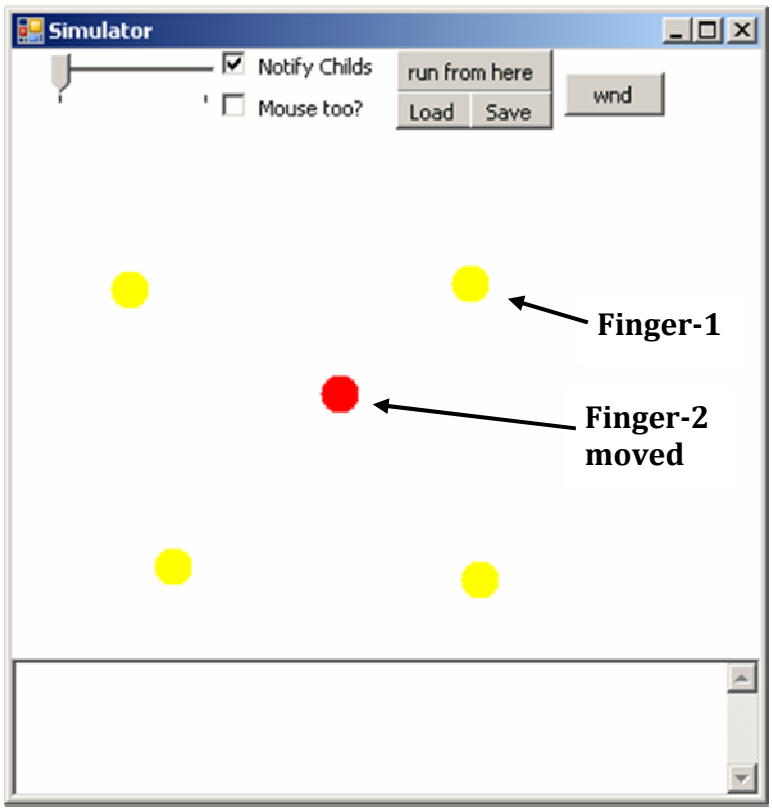


Figure 66 - Screenshot of the Simulator

The Simulator maps the finger position from its client space into screen space then to the client space of the target window (figure 67).

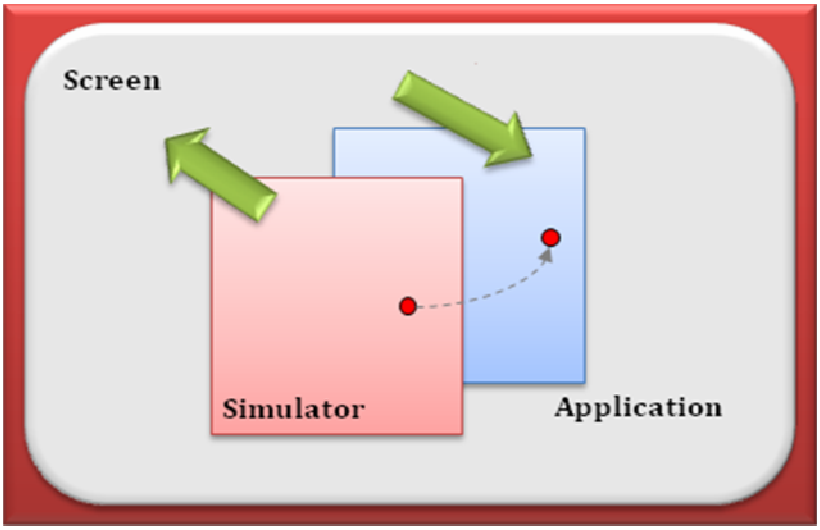


Figure 67 - Mapping positions from the simulator window to the target application window

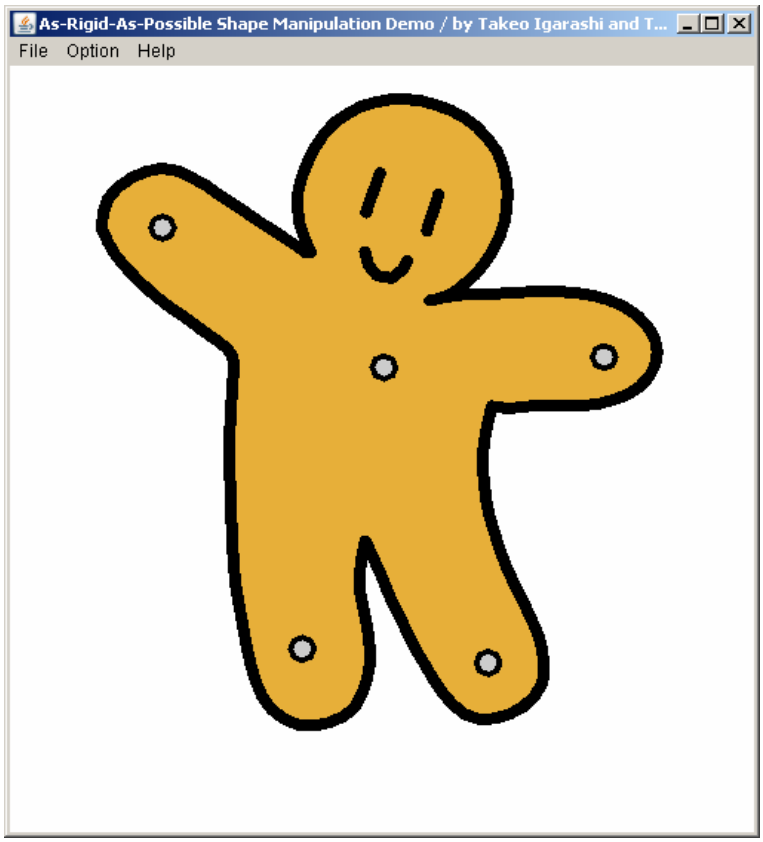


Figure 68 - An application that can respond to touch events

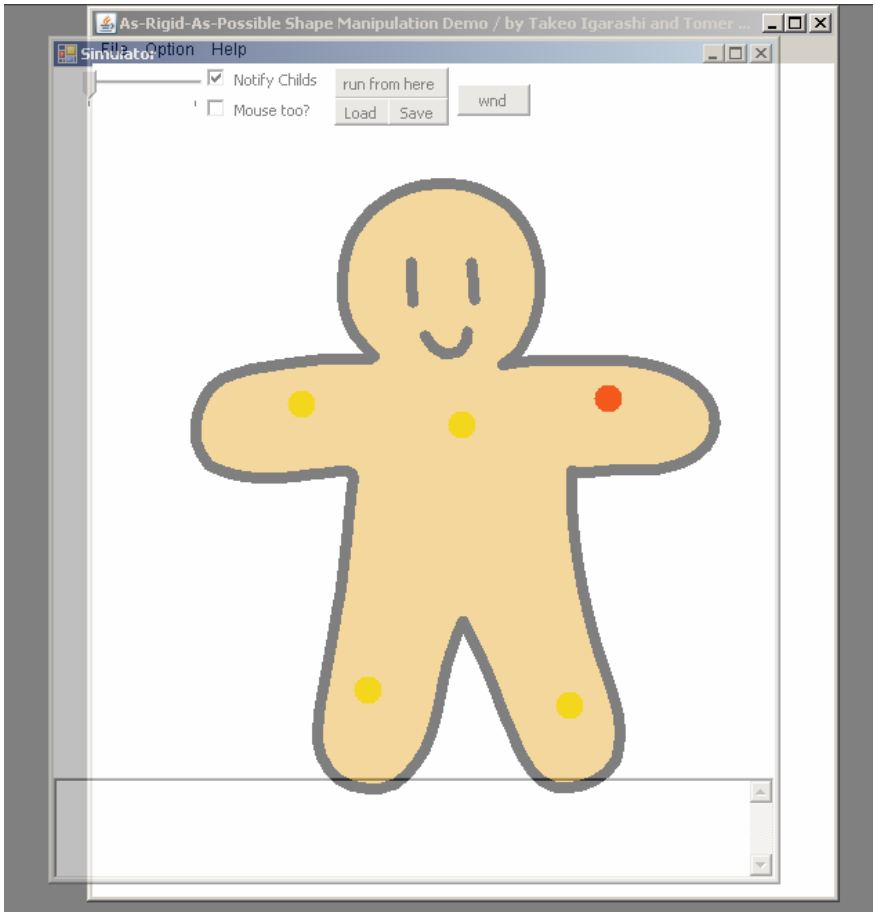


Figure 69 - The Simulator running an animation demo

6.7. Wrappers

Touchwork SDK supports programming in any language that can interact with windows messages and use windows Win32 APIs (e.g. C/C++, Pascal, and even Assembly). The SDK also provides C/C++ and C# bindings to use with Touchwork Framework:

6.7.1. C/C++ Wrapper

Touchwork SDK provides C/C++ header files that contain helper functions and classes to use in communicating with the framework along with the Touchwork Messages definitions (figure 70).

```
#define WM_TOUCH_UP      WM_APP + 0
#define WM_TOUCH_DOWN    WM_APP + 1
#define WM_TOUCH_MOVE    WM_APP + 2
```

Figure 70 - Message Definitions (from twMessage.h)

WM_APP is the minimum windows message identifier that can be used in cross-application interaction, any value less than WM_APP will interfere with System defined messages.

6.7.2. TouchworkNet (.NET and C#)

C# does not expose Windows APIs by default. Instead, it uses Event Handling with a Sender/Subscriber mechanism, where the developer defines callbacks and subscribes to the events he wants to receive, and the framework notifies the callbacks he defined on the occurrence of any of these events. The C# version, **TouchworkNet**, contains a helper class that extends .NET Controls making them aware of Touchwork messages and to expose the appropriate events handlers to the developers along with the Touchwork specific information.

TouchworkNet Contents:

1. **An interface** is defined so that the application can implement in order to receive Touch Events.

```
namespace TouchworkSDK
{
      public interface ITouchable
      {
            void OnTouchDown(object sender,TouchEventArgs e);
            void OnTouchUp(object sender,TouchEventArgs e);
            void OnTouchMove(object sender,TouchEventArgs e);
      }
}
```

Figure 71 - ITouchable C# Interface

2. **Event Arguments** that provide detailed information about the touch event including the touch position, pressure and ID. It also performs the encoding of the window message parameters to the appropriate representation in C#.

```
namespace TouchworkSDK
{
    public class TouchEventArgs : EventArgs
    {
        public Point point;
        public int pressure;
        public int ID;
        public bool handled = false;
        public TouchEventArgs(uint lparam, uint wparam)
        {
            uint x = 0, y = 0;
            GetHighLow(lparam, ref y, ref x);

            uint id = 0, pressure = 0;
            GetHighLow(wparam, ref pressure, ref id);

            point.X = (int)x;
            point.Y = (int)y;
            this.pressure = (int)pressure;
            this.ID = (int)id;
        }

        private void GetHighLow(uint number, ref uint h, ref uint l)
        {
            uint temp = (uint)number;
            l = temp & 0x0000FFFF;

            temp = temp & 0xFFFF0000;
            h = temp >> 16;
        }
    }
}
```

Figure 72 - Touch Event Arguments class

3. **Touchwork Class** that provides the means to extend .NET controls to listen to Touch Events.

```
public class Touchwork
{
      static public void NotifyOnTouch(ITouchable touchableControl)

      static public  void UnsubscribeToTouchEvents(Control ctrl)

      static public void NotifyOnTouch
         (Control control, ITouchable callback)
}
```

Figure 73 - Touchwork class

First the application implements the `ITouchable` interface, and then passes it to the `Touchwork.NotifyOnTouch()` function along with the control that needs to receive touch events, for then the SDK will notify the application on the occurrence of these events.

Part IV

[Beyond Touchwork]

Output, Demos, Similar Work, Future Work

Chapter 7: Output

7.1. Framework Deliverables

The final deliverables of the framework are:

1. **FTIR based hardware prototype.**
2. **Framework Executable – Touchwork.exe:**
   This executes the main logic behind the framework. This executable is a singleton that can only have one instance on a machine, and it is launched at system startup.
3. **Configuration Executable – Config.exe:**
   That helps users to setup the framework to best fit their hardware and environment, along with an editable *Configuration File* that stores the device configuration. It allows the user to set:
   a. **Image Calibration**:
      The user can modify the preprocessing done by the framework on the video stream received from the hardware, as noise removal and background separation.
   b. **Cropping**:
      The user can force the framework to only process a certain area of the captured images, in case if the camera exposes redundant areas where no touches are detected.
4. **Development SDK:**
   Contains the necessary development tools to help the developers with creating applications that harness the power of multi-touch technologies. The SDK contains the following components:
   a. **C++ bindings to Touchwork**:
      A set of C++ classes and functions meant for Game and Multimedia Developers that develop applications that require high performance and efficient use of memory and processing capabilities. **[Section 6.7]**
   b. **.NET bindings to Touchwork**:
      A set of interfaces and classes that are meant for developing applications that benefit from Microsoft .NET Framework, its GUI capabilities and ease of development. **[Section 6.7]**
   c. **Touchwork Simulator**:
      A tool that helps developer test how their applications interact with Touchwork in case of the absence of a hardware unit to perform live testing. The Simulator sends similar input events through Touchwork, as if a real hardware unit was present. **[Section 6.6]**

7.2. Minimum System Requirements

The minimum system requirements for the binary distribution:

Hardware Requirements:	
Processor	*Intel* Pentium™ Compatible Processor 1GHZ, 1MB Cache (Dual Core is recommended)
Memory	256MB of RAM
Peripherals	Digital Web Camera, *30fps*

Software Requirements:	
Operating System	Microsoft Windows 2000 Microsoft Windows XP™ (recommended) Microsoft Windows Server 2003 Microsoft Windows Vista™
Additional Software	.Net© Framework 2.0 or higher Microsoft DirectX 9.0c© or higher

Table 10 - Hardware and Software Minimum Requirements

7.3. Source Code Compilation Dependencies

Extra dependencies for compiling the source code:

- Windows® Server 2003 R2 Platform SDK
- Microsoft DirectX SDK April 2004
- Microsoft .Net 2.0 SDK
- Microsoft Visual Studio .Net© 2005 or higher
- C/C++ Compiler, (recommended Microsoft VC++ 7 or higher)
- *Intel* OpenCV, **[OpenCV]**

Note: Web references to download all these software files are in the reference section under [Software].

7.4. Hardware Testing

Benchmark tests were executed on every component to inspect its performance and efficiency in achieving the desired results.

7.4.1. Infrared Webcam

Requirements:
The webcam must be able to detect infrared light as visible light and display it on the software.

Expected Output:
Camera Software should be able to detect infrared light emitting from a TV-Remote Control Unit forming a very bright light spot.

Test plan:
Remove infra-red filter from the web-camera, following steps detailed in *Building the Hardware*. **[Section 2.4]**

Results
Trial 1 – PASS

Conclusion
Still the webcam sees the visible light. Adding two layers of exposed film negative enabled blocking visible light.

7.4.2. Acrylic Sheet

Requirements:
Side-illuminated acrylic must respond to touches by dissipating light blobs that can be detected by infrared enabled webcam.

Expected Output:
The webcam should detect light blobs at the positions where the fingers touch the acrylic sheet as described in *Building the Hardware.* **[Section 2.4]**

Test plan:
After the sides of the acrylic sheet are polished, an array of infrared LEDs should be aligned to the sides of the sheet. The webcam is placed on the side opposite to the one the user touches to resemble the description mentioned in *Building the Hardware*. **[Section 2.4]**

Results
Trial 1 – PASS

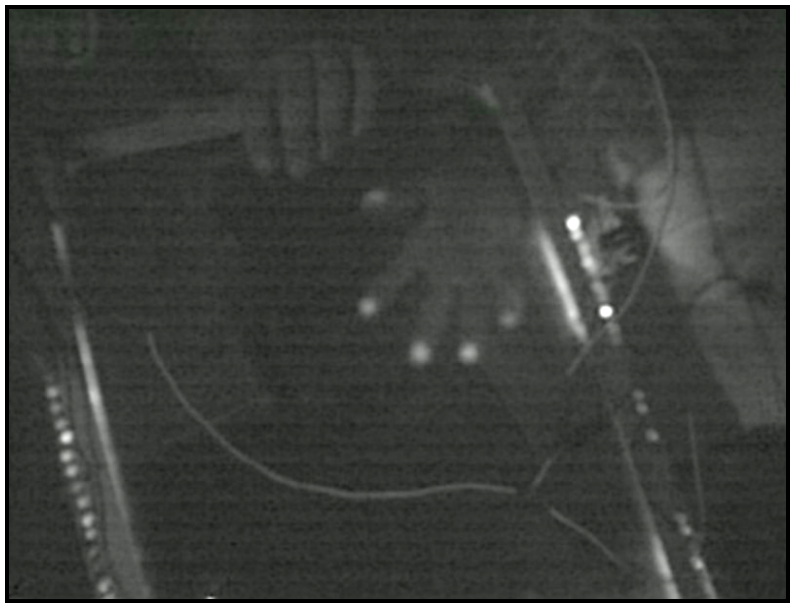


Figure 74 - Acrylic Sheet Test - Trial 1

Problems:

LEDs-Array must be held perpendicular to the acrylic surface. Refer to building a metal frame holder, *Building the Hardware*. **[Section 2.4]**

Conclusion

Even though the quality of the acrylic sheet isn't as expected (clear), it still worked. It was fine, not the best though.

7.4.3. Compliant Surface:

Requirements:

A compliant surface overlay placed over the acrylic sheet, which couples with it under pressure to set the FTIR effect, and decouples once the pressure is released. **[Section 2.4]**

Expected Output:

Clear blobs should appear when fingers touch the additional layer.

Test plan 1 – Spreading Crude Silicon:

1. Buy a silicon sealant, (available in car workshops)

Figure 75 - Silicon Sealant

2. Apply silicon to the surface.

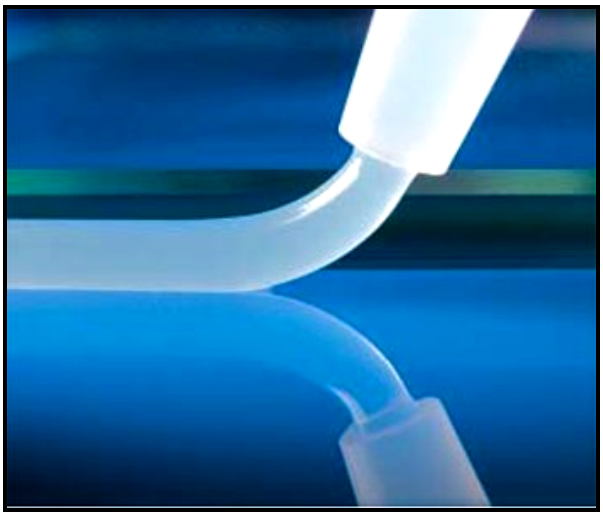


Figure 76 - Applying the Silicon to the acrylic surface

3. Spread silicon on the acrylic sheet (with a spoon), creating a very thin coating.

Results

Trial 1 – FAIL

Infrared light was reflected on the silicon layer, so fingers were drawn completely invisible to the webcam.

Test plan 2 – Spraying Liquefied Silicon:

1. Prepare the solid silicon sealant, and toluene (a solvent).
2. Prepare a glass beaker or insolvent plastic container (used usually to contain color paints and pigments).
3. For every 3 cm^3 of toluene, apply 1 cm^3 of silicon (3:1 ratio). Stir until the silicon becomes completely liquefied (figure 77).

Figure 77 – Liquefied Silicon with added Toluene and spray bottle

4. We have around 10 minutes before the Toluene evaporates and the silicon re-solidifies.
5. Pour the mixture into a glass spray bottle.

6. Spray the mixture to the acrylic sheet, and leave it till it dries.
7. Test the resulting layer.

Results

Trial 2 – FAIL

Infrared light reflected on the silicon layer, even with more brightness than the first trial.

Test plan 3 – Pouring Liquefied Silicon:

1. Place two rulers (1 mm thick) on the sides of top surface of the acrylic sheet.
2. Prepare the silicon as in steps 1 through 5 as in Test plan 2.
3. Pour the mixture directly onto the acrylic sheet.
4. Use a third ruler to evenly spread and distribute the silicon over the acrylic sheet and create a 1 mm thick layer depending on the other two rulers.

Figure 78 - Spreading the silicon layer with a ruler

5. Leave the mixture till dries completely, then test it

Results

Trial 2 – FAIL

Infrared light reflected on the silicon layer, even with more brightness than the first trial.

Conclusion

It seems that the chemical properties of the silicon sold in Egypt differ from those used abroad. The one we have doesn't pass enough light through it, and hence the fingers need to be pushed very hard against just to show a very light blob, if any.

7.5. Performance Analysis

Performance analysis was done for the framework components against their counterparts.

7.5.1. The Detection Module

The detection module was tested against OpenCV's blob detection component. The following table shows the average time (in seconds) required by each solution to track a set of blobs.

Image Size (Mega Pixels)	**OpenCV**	**Touchwork**
2	0.1678	0.099
4	0.2317	0.1173
8	0.2611	0.1318
15	0.3572	0.1559

Table 11 - Results of comparing the detection in OpenCV and Touchwork

While OpenCV's algorithm proves to be efficient and it is completed in one pass, detailed information about its source does not exist in OpenCV's official documentation, but there is a vague reference to **[Snyder]** which is much older than our reference **[Fu-Chang].**

Details about the algorithm Touchwork uses are found in **[Section 4.4.2]**. For OpenCV documentation, refer to **[OpenCV-Doc].**

7.5.2. The Tracking Module

One of the main performance gains in Touchwork among other solutions is the tracking algorithm used in the Tracking Module, which does not use image-based techniques. The following table shows the average time (in milliseconds) required by each solution to track a set of blobs in both OpenCV and Touchwork.

Blob Count	**OpenCV**	**Touchwork**
1	62.63	3.22344
2	78.9866	3.9586
3	98.0292	4.08823
4	99.414	3.51916
8	100.171	4.99561
11	231.4072	3.67952
16	546.482	5.8031

Table 12 - Results of comparing the tracking in OpenCV and Touchwork

Details about the algorithm Touchwork uses are found in **[Section 5.4]**. For OpenCV documentation, refer to **[OpenCV-Doc].**

Chapter 8: Demos

8.1. NASA World Wind

8.1.1. Introduction

World Wind is a virtual globe application (developed by *NASA Ames Research Center*) as an Open Source application for use on personal computers running Microsoft Windows. The program overlays NASA and USGS satellite imagery, aerial photography, topographic maps and publicly available GIS data on 3D models of the Earth and other planets.

Figure 79 - Screenshot of NASA World Wind

A **virtual globe** is a 3D software model or representation of the Earth or another world. It provides the user with the ability to freely move around in the virtual environment by changing the viewing angle and position. Compared to a conventional globe, virtual globes have the additional capability of representing many different views on the surface of the Earth. These views may be of geographical features, man-made features such as roads and buildings or abstract representations of demographic quantities such as population.

8.1.2. System Requirements

- Windows 2000 or XP
- Intel Pentium 3, 1 GHz, or AMD Athlon or higher.
- 256 MB of RAM at least
- 3D Graphics Card
- DSL/Cable connection or faster
- 2 GB of free disk space

Note: World Wind requires that you have .NET runtime 2.0 environment and DirectX 9.0c installed.

8.1.3. World Wind Workspace

World Wind consists mainly of a 3D model (of Earth, moon, other planets or any 3D model), where a camera, moves, rotates, zooms or tilts the 3D model.

8.1.4. World Wind Architecture

World Wind internally uses C# language of Microsoft .NET Framework 2.0, which makes World Wind works only on Microsoft Windows platforms containing .NET runtime and DirectX installed on it.

We are concerned with three main modules in World Wind:

1. **World Window:** This is the interface between the camera along with its movements and the input.
2. **Plug-in system:** It provides the user with a World Window instances to help the user gain control on it.
3. **TouchworkNet:** It is a DLL that we injected inside World Wind to allow it to interact with the Touchwork Framework so it can respond to Touch Events (TouchDown, TouchMove, and TouchUp).

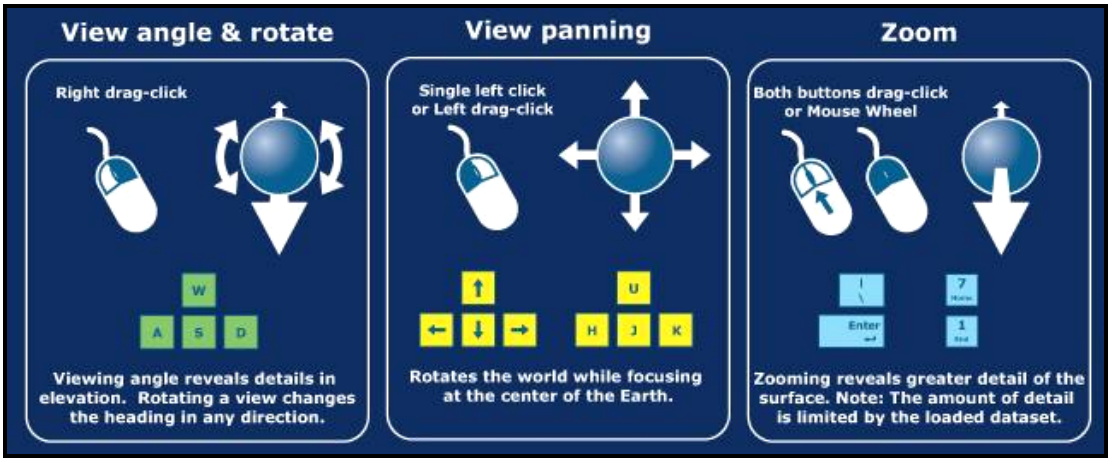


Figure 80 - World Wind basic movements

Basic movements in World Wind and their gesture mapping:

Basic movement	Normal Input	Mapped Gesture
View Panning (Movement)	Left clicking on the mouse while dragging	Moving one finger in the direction wanted for movement
Zoom	Rotating the mouse wheel up and down, or by NumPad keys 7 and 1	Moving two fingers away from each other (Zoom in), or moving two fingers towards each other (Zoom out)
Rotation	Right clicking on the mouse and dragging left and right	Rotating two fingers about their midpoint
Tilting	Right clicking on the mouse and dragging up and down	Fixing two horizontal fingers (y-axis) and moving a third finger up and down

Table 13 - Mapping of basic movements in Workd Wind to gestures

The implementation is fairly discussed in Appendix B.

Chapter 9: Similar Work

In this chapter, we mention other multi-touch solutions; either hardware prototypes or development frameworks, explaining their implementations, features and limitations.

We also mention the main critical points in each solution that we tried to avoid in the development of Touchwork.

9.1. Microsoft Surface

Microsoft Surface™, Microsoft Corporation
http://www.microsoft.com/surface

Microsoft Surface is based on DI Hardware model. **[Section 2.2.2]** In 2003, the team presented the multi-touch table idea to the Microsoft Chairman, Bill Gates, in a group review. Later, the virtual team was expanded and a prototype nicknamed T1 was produced within a month. The prototype was based on an IKEA table with a hole cut in the top and a sheet of architect vellum used as a diffuser. The team also developed some applications, including pinball, a photo browser and a video puzzle. Over the next year, Microsoft built more than 85 early prototypes for Surface. The final hardware design was completed in 2005. **[Wikipedia-MS Surface]**

Figure 81 - Microsoft Surface

Surface was unveiled by Microsoft CEO, Steve Ballmer, on May 30, 2007, at *The Wall Street Journal*'s *"D: All Things Digital"* conference in Carlsbad, California. It is now sold at AT&T retail stores starting from 5,000 to 10,000 U.S. Dollars. However, Microsoft said it expects prices to drop enough to make consumer versions feasible in 2010.

Specifications

Surface is a 30-inch (76 cm) display in a table-like form factor, 22 inches (56 cm) high, 21 inches (53 cm) deep, and 42 inches (107 cm) wide. The Surface tabletop is acrylic, and its interior frame is powder-coated steel. The software platform runs on a custom version of Windows Vista and has wired Ethernet 10/100, wireless 802.11 b/g, and Bluetooth 2.0 connectivity. At Microsoft's MSDN Conference, Bill Gates told developers of "Maximum" setup the Microsoft Surface was going to have:

- Intel Core Quad Xeon "Woodcrest" @ 2.66GHz
- 4GB DDR2-1066 RAM
- 1TB 7200RPM Hard Drive

It has a custom motherboard form factor about the size of two ATX motherboards.

Internal Setup

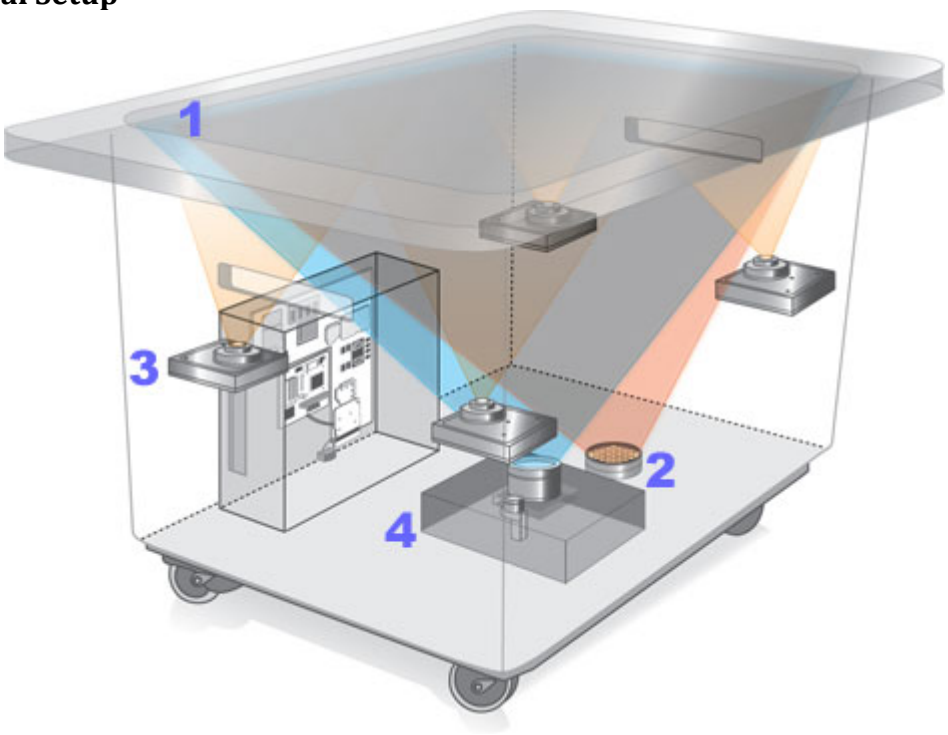


Figure 82 - Microsoft Surface Internal Setup

1. Sanded Acrylic
2. Four infrared-sensitive cameras
3. Infrared Light Diffuser
4. Projector

Features

As mentioned above, Surface uses the DI hardware model, which allows it to detect objects placed on the tabletop, so it can detect a wireless Smartphone as it is placed on the table then display all of its shared contents allowing for drag and drop interaction and communication through wireless capabilities.

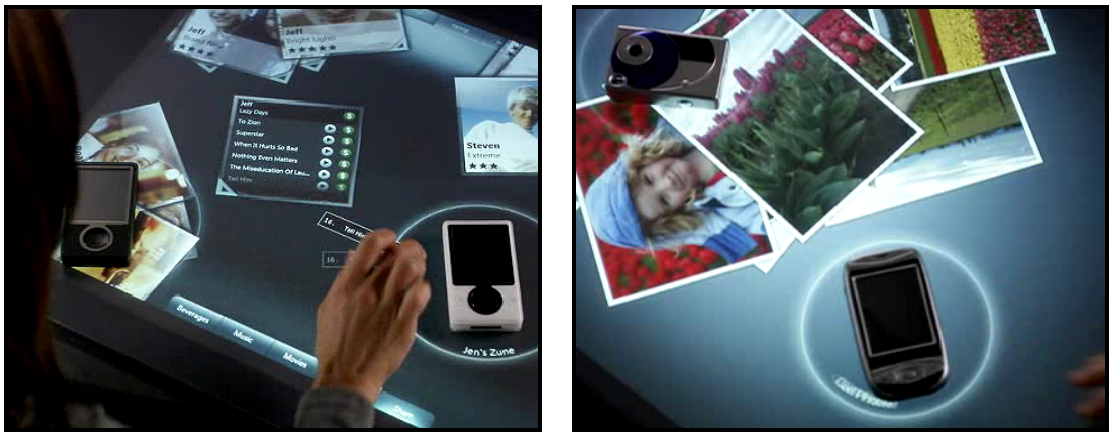


Figure 83 - Surface capability to interact with wireless devices placed on it

Application Development

Microsoft Surface applications can be written in Windows Presentation Foundation or XNA. The development process is much like normal Vista development, but custom WPF controls had to be created by the Surface team due to the unique interface of Surface. Developers already proficient in WPF have been trained in the idiosyncrasies of writing Surface apps and should be available to customize Surface deployments for the large hotels, casinos, and restaurants at which the machines will be next deployed.

A public SDK for Surface was not yet released which we might lower the adoption rate for consumer level and normal users.

Criticism

1. The price is too high compared with other uprising alternatives, which makes its adoption (even by developers) very slow.
2. Supports only WPF at this moment, which constrains the developer to a specific language (C#) and a specific OS (Windows).
3. Game programming and other applications that require C++ will not be supported.
4. There is no way to modify mainstream interactive applications (e.g. 3D Max or Maya) to work with Surface due to the language lock.
   Note: Autodesk (owner of 3D Max and Maya) already started developing its own touch-wall instead of allaying with Microsoft.

9.2. Perceptive Pixel

Perceptive Pixel, Inc., Jeff Han
http://www.perceptivepixel.com/

Perceptive Pixel, Inc. is a company founded by Jefferson Y. Han in 2006, the author of the original FTIR Model paper in 2005, on which our prototype is based. The company designed touch-walls for CIA, CNN and other recognizable clients.

The company is also working on new educational, medical and creative applications. Information about the framework powering their multi-touch walls is not released on public and still proprietary to Perceptive Pixel.

Figure 84 - Jeff Han's multi-touch walls developed at Perceptive Pixel, Inc.

Criticism

Perceptive Pixel targets certain organizations and customers, and doesn't target public domains. There isn't much information about its specification or implementation. Hence, there is nothing to criticize.

9.3. Reactable

Reactable™, reacTIVision©, Music Technology Group,
Pompeu Fabra University – Barcelona
http://reactable.iua.upf.edu

Reactable is a collaborative electronic music instrument with a tabletop tangible multi-touch interface. Several simultaneous performers share complete control over the instrument by moving and rotating physical objects on a luminous round table surface. By moving and relating these objects, representing components of a classic modular synthesizer, users can create complex and dynamic sonic topologies, with generators, filters and modulators, in a kind of tangible modular synthesizer or graspable flow-controlled programming language.

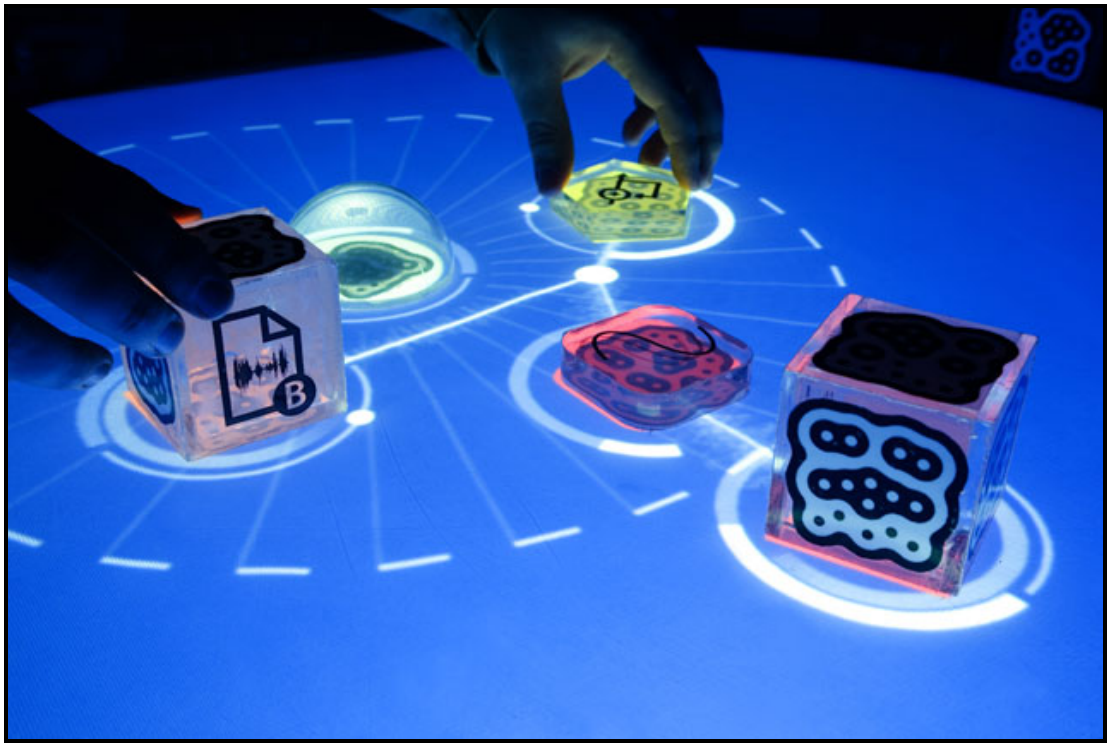


Figure 85 - Reactable Music Table

Specifications

By placing blocks, called ***tangibles***, on the table, and interfacing with the visual display via the tangibles or fingertips, a virtual modular synthesizer is operated, creating music or sound effects.

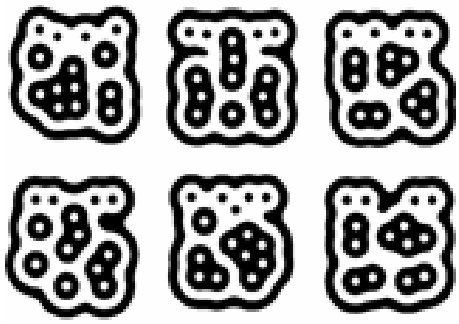


Figure 86 - Reactable Tangibles

There are various types of tangibles representing different modules of an analog synthesizer. Audio frequency VCOs[13], LFOs[14], VCFs[15], and sequencers [16]are some of the commonly-used tangibles. There are also tangibles that affect other modules: the *radar*, which is a periodic trigger, and another one that limits a VCO to the notes of a musical scale.

Internal Setup

Reactable also uses DI in order to support Fiducials.

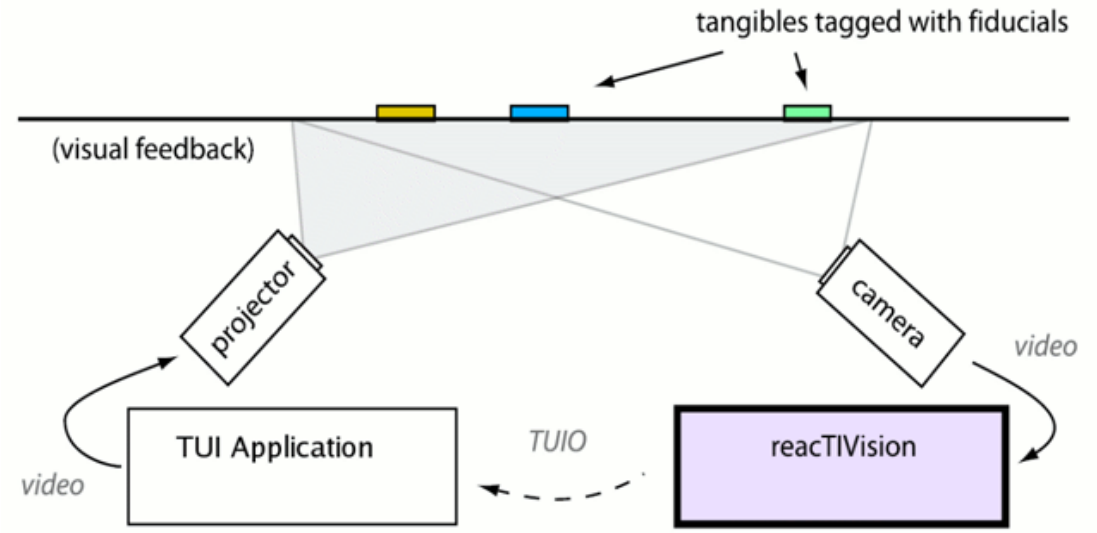


Figure 87 - Reactable Internal Setup

Application Development

Reactable uses reacTIVision framework. **ReacTIVision** is an open source, cross-platform computer vision framework for the fast and robust tracking of fiducial markers attached onto physical objects, as well as for multi-touch finger tracking. It was mainly designed as a toolkit for the rapid development of table-based tangible user interfaces (TUI) and multi-touch interactive surfaces. ReacTIVision is a standalone application, which sends OpenSound Control (OSC) messages via UDP port 3333 to any connected client application. It implements the TUIO protocol, which was specially designed for transmitting the state of tangible objects and

[13] **Voltage-Controlled Oscillator (VCO)** is an electronic oscillator designed to be controlled in oscillation frequency by a voltage input. The frequency of oscillation is varied by the applied DC voltage

[14] **Low-Frequency Oscillation (LFO)** is an audio signal usually below 20 Hz which creates a pulsating rhythm rather than an audible tone. LFO predominantly refers to an audio technique specifically used in the production of electronic music. The abbreviation is also very often used to refer to low-frequency oscillators themselves, which produce these effects.

[15] **Voltage-Controlled Filter (VCF)** is an electronic filter whose operating characteristics can be controlled by means of a control voltage applied to one or more inputs. By varying the cutoff frequency (the maximum frequency passed by the filter), the synth operator can add or remove some of the overtones to create more interesting and textured sounds.

[16] **Music Sequencer**: (Also **MIDI sequencer** or just **sequencer**) is software or hardware designed to create and manage computer-generated music.

multi-touch events on a table surface. The framework includes a set of free example client projects for various programming languages, which serve as the basis for the development of tangible user interface applications. See Section 6.5.1 for more details about TUIO and OSC.

Criticism

1. Reactable mainly targets developing interactive Music Applications which justifies their usage of TUIO which we think is not the best way to interact with applications.
2. Fiducials number is limited by the size of the tangibles, which means the introduction of other Audio Components will force Reactable to increase the size of the component which is against usability.
3. There is no explanation why Reactable does not display the Virtual Audio Modules and allow the users to drag them using their fingers, instead of using real objects (Fiducials).

9.4. TouchLib and OpenTouch

TouchLib©, David Wallin, Pennsylvania, United States, White Noise Audio©
http://nuigroup.com/touchlib
http://whitenoiseaudio.com

OpenTouch©, Pawel Solyga, NUI Group©
http://opentouch.info

Both were meant to provide a Framework for multi-touch application development, (both FTIR and DI models). TouchLib went further and got more adoption especially because it was based on ReacTIVision TUIO, it also used very popular Libraries as its main components like OpenCV and VideoWrapper for blob detection, tracking and input.
Both libraries are released under Open Source Compliant Licenses (LGPL for OpenTouch and New-BSD for TouchLib), which gives the possibility to use and modify in every circumstance.

Application Development

TouchLib can broadcast events in the TUIO protocol (which uses OSC). This makes TouchLib compatible with several other applications that support this protocol, such as Flash, VVVV and Processing etc. Supporting the TUIO protocol also enables a distributed architecture where one machine can be devoted to detection and tracking and another machine can handle the application.

Criticism

1. TouchLib uses TUIO, which is a UDP based solution. Touchwork provides a different interaction model based on Windows Messages and each has its advantages and disadvantages. **[Section 6.3.1]**
2. The main problem of TouchLib is the use of OpenCV in performing blob tracking, which is criticized by application developers as a performance bottleneck due to its reliance on image based tracking which consumes valuable processing time. This can be justified for Fiducial tracking under DI but it is not justified for simple blob tracking under FTIR.
   On the other hand, Touchwork uses a non-image-based algorithm to track blobs. **[Section 5.4]**
3. TouchLib server and client applications need to obtain network-communication privileges in order to perform interaction under UDP based TUIO. Using Touchwork, any applications can send and receive windows messages by default.
4. In TouchLib, if the administrator chooses to let the TUIO server listen to a different UDP port other than the default 3333, the application must ask the user for the new port which may not be known to a roaming user, a more automated mean must be provided for the applications to detect the TUIO port.
   Touchwork uses string based methods to obtain the values of the windows messages used in the interaction.

9.5. AudioTouch: A prototype based on TouchLib

AudioTouch, Seth Sandler, Bachelors degree student in "Interdisciplinary Computing and the Arts with an emphasis in Music" at the University of California, San Diego
http://ssandler.wordpress.com

AudioTouch is an interactive multi-touch interface for computer music exploration and collaboration. The interface features audio based applications that make use of multiple users and simultaneous touch inputs on a single multi-touch screen. A natural user interface, where users can interact through gestures and movements, while directly manipulating musical objects, is a key aspect of the design; the goal is to be able to interact with the technology (specifically music based) in a natural way. Users can load, exit, and move musical applications using gestures as well as have a multiple applications operate within the same work environment. The design is built into a table that hides all the hardware necessary for tracking multiple inputs as well as displaying visual feedback on the touch surface.

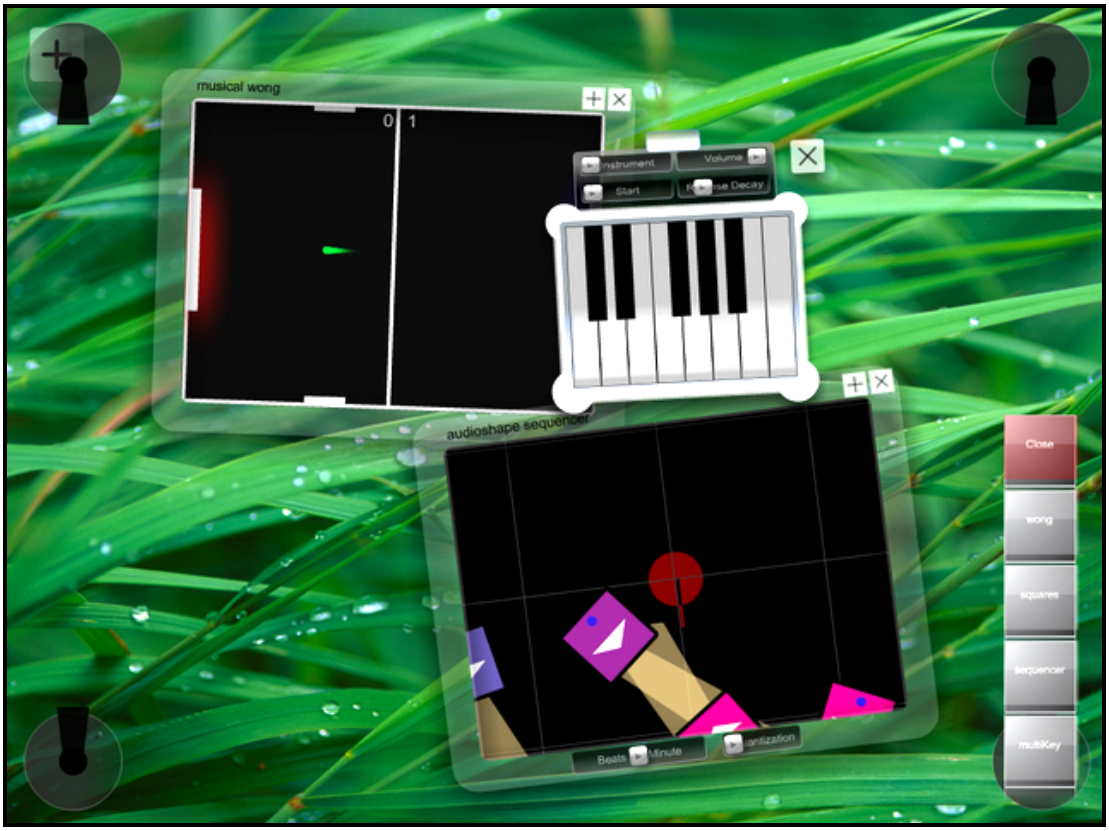


Figure 88 - AudioTouch Music Application

9.6. Tangible Table

Tangible Table™, Daniel Guse and Manuel Hollert, HfG Schwäbisch Gmünd™.
http://www.tangibletable.de

The goal of this project was to build a working prototype of a tangible table-based user interface. In contrast to a simulation, this environment facilitates the evaluation and testing of user interactions. That is why the visual components on the table surface (such as scales) are quite basic and rough. The principles of interaction and graphical behavior had higher priority.

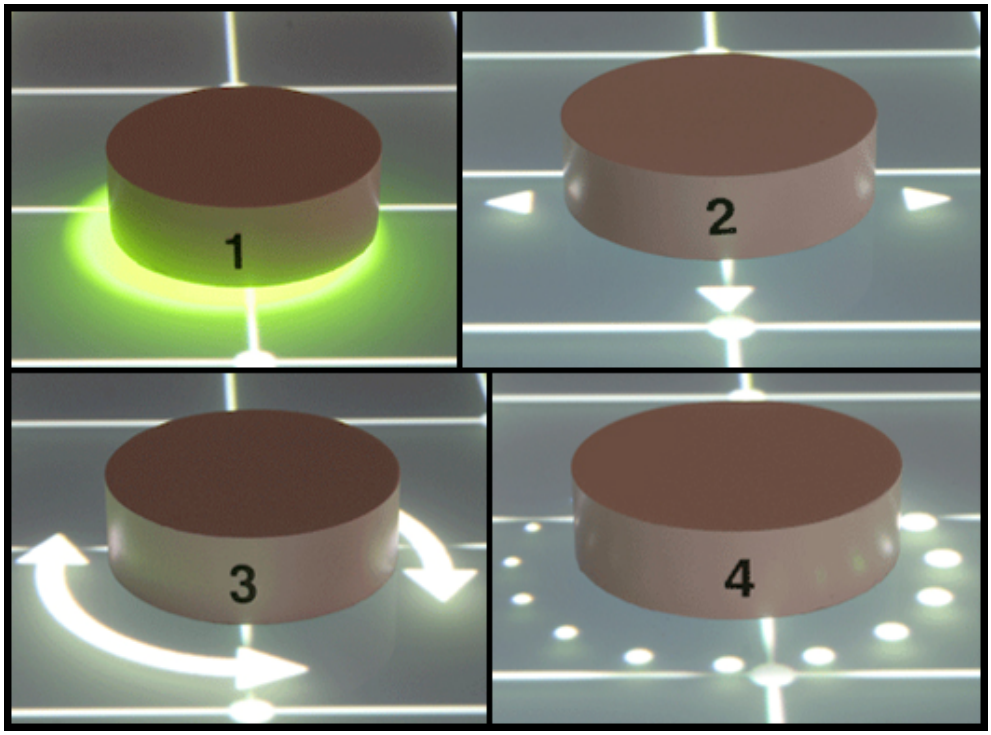


Figure 89 - Tangible Table Controllers

Controllers were created to represent certain properties
(LTR) On, Move, Rotate and Resize Controllers

Internal Setup

Tangible Table prototype is based also on DI model.

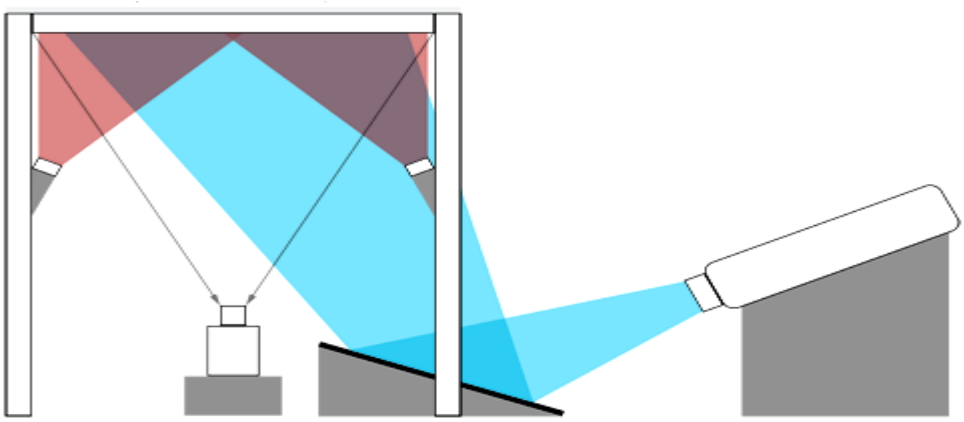


Figure 90 - Tangible Table implements a standard DI setup

The projection of the beamer is reflected by a special mirror (i.e. the reflection occurs straight on the surface of the mirror, there is no extra glass layer above; this is important to avoid duplicated reflections). The rays of light meet the glass surface from underneath.
The project used the same set of Fiducials from the Reactable project along with the same tracking software.

Application Development

Tangible Table was realized in **VVVV**[17], which is a very powerful tool for developing real-time-based multimedia applications such as motion-graphics, sound and video or multiuser environments.

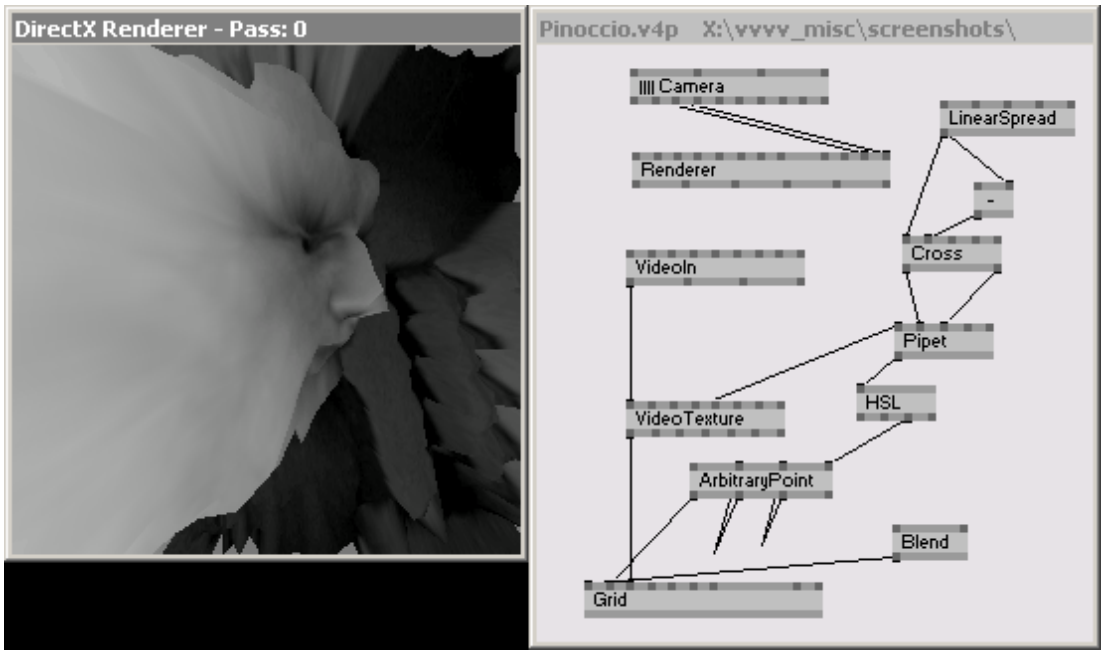


Figure 91 - A visual program in VVVV

The program is created by connecting modules that perform certain functions

Criticism

1. Although VVVV is a very powerful tool, but it has its limitations due to its visual nature which makes it hard to develop complicated interactive applications using it.
2. The use of VVVV as a default development language makes Tangible Table suitable only for experimental purposes or multimedia based applications. That limits Tangible Table to its main purpose "a prototype for exploring Tangible user interfaces" but does not offer much use in other fields.

[17] **VVVV:** A toolkit for real time video synthesis, designed to facilitate the handling of large media environments with physical interfaces, real-time motion graphics, audio and video that can interact with many users simultaneously, through a visual programming interface. VVVV is available for free for non-commercial purposes, available for Windows only. [http://vvvv.org/tiki-index.php]

9.7. Summary

In table 14, Touchwork is compared with other similar work as Microsoft Surface, Perceptive Pixel, Reactable, TouchLib and Tangible Table.

	DI Support	FTIR Support	Consumer	Uses Open Protocols	C#	C++	Fiducial Support	Simulator
Microsoft Surface	✓				✓		✓	
Perceptive Pixel		✓		*		✓		
Reactable	✓			✓		✓	✓	✓
TouchLib	✓		✓	✓	✓	✓	✓	✓
Tangible Table	✓	✓		✓			✓	
Touchwork		✓	✓	✓	✓	✓		✓

Table 14 - Comparison between Touchwork and other similar work

* Not for sale nor custom made, or no information available

Chapter 10: Future Work

10.1. Fiducials and Object Support

Currently the Framework only detects normal finger blobs, but does not support hovering hands or objects placed on top of the screen surface. Due to the modular design of the framework, introduction of a new detector that detects objects, and a tracker that tracks orientation, is easily done provided that a Diffused Illumination hardware prototype is present (which was not the case while developing the current version of Touchwork).

Figure 92 - Object Support

Smartphones and Digital Cameras Support in Microsoft Surface

10.2. Full Gesture Recognition

Currently the framework offers very limited gesture support, a separate module, **twGesture**, should be introduced to allow application developers to define new gestures easily without writing the computational logic for them. Currently, this area is still under heavy research because the different nature of multi-touch gestures from traditional mouse gestures.

Furthermore, a Gesture Definition Language (GDL) could be used for this purpose. Application developers could use this GDL to define new gestures and their properties into a configuration file (usually an XML file), and the generic application logic of the Gesture Module does the processing of the gestures according to this file.

10.3. GPU Utilization

Most of the mainstream home computers and even business workstations are provided with powerful GPU (Graphics Processing Unit) cards that are idle most of the time, unless the user launches an application that performs heavy graphics processing (e.g. Games, Modeling Tools, Video Players).

If present, the framework should use the GPU to perform heavy computations (Fiducial tracking or complex gesture analysis), because the use of a secondary machine to perform these steps will require a very fast connection, which is not available in the consumer market.

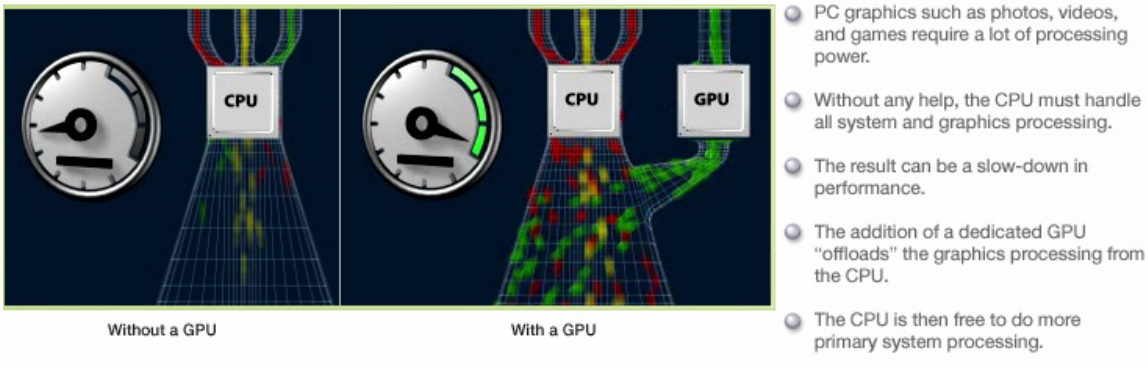


Figure 93 - Performance gain resulting from the usage of a GPU

This field is also still under heavy research, especially because the conflicts that may arise when the GPU is under control by some interactive application using Touchwork itself as a front-end interface. This situation can be solved by the use of SLI™ Technology[18], but it is still arguable that the same interactive application might scale its needs and use the second GPU card. **[Nvidia]**

Nvidia GeForce 9800

MSI X1300 Pro

Figure 94 - Examples of GPU

[18] **Scalable Link Interface** (**SLI**) is a brand name for a multi-GPU solution developed by Nvidia for linking two or more video cards together to produce a single output. SLI is an application of parallel processing for computer graphics, meant to increase the processing power available for graphics.

Part

V

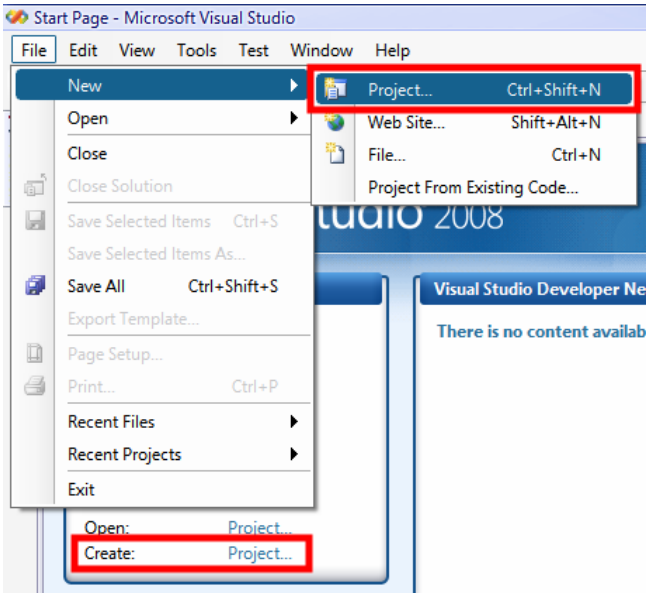
[Appendices]

Appendices

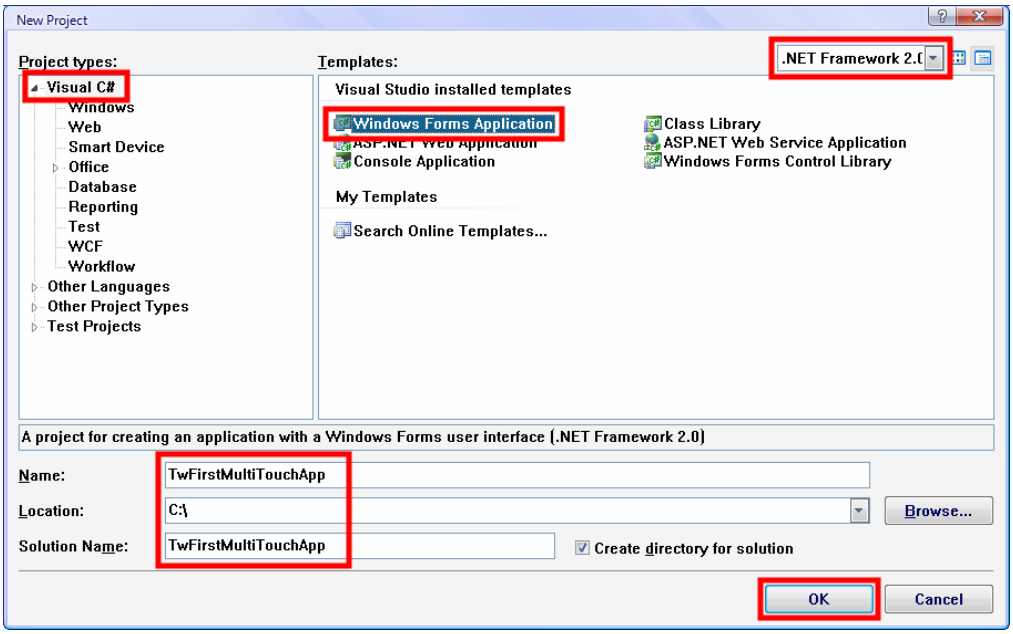
Appendix A: TouchworkNET Tutorial

In this tutorial, we will create a C# Multi-touch Windows Application using Touchwork and Microsoft Visual Studio .NET 2008. Note that the same tutorial works with Microsoft Visual Studio .NET 2005.

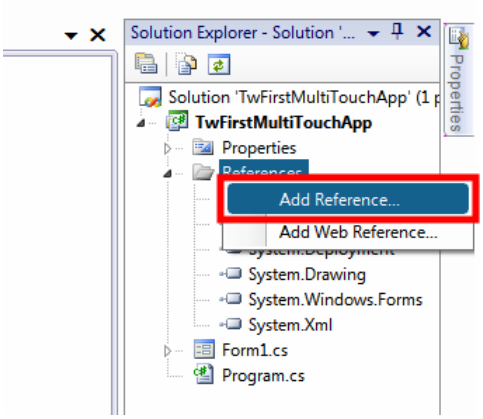
1. Open **Microsoft Visual Studio .NET 2008**. Create a new project from the **Start Page** or by clicking File > New > Project (or pressing Ctrl+Shift+N).



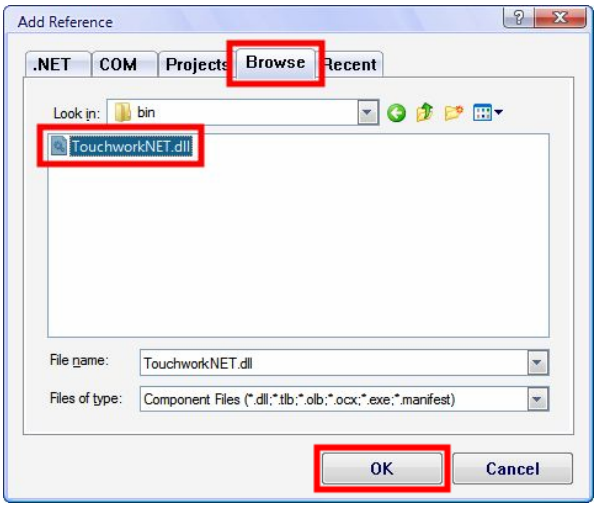
2. The **New Project** dialog pops up. Choose **Visual C#** under **Project Types**, and **Windows Forms Application** under **Templates** as shown below. In this tutorial, the project is set to **.NET Framework 2.0**; this can be changed to any higher version if desired. Make sure you enter a name and location for the project before pressing the **OK** button.



3. In the **Solution Explorer**, right-click the project **References** and click **Add Reference**. If you can't see the Solution Explorer, open it from View > Solution Explorer (or press Ctrl+W, S).

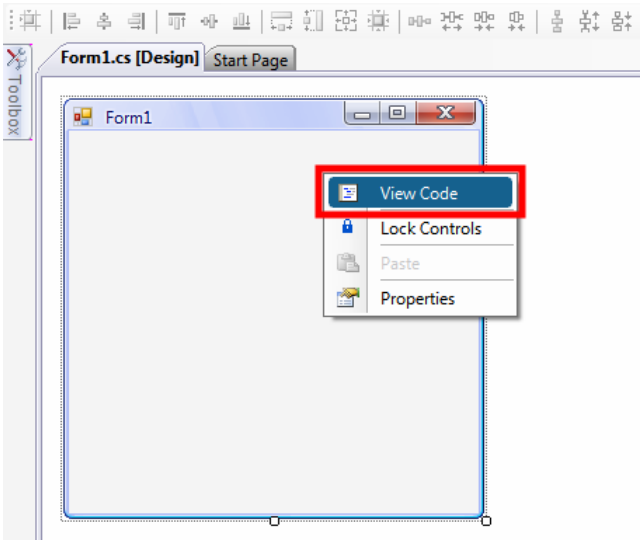


4. In the **Add Reference** dialog, go to the **Browse** tab and navigate to the location of **TouchworkNET.dll** and select it before clicking the **OK** button.

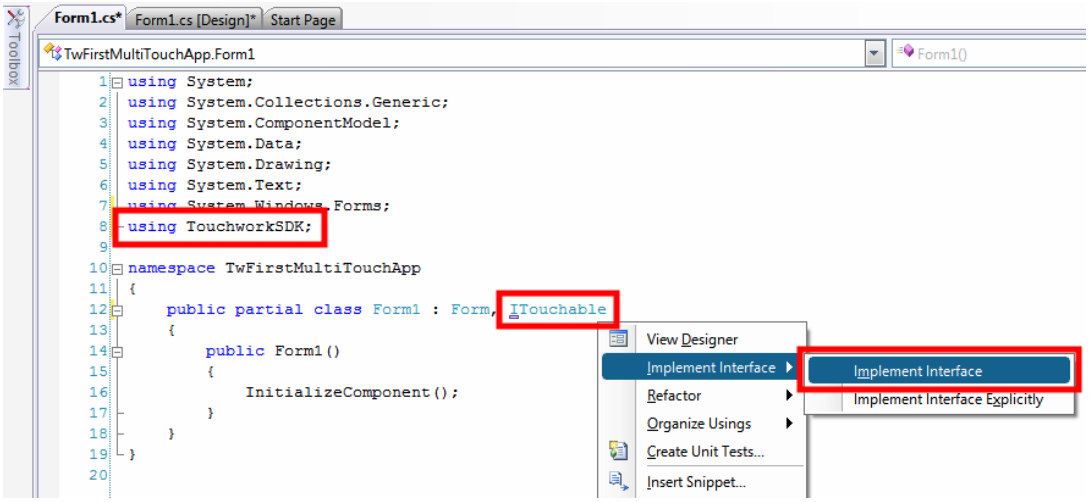


You should see the references added to you project references.

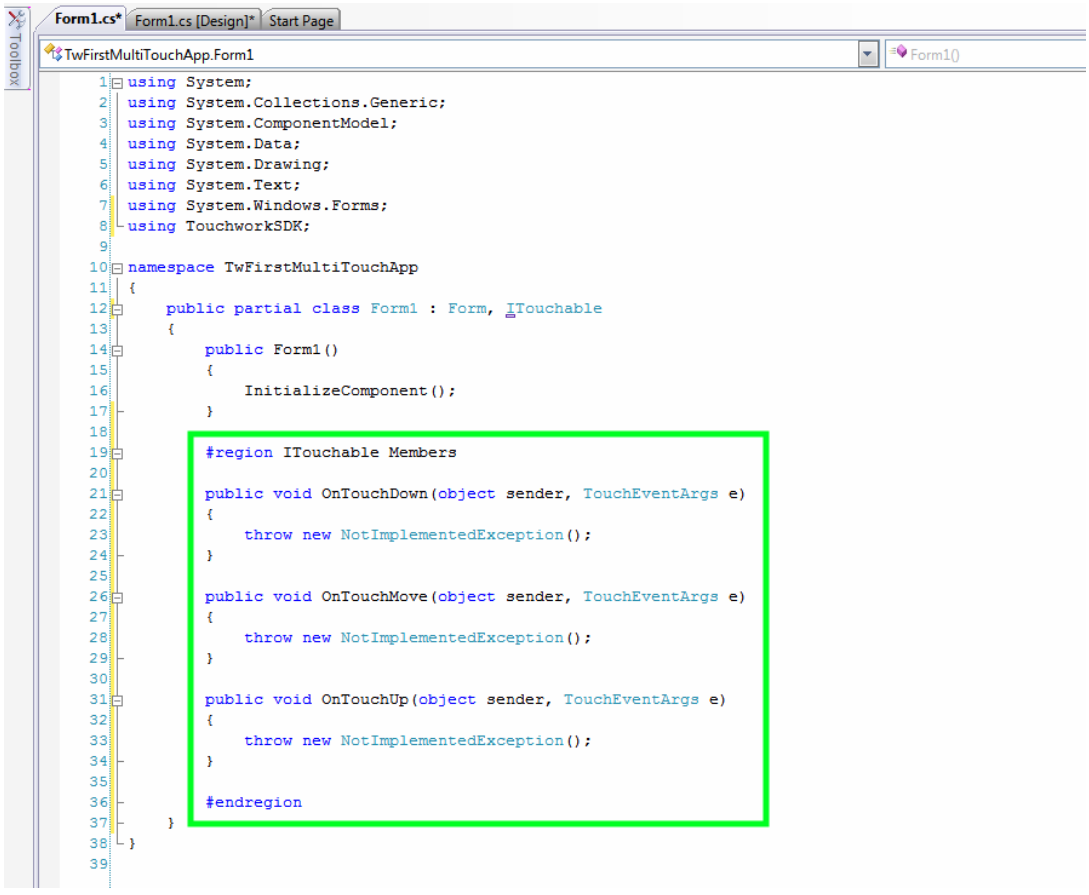
5. Right-click on the form in **Form1.cs** and click **View Code** (or press F7).



6. In the code view, first, add the *using* line to **TouchworkSDK** below your *using*-list to include the reference. Next, let **Form1** implement the **ITouchable** interface as shown below. Then, right-click `ITouchable` and click Implement Interface > Implement Interface (or press Alt+Shift+F10, Enter)



7. By now, you should see a new region, **ITouchable Members**, containing three callback methods: *OnTouchDown*, *OnTouchMove*, and *OnTouchUp*.



8. Write you application logic to handle the *TouchDown*, *TouchMove* and *TouchUp* events into those three methods. Refer to **Section 6.7.2** for more details about TouchworkNET. Also, check the implementation of our NASA World Wind demo in **Appendix B:** *World Wind Implementation* for a more solid example.

Appendix B: World Wind Implementation

These are the procedures made so that World Wind can work with the Touchwork Framework:

1. First, we made a plug-in that used the **TouchworkNET.dll** so as World Wind can receive windows messages from Touchwork, and registered the World Window in the list of windows that the Agent Module sends message to:

```
public class twPlugin: WorldWind.PluginEngine.Plugin
{
      public override void Load()
      {
            Touchwork.NotifyOnTouch(ParentApplication.WorldWindow);
            base.Load();
      }
      public override void Unload()
      {
            base.Unload();
      }
}
```

2. Second, we made World Window implements the `ITouchable` interface , so as we can override the Touch Events:

```
public class WorldWindow : Control, IGlobe , ITouchable
```

3. Third , we overrided the Touch Events (TouchDown, TouchUp, and TouchMove) , and handled the different gestures that will replace the normal input of the mouse and keyboard:

- **TouchDown:** Which adds a new finger to the list of fingers, **mFingers**, and does some processing that will be useful when that finger is moved:

```
public void OnTouchDown(object sender, TouchEventArgs e)
{
      mFingers.Add(e.ID, new Finger(e.ID, e.point));
      if (mFingers.Count == 1)
      {
            MouseEventArgs args = new MouseEventArgs(
               MouseButtons.Left, 1, e.point.X, e.point.Y, 0);
            OnMouseDown(args);
      }
      else if (mFingers.Count == 2)
      {
            int i = 0;
            Finger f1 = null, f2 = null;
            foreach (Finger f in mFingers.Values)
            {
                  if (i == 0)
                        f1 = f;
                  else
                        f2 = f;
                  i++;
            }
            mDistanceBetweenTwoFingers = Finger.GetDistance(
              f1.mPosition,f2.mPosition);
      }
```

```
}
```

- **TouchUp:** It removes the finger from the finger list.

```
public void OnTouchUp(object sender, TouchEventArgs e)
{
      if (mFingers.Count == 1)
      {
            MouseEventArgs args = new MouseEventArgs(
              MouseButtons.Left, 1, e.point.X, e.point.Y, 0);
            OnMouseUp(args);
      }
      mFingers.Remove(e.ID);
}
```

- **TouchMove:** All the gesture recognition processing is done in this function, and applying the corresponding action for every gesture (move, zoom, rotate, and tilt) depending on the number of fingers:
  - **One Finger:** It must a view panning, and the finger will act like a dragging mouse.
  - **Two Fingers:** It will be either a zoom , or a rotation. If the distance between the two fingers increase or decrease we will apply a zoom in/out, and if there is an angle between the old fingers position and the new fingers poistion, then we will rotate with this angle.
  - **Three Fingers:** The only gesture that can happen is the tilting gesture, so we take the first two fingers as the y-axis, and the movement of the third finger tilts camera up or down depending on the direction of its motion.

```
public void OnTouchMove(object sender, TouchEventArgs e)
{
      mFingers[e.ID].Move(e.point);

      if (mFingers.Count == 1)
      {
            MouseEventArgs args = new MouseEventArgs(
               MouseButtons.Left, 1, e.point.X, e.point.Y, 0);
            OnMouseMove(args);
      }
      else if (mFingers.Count == 2)
      {
            int i = 0;
            Finger f1 = null, f2 = null;
            foreach (Finger f in mFingers.Values)
            {
                  if (i == 0)
                        f1 = f;
                  else
                        f2 = f;
                  i++;
            }
            if (prevID1 == -1 && prevID2 == -1)
            {
                  prevID1 = f1.mID;
                  prevID2 = f2.mID;
            }
            if ((f1.mID == prevID1 || f1.mID == prevID2) &&
```

```
            (f2.mID == prevID1 || f2.mID == prevID2))
        {
            //Zooming
            double distance = Finger.GetDistance(f1.mPosition,
               f2.mPosition);
            Point midPoint = Finger.GetMidPoint(f1.mPosition,
               f2.mPosition);

            OnMouseWheel(new MouseEventArgs(MouseButtons.None,
               0, midPoint.X, midPoint.Y, ((int)(distance -
               mDistanceBetweenTwoFingers) * 15)));
            mDistanceBetweenTwoFingers = distance;

            //Rotation
            Finger movedFinger = mFingers[e.ID];
            Finger staticFinger = null ;
            if (f1.mID == e.ID)
                staticFinger = f2;
            else
                staticFinger = f1;

            double angle = GetAngle(staticFinger.mPosition,
               movedFinger.mOldPosition,staticFinger.mPosition,
               movedFinger.mPosition);

            Angle rotate = Angle.FromDegrees(angle);
            this.drawArgs.WorldCamera.Heading += rotate;

            this.drawArgs.WorldCamera.RotationYawPitchRoll(
               Angle.Zero, Angle.Zero, rotate);
        }
        prevID1 = f1.mID;
        prevID2 = f2.mID;
    }
    else if (mFingers.Count == 3)
    {
        int i = 0;
        Finger f1 = null, f2 = null, f3 = null;
        foreach (Finger f in mFingers.Values)
        {
            if (i == 0)
                f1 = f;
            else if (i == 1)
                f2 = f;
            else
                f3 = f;
            i++;
        }
        if (prevTiltID1 == -1 && prevTiltID2 == -1)
        {
            prevTiltID1 = f1.mID;
            prevTiltID2 = f2.mID;
        }
        if ((f1.mID == prevTiltID1|| f1.mID == prevTiltID2) &&
            (f2.mID == prevTiltID1|| f2.mID == prevTiltID2))
        {
            if (f3.mOldPosition != Point.Empty)
            {
                double distance =
                Finger.GetDistance(f3.mPosition,
                  f3.mOldPosition);
```

```
                        if( f3.mPosition.Y > f3.mOldPosition.Y )
                              this.drawArgs.WorldCamera.Tilt +=
                                 Angle.FromDegrees(distance);
                        else
                              this.drawArgs.WorldCamera.Tilt +=
                                 Angle.FromDegrees(-distance);

                  }
            }
      }
}
```

Appendix C: Class Diagrams

This section provides UML [19]2.0 Class diagrams representing Touchwork modular design.

C.1. Touchwork Framework:

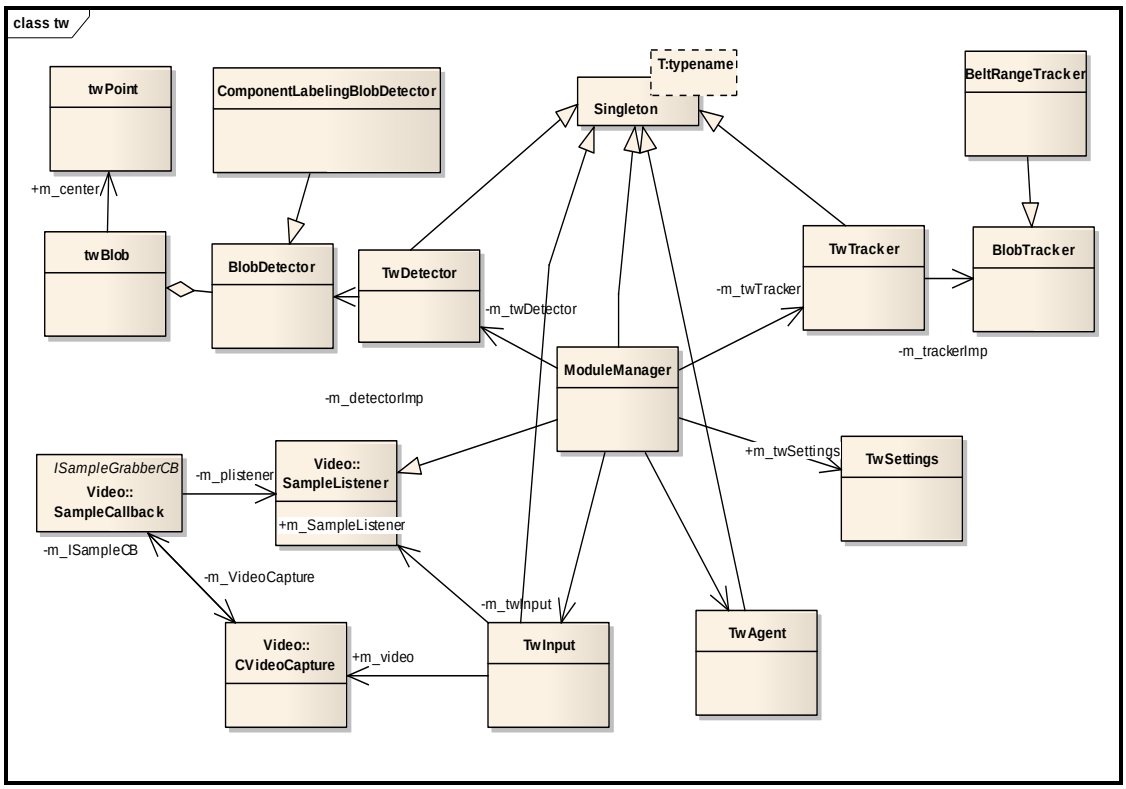


Figure 95 - Class Diagram - Touchwork Framework

ModuleManager class interacts with the Detection Module and the Tracking Module through *TwDetector* and *TwTracker* classes that implement the **Bridge Pattern**; allowing interactions between the *Module manager* and the actual implementations of the detection and tracking algorithms.

A **Bridge Pattern** is used to decouple an abstraction from its implementation so that the two can vary independently. **[GoF]**

BlobDetector is the common interface implemented by the blob detection algorithms, but the Module Manager needs a different interface in order to use these algorithms and yet stays ignorant to their implementation details. This is where *TwDetector* class fits defining a **Bridge** between the detection algorithms and interfaces and the module interfaces.

[19] **UML:** Unified Modeling Language, a standardized visual specification language for object modeling.

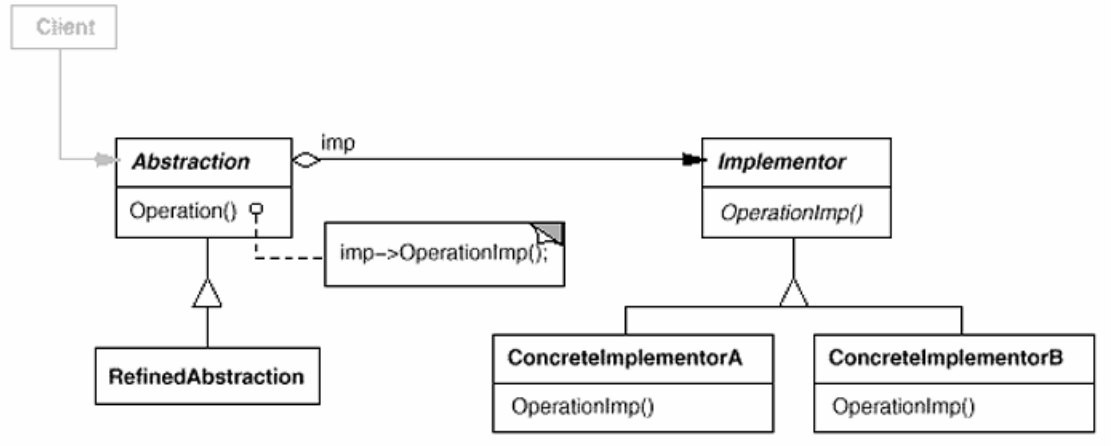


Figure 96 - The Bridge Pattern

In this diagram:

- **Client** *(ModuleManager)*
  This class uses a *ConcreteImplementor* without actually knowing the *Implementor* interface.
- **Abstraction** *(TwDetector, TwTracker)*
  - Defines the *Abstraction*'s interface.
  - Maintains a reference to an object of type *Implementor*.
- **RefinedAbstraction**
  A *RefinedAbstraction* does not exist because Touchwork Modules are not homogenous and does not accept the same input and output neither they perform similar tasks.
- **Implementor** *(BlobTracker, BlobDetector)*
  Defines the interface for implementation classes. This interface doesn't have to correspond exactly to *Abstraction*'s interface; in fact in Touchwork's case the two interfaces are quite different. Typically the *Implementor* interface provides only primitive operations, and *Abstraction* defines higher-level operations based on these primitives.
- **ConcreteImplementor** *(ComponentLabelingDetector, BeltRangeTracker)*
  Implements the *Implementor* interface and defines its concrete implementation. These two classes also follow the **Strategy pattern**.

A **Strategy Pattern** defines a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. **[GoF]** In our case, it allows the framework to use different implementations of the detection and tracking algorithms, and can even replace them at runtime which provides extensibility and modularity for future enhancements.

For the Tracking Module, this pattern is defined between *TwTracker*, *BlobTracker* and *BeltRangeTracker* classes. The same pattern is also defined between *TwDetector*, *BlobDetector* and *ComponentLabelingDetector* classes for the Detection Module.

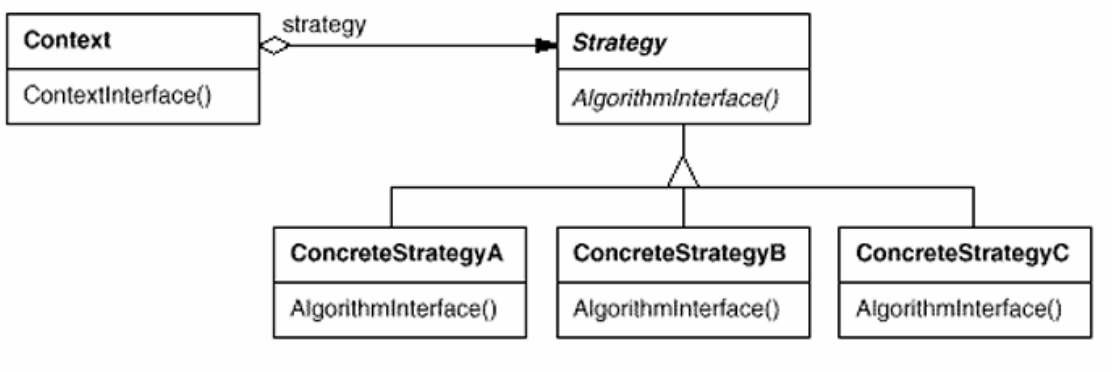


Figure 97 - The Strategy Pattern

This is how Touchwork implements the Strategy pattern:

- **Strategy** *(BlobDetector, BlobTracker)*
  Declares an interface common to all supported algorithms. *Context* uses this interface to call the algorithm defined by a *ConcreteStrategy*.
  - Is configured with a *ConcreteStrategy* object.
  - Maintains a reference to a *Strategy* object.
  - Define an interface that lets *Strategy* access its data.
- **ConcreteStrategy** (*ComponentLabelingDetector, ContourTracingDetector)*
  Implements the algorithm using the *Strategy* interface.
- **Context** *(TwDetector, TwTracker)*
  - Is configured with a *ConcreteStrategy* object.
  - Maintains a reference to a *Strategy* object.

Strategy and *Context* interact to implement the chosen algorithm. A *Context* may pass all data required by the algorithm to the *Strategy* when the algorithm is called. Alternatively, the *Context* can pass itself as an argument to *Strategy* operations. That lets the *Strategy* call back on the *Context* as required.

A *Context* forwards requests from its clients to its strategy. *Clients* usually create and pass a *ConcreteStrategy* object to the *Context*; thereafter, clients interact with the *Context* exclusively. There is often a family of ConcreteStrategy classes for a client to choose from.

C.2. Touchwork Bindings for .NET Framework:

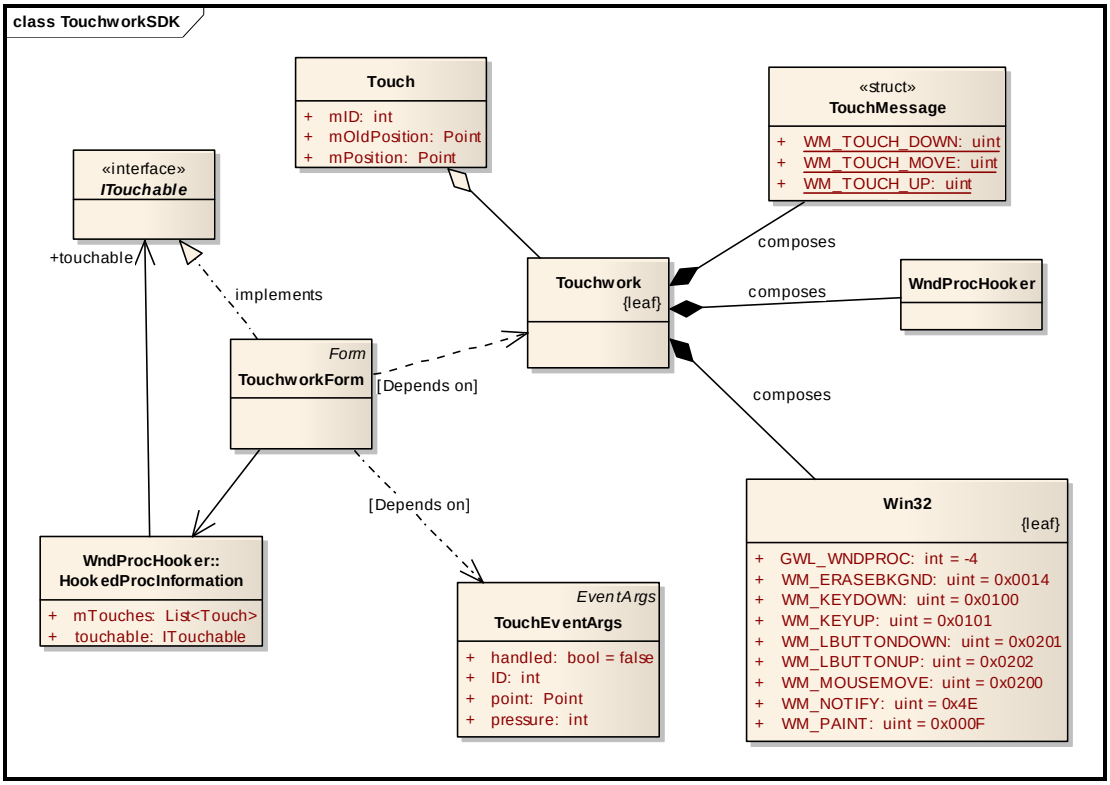


Figure 98 - Class Diagram - C# Bindings in Touchwork

This package implements the **Observer Pattern**, where **TouchworkForm** *subscribes* to touch events by implementing the **ITouchable** interface in order to receive *TouchEventArgs* form Touchwork, the *observer.*

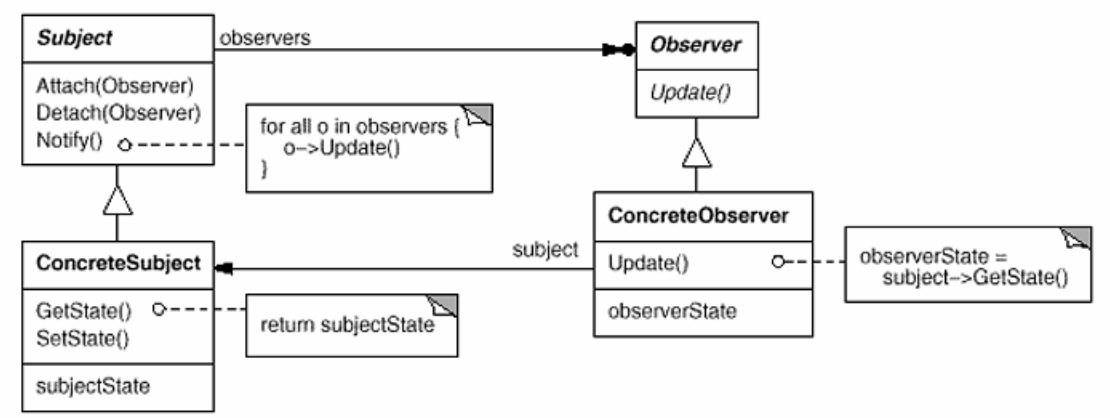


Figure 99 - The Observer Pattern

The **Observer Pattern** (also known as Publisher/Subscribers Pattern) helps to keep the state of cooperating components synchronized. To achieve this it enables one-way propagation of changes: one publisher notifies any number of subscribers about the occurrence of some event. **[GoF]**

This is how Touchwork implements the Observer pattern:

- **Subject** *(TouchEvent)*
  Knows its observers. Any number of Observer objects may observe a subject.
- **Observer** *(TouchworkForm's child controls)*
  Defines an updating interface (*ITouchable*) for objects that should be notified of changes in a subject.
- **ConcreteSubject** *(Touchwork)*
  - Stores state of interest to *ConcreteObserver* objects.
  - Sends a notification to its observers when its state changes.
- **ConcreteObserver** *(TouchworkForm)*
  - Maintains a reference to a *ConcreteSubject* object.
  - Stores state that should stay consistent with the *Subject*'s.
  - Implements the *Observer* updating interface (event handlers) to keep its state consistent with the *Subject*'s.

Appendix D: Use Case Diagrams

Since the system is mainly a development framework, then the main actor interacting with the system is the application developer, this use case mentions the functionalities through which the developer can use the framework.

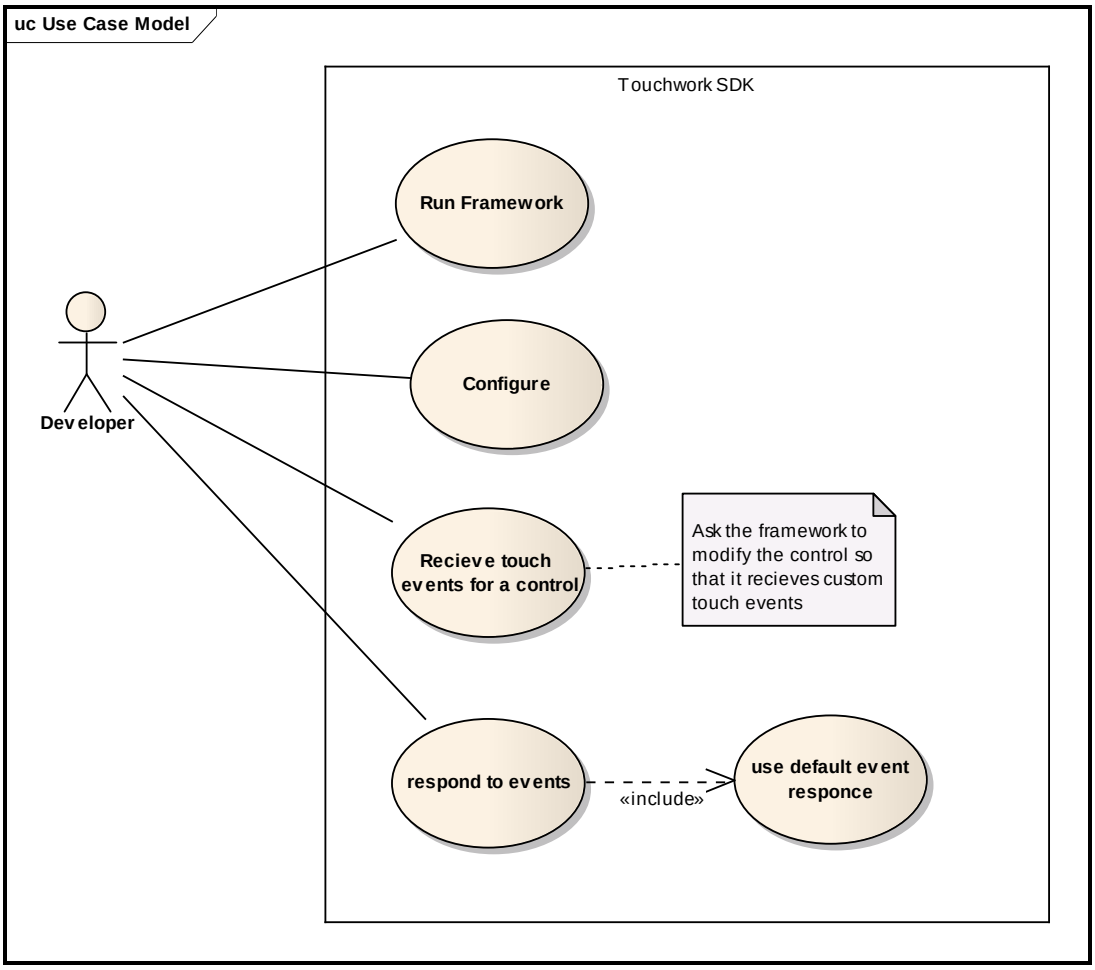


Figure 100 - Use Case Diagram

The Developer Interaction with the System

Appendix E: Activity Diagrams

This section provides a series of diagrams, detailing the internal activities held by each module in the system.

E.1. The Input Module

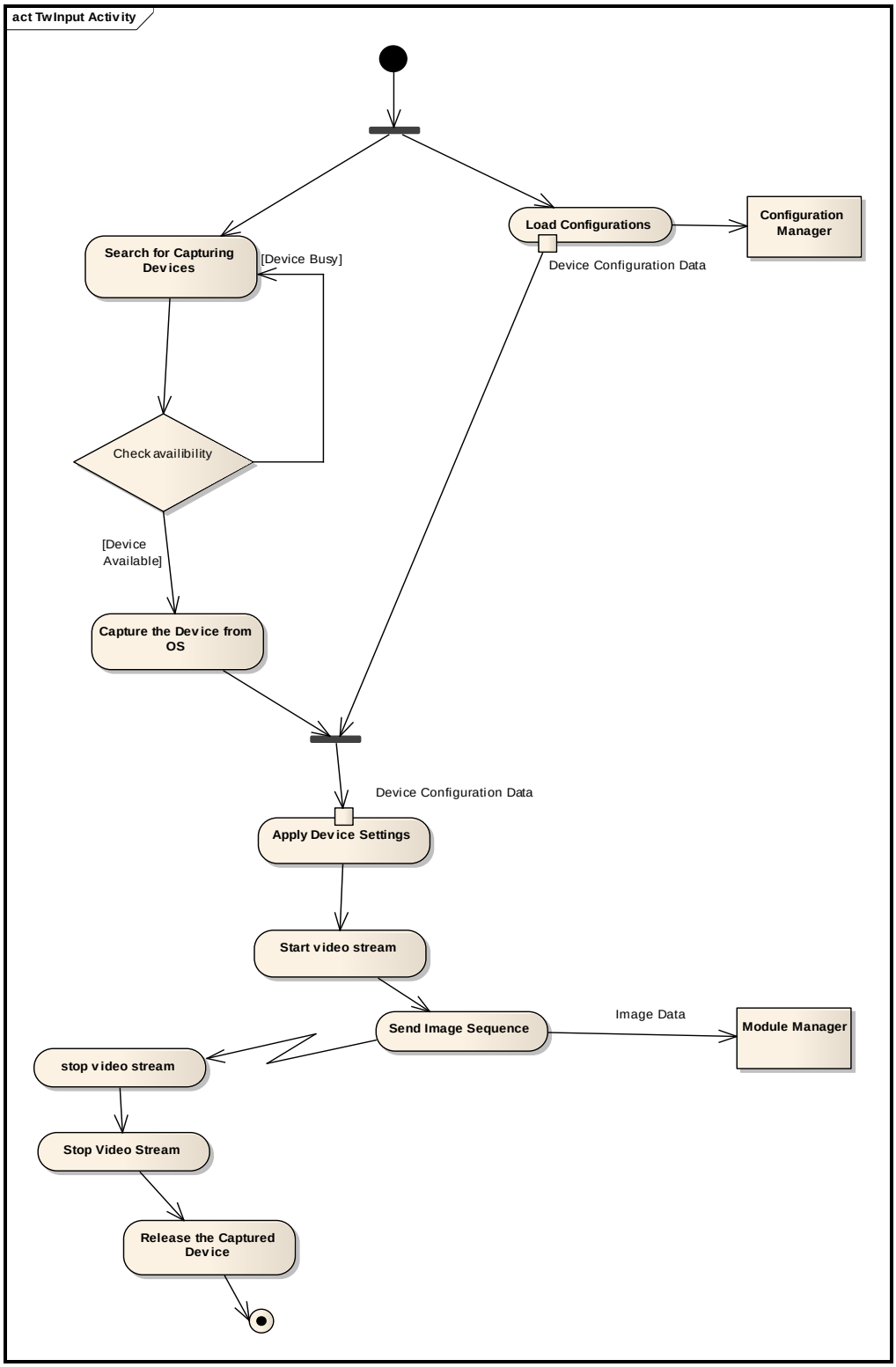


Figure 101 - Activity Diagram - The Input Module

E.2. The Detection Module

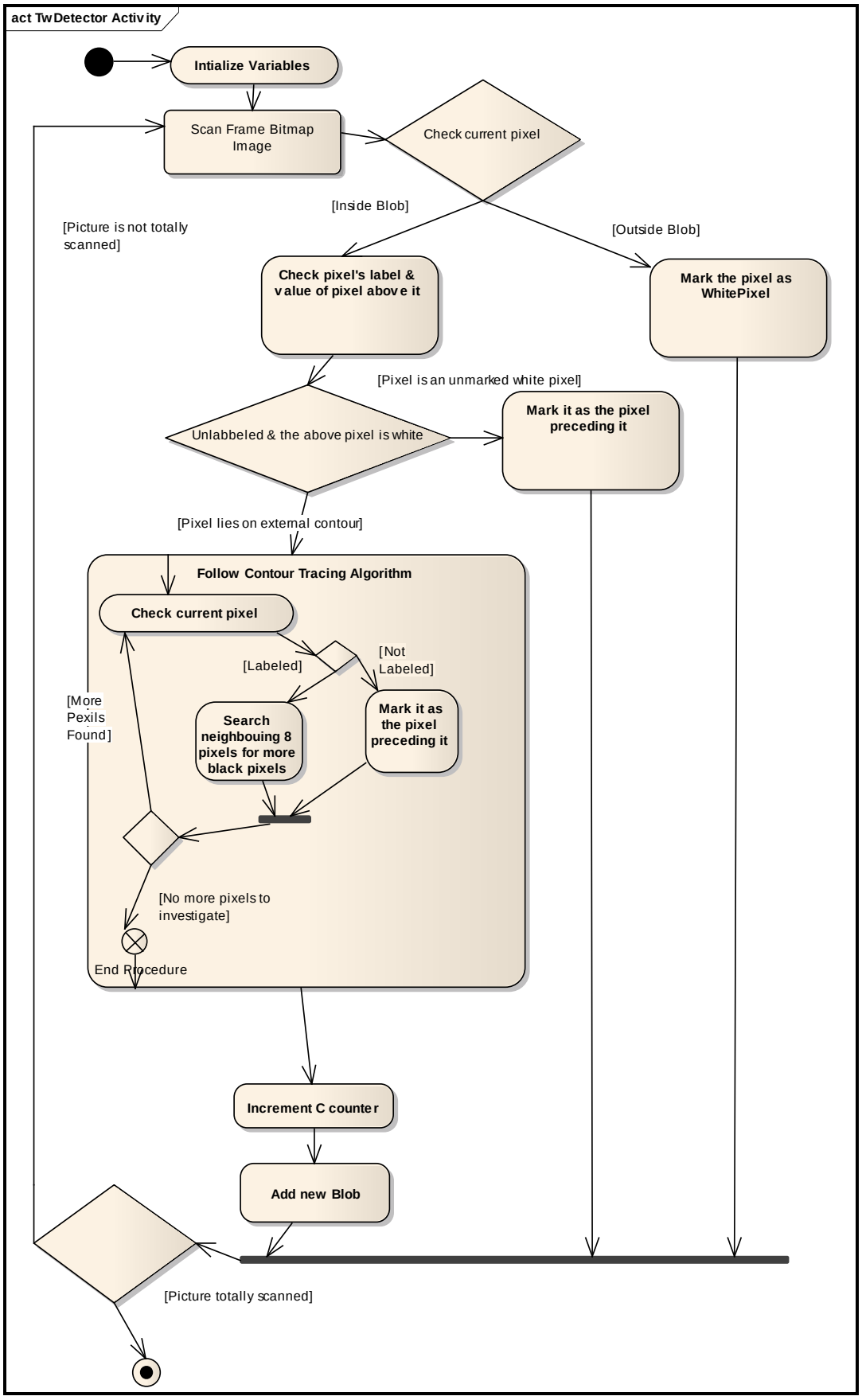


Figure 102 - Activity Diagram - The Detection Module

E.3. The Tracking Module

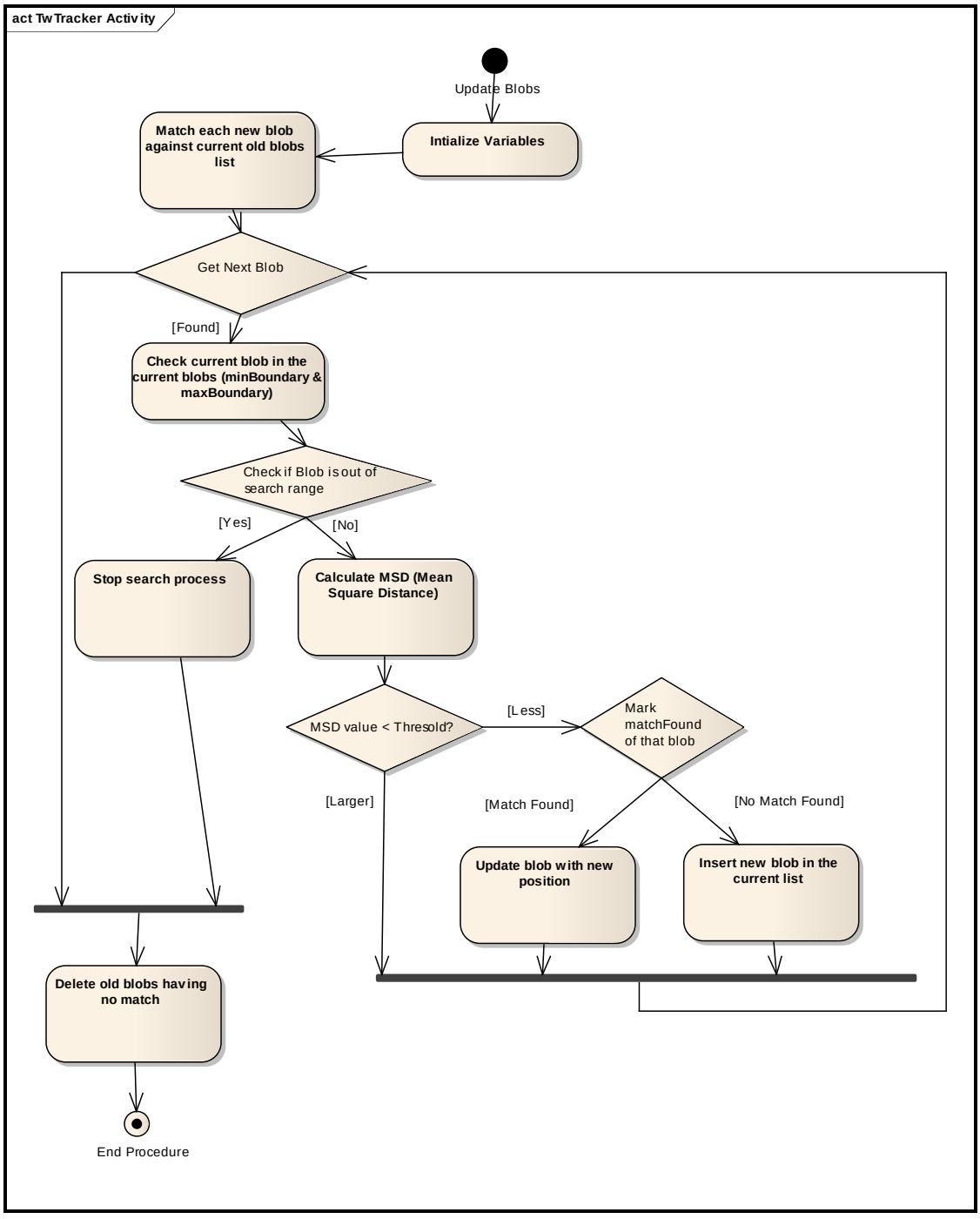


Figure 103 - Activity Diagram - The Tracking Module

E.4. The Agent Module

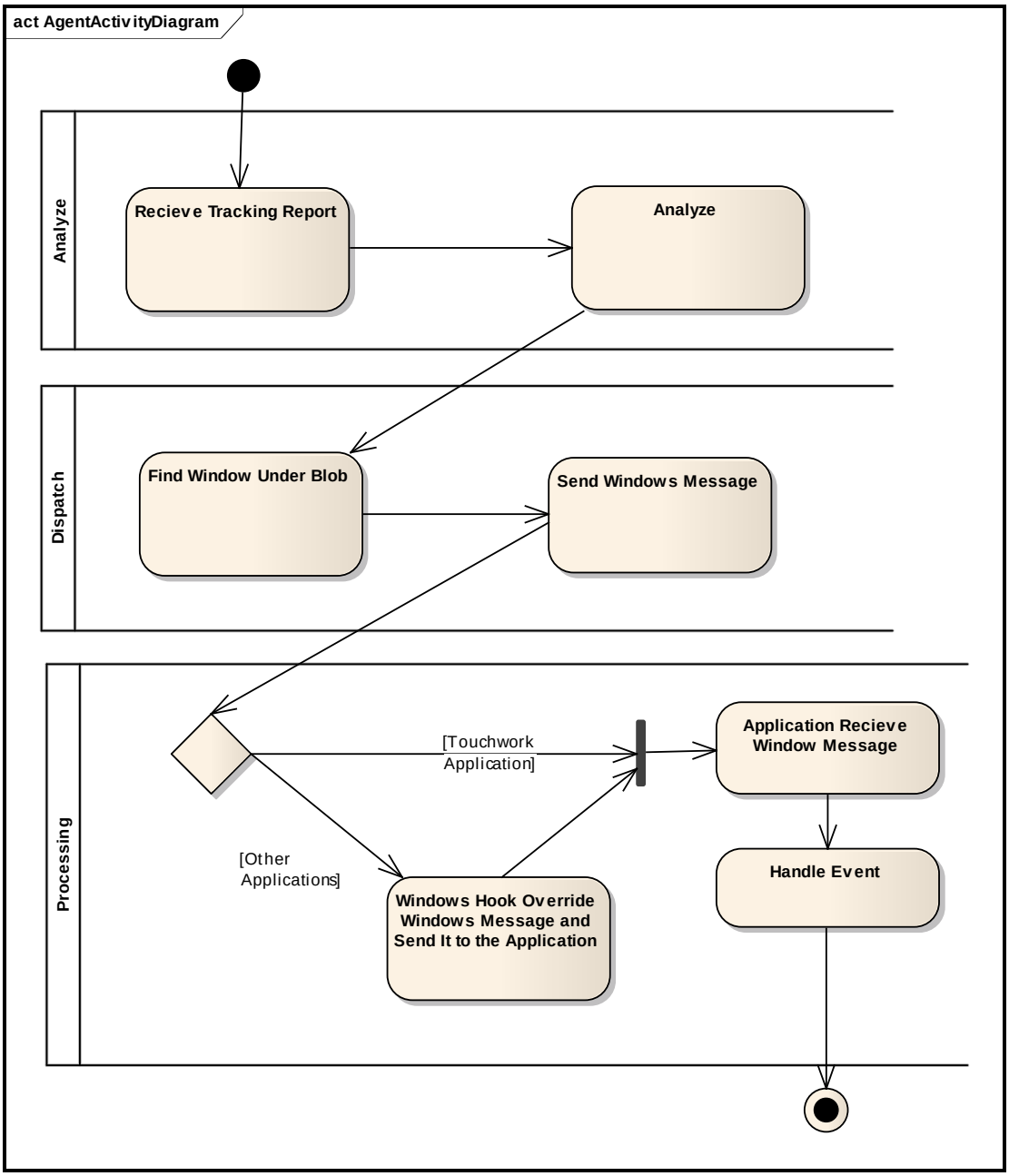


Figure 104 - Activity Diagram - The Agent Module

Appendix F: Sequence Diagram

This section provides details about the sequence of execution of events within the system.

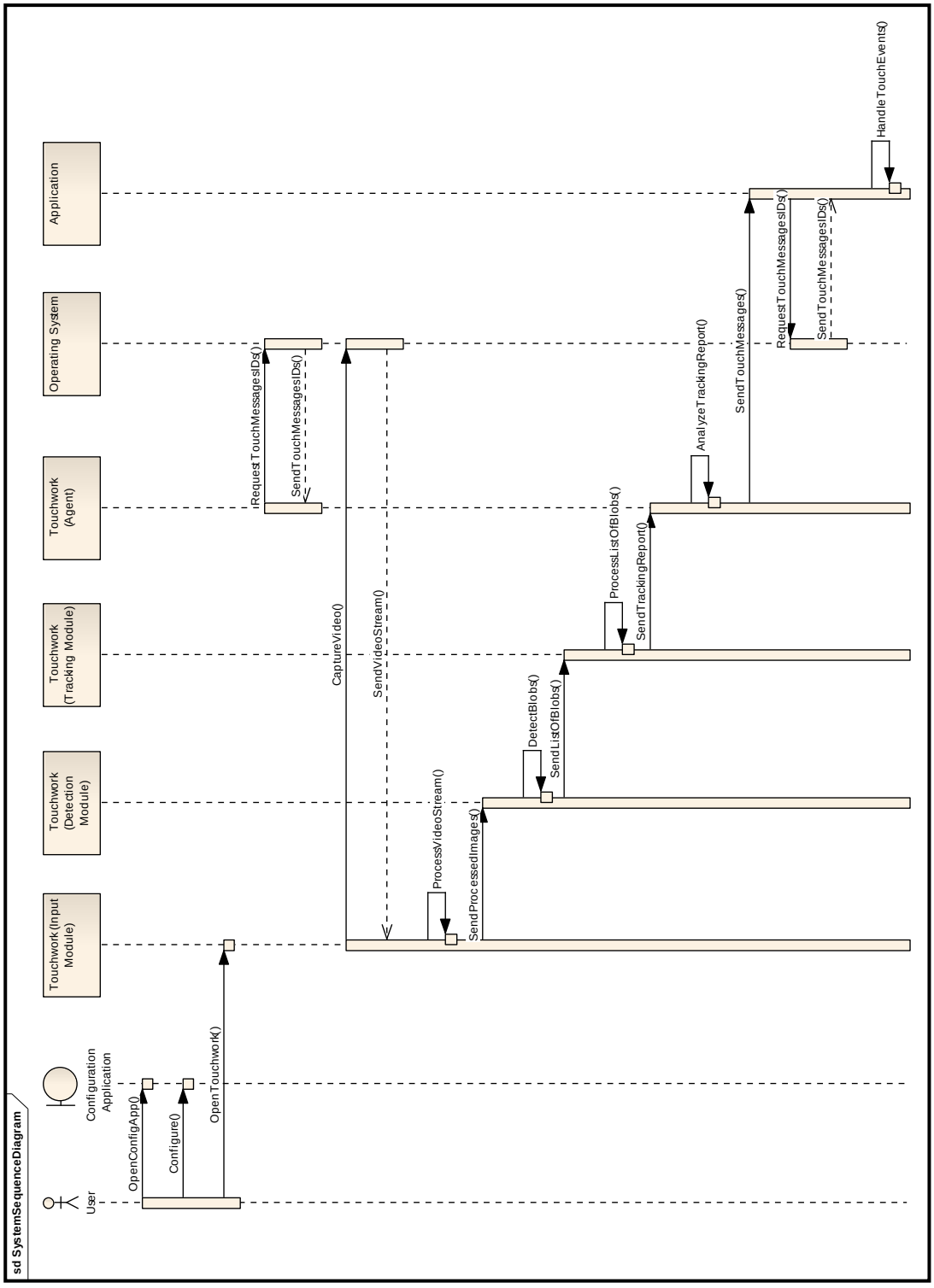


Figure 105 - Sequence Diagram

Appendix G: The Team Members

The team consists of four members, seniors at the Faculty of Computers and Information Sciences, AinShams University. All four members share various knowledge backgrounds, but share the fact that all are ACM Contestants and are members of the acmASCIS Student Chapter Steering Committee.

Alaa Shaker and **Roaa Mohammed** are in Computer Science (CS) department. They study Computer Networks, Image Processing, and Compiler Theory. Alaa has a wide experience in Design Patterns, and Software Design and Architecture. He worked on and studied the architecture and design of a few frameworks in .NET and Java. He also worked in Game Programming for a while. Moreover, Alaa has a great experience in Graphics Design and Web Development. Roaa is a Microsoft Student Partner, which allowed her to gain a lot of experience dealing with Microsoft Technologies. She has a good knowledge of WinAPIs Programming as well. Alaa had internship programs at ITWorx (2007), AXMO (2007) and SmartLabz (2006), while Roaa was at Microsoft (2007) and Shadi Systems (2006). Alaa is the president and Roaa is the vice president of the acmASCIS Student Chapter.

Mahmoud Osama and **Mohamed Fouad** are in Scientific Computing (SC) department. They study Computer Visualization, Computational Geometry, and High Performance Computing. Mahmoud worked for quite a while in Artificial Intelligence in Game Development. He has a good experience in Computer Graphics and DirectX. Mohamed has a great knowledge of the C++ Object Model, WinAPIs Programming, API Interception, and COM/ATL/WTL Libraries. They both worked in graphics solutions using OGRE Rendering Engine, and in web development using PHP. They had an internship program at AXMO (2007) and at a game development company (2007). Mahmoud is the vice president of Seminars and Events and Mohamed is the vice president of IT and Online Community in the acmASCIS Student Chapter.

Alaa Shaker	alaashaker@gmail.com
Roaa Mohammed	roaa.mohammed@student-partners.com
Mahmoud Osama	m.osama28@gmail.com
Mohamed Fouad	m8fouad@gmail.com

Appendix H: References

Journal Articles:

1. **[Baudel] Baudel, T.**, *A Mark-based Interaction Technique for Free-Hand Drawing*. Proceedings of the ACM UIST'94 Conference on User Interface Software and Technology, 185-192
2. **[Fiorio] C.Fiorio and J.Gustedt**, *Two linear time Union-Find strategies for image processing*, Theoretical Computer Science, 1996.
3. **[Han-05] Jefferson Y. Han**, *Low-cost multi-touch sensing through frustrated total internal reflection*, New York University, ACM Press 2005
4. **[Haralick] R.H.Haralick**, *Real Time/Parallel Computing Image Analysis*, Plenum Press, New York, 1981.
5. **[Lumia] R.Lumia, L.Shapiro and O.Zuniga,** *A New Connected Components Algorithm for Virtual Memory Computers, Computer Vision, Graphics, and Image Processing*.
6. **[Moscovich] T. Moscovich, T. Igarashi, J. Rekimoto, K. Fukuchi, J. F. Hughes.**, *A Multi-finger Interface for Performance Animation of Deformable Drawings*, Demonstration at UIST 2005 Symposium on User Interface Software and Technology, Seattle, WA, October 2005
7. **[Rosenfeld] A.Rosenfeld and P.Pfaltz**, *Sequential Operations in Digital Picture Processing*, Journal of the Association for Computing Machinery, 1966
8. **[Shima] Y.Shima, T.Murakami, M.Koga, H.Yashiro and H.Fujisawa**, *A High Speed Algorithm for Propagation-type Labeling based on Block Sorting of Runs in Binary Images*, 1990.

Books and Papers:

1. **[Boer-03] Willem den Boer, Pat Green, Terrance Larsson**, *Active Matrix LCD with Integrated Optical Touch Screen*,
   Planar Systems, Inc., 2003.
2. **[Bradski] Gary R. Bradski**: *Computer Vision Face Tracking For Use in a Perceptual User Interface*,
   Microcomputer Research Lab, Santa Clara, CA, Intel Corporation
3. **[Dudley] Dudley, Brier**, *Leading the Charge on iPhone*,
   The Seattle Times, 2007
4. **[GoF] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides** (Gang of Four):
   Design Patterns Elements of Reusable Object-Oriented Software,
   Addison Wesley, 1995
5. **[Gerpheide] Gerpheide; George E.** : *Methods and Apparatus for Data Input*,
   United States Patents, 1992
6. **[Fu-Chang] Fu Chang**, *A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique,*
   Institute of Information Science, Taipei, Taiwan
7. **[Huelsman] Huelsman, Lawrence**, *Basic Circuit Theory with Digital Computations*,
   Series in computer applications in electrical engineering. Prentice-Hall, ISBN 0-13-057430-9, 1972.
8. **[Matuszczyk] Marek Matuszczyk**, *Liquid crystals in displays*,
   Chalmers University Sweden, ca. 2000.
9. **[MSDN-05]** *Microsoft Developer Network*, Microsoft, 2005 Edition.
10. **[Oney] Walter Oney**, *Microsoft Windows Driver Model,*
    Microsoft Press, 1999
11. **[Petzold-98] Charles Petzold**, *Programming Windows, Fifth Edition,*
    Microsoft Press, 1998
12. **[Rekimoto] Jun Rekimoto**, *SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces*,
    CHI2002, 2002
13. **[Richter-99] Jeffrey Richter**, *Programming Applications for Microsoft Windows, Fourth Edition,*
    MS Press, 1999

14. **[Russinovich-04] Mark E. Russinovich, David A. Solomon**, *Microsoft® Windows® Internals, Fourth Edition*,
MS Press, 2004
15. **[Shi] J. Shi and C. Tomasi,** *Good Features to Track*, 1994
16. **[Stiftung-88] Dr. T. Peter Brody**, *Basic development of TFT liquid crystal display*,
Eduard Rhein Stiftung, 1988 Technology Award

Online:

1. **[Brand]** Thomas M. Brand, *FTIR Multi-touch and Display Device - A Guide to build your own - Experiments with Processing, OSC,*
http://lowres.ch/ftir/
2. **[Han]** Jefferson Y. Han, *Multi-Touch Sensing through Frustrated Total Internal Reflection,*
http://www.cs.nyu.edu/~jhan/ftirsense/index.html
3. **[Hewitt]** Robin Hewitt, *Literature Survey on Tracking and Segmentation Methods,*
http://www.robinhewitt.com/research/track/methods.html
4. **[iPhone-specs]** Apple, *iPhone - High Technology*, 2007,
http://www.apple.com/iphone/technology/specs.html
5. **[NUI-Group]** *Natural User Interface Forums*,
http://nuigroup.com/forums/
6. **[Nvidia]** *NVIDIA© Corporation*,
http://www.nvidia.com
7. **[OpenCV-Doc]** *Blob Extraction Library*, OpenCV Wiki, 2006,
http://opencvlibrary.sourceforge.net/cvBlobsLib
8. **[PSDK]** *Windows® Server 2003 SP1 Platform SDK*, Microsoft Corporation,
http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en
9. **[Roth-08]** Tim Roth, *Zhdk Multitouch Technologies*, 2008,
http://iad.projects.zhdk.ch/multitouch/?p=90
10. **[TED]** *Technology, Entertainment, Design*
http://www.ted.com/
11. **[Wikipedia]** *Open Content Encyclopedia Project*, Wikimedia Foundation,
http://wikipedia.org

Software:

1. **[DX]** *Microsoft Direct X*, Microsoft Corporation,
http://msdn.microsoft.com/en-us/directx/aa937788.aspx
2. **[OpenCV]** *Open Source Computer Vision Library*, Intel Corporation,
http://sourceforge.net/projects/opencvlibrary/
3. **[PSDK]** *Windows® Server 2003 SP1 Platform SDK*, Microsoft Corporation,
http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en
4. **[VVVV]** *Visual, Real-time Programming Toolkit*,
http://vvvv.org/tiki-index.php

Index

N

O

P

R

S

T

U

V

W

Y, Z