EVALUATION OF POINTING STRATEGIES
FOR MICROSOFT KINECT SENSOR DEVICE

FINAL PROJECT REPORT
(Master of Science in Computer Science)

Student: Daria NITESCU

Referees: Denis Lalanne
           Matthias Schwaller

14 February 2012
Abstract

In this final master project, I investigate different strategies for pointing and selecting from a distance using only the human hand, by being tracked by the Microsoft Kinect Sensor Device for Xbox 360. The implemented system provides a hand-free interaction on a gesture based interface. The user interaction on the graphical interface consists in the action of “point and click” on a fixed number of targets.

This report describes the Kinect technology and its applications, as well as the available software development tools. I present different selection strategies that allow the user to perform freehand gestures to accomplish a task. The project introduces two selection strategies, the temporal one and the dart one. The temporal strategy takes into consideration the time parameter, while the dart strategy considers the distance parameter. Then I introduce the visual feedback strategies developed in my project, which are based of geometrical figures and colors, with the goal to show the user how the system recognizes the pointing and selections. In order to achieve the desired user interaction, I developed the system in Linux OS, using the OpenNI framework and NITE middleware, Eclipse platform and C++ programming language. Five performance strategies were implemented considering the time and the distance parameter. Only three strategies were tested with real users with a defined selection of targets stored in the configuration files.

In the end of the project, I provide an evaluation on performance of the implemented pointing strategies using the Fitts’s Law. A detailed statistical evaluation is presented for the three tested strategies, using an ANOVA test. The calculated index of performance shows that the Hand to Kinect Relative Distance (2) strategy has the best performance. The higher the index of performance, the better is the strategy. The ANOVA test confirms that the Hand to Kinect Relative Distance (2) is statistically better than the other two strategies.
# Contents

List of Figures .................................................................................................................. 4  
List of Graphs .................................................................................................................. 5  
List of Tables ................................................................................................................... 5  
Glossary ........................................................................................................................... 6  
1. Introduction ................................................................................................................ 7  
   1.1. Context .............................................................................................................. 7  
   1.2. Motivations and Goals ..................................................................................... 7  
   1.3. Outline of the Report ....................................................................................... 8  
2. Technology .................................................................................................................. 9  
   2.1. Microsoft Kinect Sensor Device for Xbox 360 .................................................. 9  
   2.2. Applications of Kinect Sensor Device ............................................................. 11  
   2.3. Software Development Tools for Microsoft Kinect Sensor Device ............... 13  
   2.4. Summary and Discussion .............................................................................. 22  
3. Selection and Feedback Strategies .......................................................................... 23  
   3.1. Selection Strategies ....................................................................................... 24  
   3.2. Feedback Strategies ....................................................................................... 26  
   3.3. Summary and Discussion .............................................................................. 29  
4. Implementation ......................................................................................................... 30  
   4.1. Temporal Strategy ....................................................................................... 35  
   4.2. Dart Strategy .................................................................................................. 36  
      4.2.1. Hand to Shoulder Absolute Distance Approach .......................................... 37  
      4.2.2. Hand to Kinect Relative Distance Approach .............................................. 38  
      4.2.3. Hand to Kinect Absolute Approach ......................................................... 42  
   4.3. Summary and Discussion .............................................................................. 44  
5. Evaluation .................................................................................................................. 45  
   5.1. Fitts’s Law ..................................................................................................... 45  
   5.2. Design of the Evaluation .............................................................................. 47  
   5.3. Results and Interpretation ............................................................................. 48  
   5.4. Summary and Discussion .............................................................................. 52  
6. Conclusions and Possible Research Directions....................................................... 53  
References ..................................................................................................................... 54  
Appendix A ..................................................................................................................... 55  
Appendix B ..................................................................................................................... 56
List of Figures

2.1. Xbox 360 Kinect Sensor Technology....................................................10
2.2. Abstract Layered View of OpenNI Concept.........................................15
2.3. PrimeSensor Reference Design hardware...........................................19
3.1. Selection Strategies.................................................................24
3.2. Selection Temporal Strategy .......................................................25
3.3. Selection Dart Strategy (Hand to Shoulder) ......................................25
3.4. Selection Dart Strategy (Hand to Kinect) .........................................26
3.5. Design of the user’s hand............................................................27
3.6. Variation of shapes .....................................................................27
4.1. Raw depth data from the Kinect Sensor Device ..................................30
4.2. User calibration .........................................................................31
4.3. User tracking ............................................................................31
4.4. User interaction .........................................................................32
4.5. The Dart Game ...........................................................................36
4.6. The Dart Approach ....................................................................36
4.7. Hand to Shoulder Absolute Distance Approach – graphical representation ......38
4.8. Hand to Kinect Relative Distance Approach (1) – graphical representation ......39
4.9. Hand to Kinect Relative Distance Approach (2) – graphical representation ......40
4.10. Hand to Kinect Absolute Approach – graphical representation ..........43
5.1. The average performance on screen for the three tested strategies.........51
List of Graphs

1. The average IP for the three tested strategies ........................................ 49
2. The performance per target for the three tested strategies ...................... 49

List of Tables

1. The balance conditions to remove bias during experiment ...................... 48
2. Pointing efficiency with three different performance algorithm ............... 50
3. The total number of targets on each area tested for all the three strategies... 50
4. Average user ranking of the three tested strategies ................................. 51
## Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>FAAST</td>
<td>Flexible Action and Articulated Skeleton Toolkit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HKR</td>
<td>Hand to Kinect Relative</td>
</tr>
<tr>
<td>HKA</td>
<td>Hand to Kinect Absolute</td>
</tr>
<tr>
<td>HSA</td>
<td>Hand to Shoulder Absolute</td>
</tr>
<tr>
<td>ID</td>
<td>Index of Difficulty</td>
</tr>
<tr>
<td>IP</td>
<td>Index of Performance</td>
</tr>
<tr>
<td>IR</td>
<td>Infra Red</td>
</tr>
<tr>
<td>MT</td>
<td>Movement Time</td>
</tr>
<tr>
<td>NI</td>
<td>Natural Interaction</td>
</tr>
<tr>
<td>NUI</td>
<td>Natural User Interface</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PC</td>
<td>Production Chain</td>
</tr>
<tr>
<td>PN</td>
<td>Production Node</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>T</td>
<td>Temporal</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>VRPN</td>
<td>Virtual Reality Peripheral Network</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1. Context

There has been always considered a challenge the development of a natural interaction interface, where people interact with technology as they are used to interact with the real world. A hand free interface, based only on human gestures, where no devices are attached to the user, will naturally immerse the user from the real world to the virtual environment.

Microsoft Kinect sensor device brings the long-expected technology to naturally interact with graphical interfaces to the masses. The user interacts having no physical device in his hands or attached on his body. Kinect captures the user’s movements without the need of a controller, but through a Natural User Interface, using just gestures and spoken commands.

1.2. Motivations and Goals

To motivation of my project is to find better ways of selection strategies using Microsoft Kinect Sensor Device. The current selection strategies are not satisfactory; it is a selection method based on a temporal threshold.

The goal of my project is to investigate different strategies for pointing and clicking from a distance using only the human hand, and also to improve the user’s performance and sensation of realism while interacting with a virtual environment.

There has also been taken into consideration the goal of developing natural gestures that the user has to perform while selecting and clicking on a target, the goal of creating hand gestures that are less stressful or tiring in freehand interaction, and the goal of compensating the lack of kinesthetic feedback while interacting with a gestured based interface at distance.
1.3. Outline of the Report

In the first chapter of the report I introduce the context, the motivations and the goals of my project regarding the evaluation of pointing strategies for Microsoft Kinect sensor device.

In the second chapter, I describe Kinect technology and its applications, as well as a review of the available software development tools. I also justify in this chapter why I chose for my project the OpenNI software development tool as framework.

In the third chapter of my report, I introduce different strategies in which a user can do selection using the Kinect device, and in which visual feedback can be provided.

In the fourth chapter a system was implemented to evaluate the performance of different pointing and selection strategies using the Kinect device. Two selection strategies were implemented, the temporal one and the dart one. The temporal strategy takes into consideration the time parameter, while the dart strategy considers the distance parameter.

In the fifth chapter, I present the results of the three tested strategies (Temporal, hand to Shoulder Absolute Distance, and Hand to Kinect Relative Distance (2)).

At the end of the report, in the last chapter, I layout the conclusions and then I propose future work developments.
Chapter 2

Technology

2.1. Microsoft Kinect Sensor Device for Xbox 360

Kinect for Xbox 360 is “a new way to control games through your speech, gestures, and your full body”, as it was declared at the E3 video game conference on June 1, 2009, by Shane Kim, the corporate vice president for strategy and business development at Microsoft’s game division.

Kinect is a camera peripheral by Microsoft for the Xbox 360 video game console. It is a motion control system which captures the user’s movements and translates them into control actions for Xbox 360, without the need of a controller, but through a Natural User Interface (NUI), using just gestures and spoken commands.

Previously known as “Project Natal”, Kinect was first announced on June 1, 2009, at E3. The name Natal means in Latin “to be born” and it was chosen because it reflects Microsoft’s view of the project as "the birth of the next-generation of home entertainment". Afterwards, on June 13, 2010 it was announced that the system would officially be called Kinect, a blend of the words "kinetic" and "connect", which describe key aspects of the initiative. On November 4, 2010, Kinect was launched in North America, while in Europe on November 10, 2010.

Kinect holds the Guinness World Record of being "the fastest selling consumer electronics device", after selling a total of 8 million units in its first 60 days, from 4 November 2010 to 3 January 2011. Gaz Deaves, gaming editor for Guinness World Records, said that, "According to independent research, no other consumer electronics device sold faster within a 60-day time span, which is an incredible achievement considering the strength of the sector".

What it is revolutionary about Kinect is that it’s the world’s first project to combine full-body 3D motion capture, facial and voice recognition, with particular software, all in one device. The actual combination of hardware and software leads to a new way to control and interact. It is no need to hold any peripherals (no buttons, no remotes, and no joysticks); you just need to stand in front of the Kinect device and to use your body and natural movements, like speech and gestures.
The Kinect platform encompasses as technology an RGB camera, 3D depth sensors, a multi-array microphone and a motorized tilt, which are represented below, in Figure 2.1:

![Figure 2.1: Xbox 360 Kinect Sensor Technology](image)

**The RGB camera** delivers the three basic color components, displays the video and helps enable facial recognition. It outputs video at a frame rate of 30 Hz and uses a maximum resolution of $640 \times 480$ pixels, 32-bit color.

**The 3D depth sensor** consists of an infrared laser projector which captures video data in 3D under any lightning conditions. The laser is projected into the room. The sensor is able to detect the information based on what is reflected back at it. Together, the projector and sensor create a depth map. Thus, the 3D depth camera provides detailed 3D information about the environment. Simply said, it determines how far away an object is from the camera. It has a practical ranging limit of 1.2–3.5 m distance when used with the Xbox software.

**The infrared (IR) camera** is used for tracking the movement and the depth. Combined with an IR emitter, the IR camera spotlights the room with invisible infrared light. Thus, the eye does not see the IR light, and the lightening becomes a non-issue for Kinect.

**The multi-array microphone** enables voice recognition to recognize different voices in a room among the different players, and it extracts the ambient noise. The four microphones are located along the bottom of the Kinect and they dictate the size and shape of the sensor device. The microphone array operates with each channel processing 16-bit audio at a sampling rate of 16 kHz.
The motorized tilt is a pivot for sensor adjustment to track the users, even if they move around. It is capable of tilting the sensor up to 27° either up or down, while the angular field of view is of 57° horizontally and 43° vertically.

Kinect is capable of simultaneously tracking up to six people, including two active players and it can track 20 joints per player in real time. However, PrimeSense, which developed the 3D depth sensors, has stated that the number of people the device can "see" (but not process as players) is only limited by how many will fit in the field-of-view of the camera.

The skeleton is generated “using a built-in database of 20 million images with 200 distinct poses”, said Dr. Ilan Spillinger, Vice President of hardware and technology for Microsoft's Interactive Entertainment Business. He also added that Kinect “is able to make reasonable guesses about where all of your body parts are”, but shoulders and long hair can still prevent the generation of the skeleton.

The area required to use Kinect is approximately 6m², although the sensor can maintain tracking through an extended range of near 0.7 m to 6 m. In the Kinect manual it is specified that the sensor can detect the users approximately 2 meters from the sensor. While for two people, the user should stay approximately 2.5 meters from the sensor.

The original intended use of Kinect was to deliver control-free entertainment through the NUI technology. The Kinect sensor device was easy to use, not only for normal users, but also for children with different disabilities or suffering from autism, cerebral palsy or hydrocephalus. For example, a user standing in a wheelchair will immediately be detected as a shorter person.¹

Afterwards, researchers and academic communities started to exploit interesting new uses of the Kinect, not only for fun and entertainment. Thus, developers created open source drivers that make Kinect to work on other devices.

2.2. Applications of Kinect Sensor Device

Forward, I am going to present several new uses of Kinect that caught my attention and are made by developers and academic researchers.

¹ http://news.softpedia.com/news
One of the new applications of Kinect enables teleconferencing in 3D and it was developed by Oliver Kreylos, a research scientist at University of California, Davis. The low-cost solution that makes 3D teleconferencing a reality caught also the attention of NASA, that shown interest in. Oliver Kreylos improves 3D live videoconferencing just by using the Microsoft Kinect peripheral and some developed software. He demonstrated his 3D teleconferencing idea with a friend of him who was working in a remote location. They appeared together in an animated environment using only the Kinect 3D cameras. Kreylos’s future plans to improve teleconferencing include Kinect to track users' head movements.

Another application of Kinect sensor device is developed by Philipp Robbel of MIT. His project is a combination of iRobot’s Create with a Kinect attached to the top. The goal is to map a room in 3D and to have the robot respond to human gestures. When attached to the robot, the camera allows its host to perceive depth accurately. The iRobot moves around the room while Kinect is detecting the objects in the room. The data is collected by robot, and then it is transferred wirelessly to laptop where a 3D map of the room can be constructed. The robot responds to gestures and voice commands and it detects humans using the Kinect’s sensors. At present, Philipp Robbel is researching interesting future applications in robot control through gesture.

A third interesting application of Kinect is also in the field of mobile robotics and collision avoidance. Patrick Bouffard, UC Berkeley graduate student, mounted a Kinect to a robot helicopter, which is called a quadrotor, to detect the altitude and to avoid obstacles. If the robot detects an object in front of it, then it pauses, otherwise, if the object is removed, the robot continues its autonomous flight through the predefined waypoints. The altitude control is made using only data from Kinect.

The Microsoft Kinect device went beyond than its original intended use. And amazing ideas that before were just a dream, now become reality. Interesting applications were developed, such as: controlling the browser with hand gestures, play a virtual piano by tapping the fingers on an empty desk, a video surveillance system that tracks groups of people even in complete darkness, drawing in 3D in the

---

2 http://www.idav.ucdavis.edu/~okreylos/
3 iRobotCreate is based on the Roomba platform and it is explicitly designed for robotics development. Roomba is an autonomous robotic vacuum cleaner sold by iRobot and it is able to navigate a living space and its obstacles while vacuuming the floor.
4 http://bostinnovation.com/2010/12/03/
5 http://www.ros.org/news/2011/01/
air and then rotating the drawing with a nudge of the hand etc.\textsuperscript{6} But what I really think that is \textbf{an amazing purpose} for Kinect is its usage as a vital machine on doing surgical operations. Researchers at the Institute of Forensic Medicine Virtopsy Project at the University of Bern in Switzerland created an application for surgeons to manipulate imaging techniques, used in radiology to visualize detailed internal structures. Researchers used a Kinect 3D camera and a wireless headset to capture hand motions to direct the imaging. The doctors from the Sunnybrook Health Sciences Centre in Toronto adopted this technology to guide imaging during cancer surgery.\textsuperscript{7} Doctors are using it to view and access radiological images for reference during surgery with simple gestures. And this makes the operation time shorter and it reduces the chance of contamination; as they do not need to leave the operation room to read the data and they do not need to sterilize themselves again, for the operation.

\textbf{In conclusion}, Kinect is a motion-tracking peripheral for the Xbox console that contains a blend of cameras and sensors. It was first designed for fun and social entertainment, but scientists and researchers discovered Kinect as an amazing device that can be used in many different areas. Kinect has enabled to build 3D models of different environments, to make robots respond to human gestures and to avoid collisions, to teleconference in 3D and also to guide imaging during cancer surgery.

\section*{2.3. Software Development Tools for Microsoft Kinect Sensor Device (Kinect Libraries)}

\subsection*{2.3.1. OpenNI Framework}

OpenNI (Open Natural Interaction) is an open source framework that defines an API for writing applications using natural interfaces\textsuperscript{8}. OpenNI APIs are composed of a set of interfaces for writing NI applications to be implemented by the sensor devices and by the middleware components\textsuperscript{9}.

\begin{thebibliography}{9}
\bibitem{nytimes}http://www.nytimes.com/2010/11/22/technology/22hack.html?_r=1
\bibitem{qj}http://www.qj.net/qjnet/xbox-360/medical-practice-finds-use-for-kinect-hack.html
\bibitem{openni}http://openni.org
\bibitem{openni_doc}http://openni.org/documentation
\end{thebibliography}
Currently, the available interfaces for OpenNI are only for C and C++, while the available platforms are Windows XP for 32-bit only, and Linux Ubuntu 10.10 and later, for x86.

OpenNI API enables communication with both **physical devices (low-level)**, which are the vision and audio sensors, and **middleware applications (high-level)**, which are the software components. The sensors and the software components are used to produce and process the sensory data.

**The vision and audio sensors** are the devices that “see” and “hear” the figures and their surroundings. **The sensor modules** that are currently supported are: 3D sensor, RGB camera, IR camera, Audio device (a microphone or an array of microphones).

The software components analyze and comprehend the audio and visual data. The middleware components that are currently supported are: full body analysis (generates body related information), hand point analysis (generates the location of a hand point), gesture detection (identifies predefined gestures and alerts the application), and scene analyzer (analyzes the image of the scene, in order to produce the separation between the foreground and the background, the coordinates of the floor plane, and the individual identification of figures in the scene).

There are two important advantages about OpenNI API. One is that it enables applications to be written and ported with no additional effort to operate on top of different middleware modules. And the other advantage is that applications can be written on top of raw data formats regardless of the sensor or middleware providers.

A three layered view of the OpenNI concept is displayed in **Figure 2.2**.

![Figure 2.2: Abstract Layered View of OpenNI Concept](http://openni.org/documentation)
The OpenNI Framework is an abstract layer that provides the interface for both physical devices and middleware modules that produce and process the sensory data. On top of the abstract view is the Application Layer which represents the developed software that implements NI applications. The middle layer is the OpenNI API providing the interfaces that interact with the low-level and the high-level modules. And the bottom layer is represented by the hardware devices or different sensors that capture visual and audio data. The sensor devices produce a form of raw output data, which is a data that can comprehend, understand and translate the real-life 3D scenes. Usually, this data type is a depth map, where each pixel is calculated by its distance from the sensor. A dedicated middleware is used to process this raw output, which can be then used by the application.

The fundamental elements of the OpenNI interface are the Production Nodes (PN). The PN’s are defined as a set of components that have a productive role in the process of creating the data (data generation) for Natural Interaction based applications. An example of PN is the User Generator because it produces body data, but also the Depth Generator is a PN because it produces a depth map, taking raw sensory data from the depth sensor as a stream of X frames per second. Each PN encapsulates the functionality that relates to the generation of the specific data type and can provide this type of data to any object. The object can be another PN or the application itself. Some PN’s use other PN’s that represent lower level data types to produce higher level data for the application. Examples of higher level output can be:

- the current location and orientation of the joints to identify a figure in the scene;
- the center of the palm or the finger’s tips to locate a user’s hand;
- an alert to the application to identify a hand gesture (for example, waving).

There are three types of PN’s, Sensor-Related PN’s, Middleware-Related PN’s and Recording PN’s.

The currently supported Sensor-Related PN’s are: Device (to enable the device configuration), Depth Generator (to generate a depth-map), Image Generator (to generate colored image-maps), IR Generator (to generate IR image-maps), Audio Generator (to generate an audio stream).

The currently supported Middleware-Related PN’s are: Gestures Alert Generator (generates callbacks to the application when specific gestures are identified), Scene Analyzer (the main output is a labeled depth map, in which each
pixel holds a label that states whether it represents a figure, or it is part of the background), Hand Point Generator (supports hand detection and tracking, and generates callbacks that provide alerts when a hand point is detected, and when a hand point currently being tracked, changes its location), User Generator (generates a representation of a full or partial body in the 3D scene).

The currently supported Recording PN’s are: Recorder (implements data recordings), Player (reads data from a recording and plays it), Codec (used to compress and decompress data in recordings).

The Production Chains (PCs) are optional node sequences which are reliant on each other. This topology offers applications the flexibility to select the specific sensor devices and middleware components with which to produce and process the data. Thus, the modules can be simultaneously registered to a single OpenNI implementation. An example of a PC is the sequence of nodes User Generator and Depth Generator because the User Generator type of PN (created by the application) uses a lower level Depth Generator, to read data from a sensor to produce body data.

To conclude, several notes must be mentioned. First, we have to observe that OpenNI enables the application to define which modules, or production chain, to use. OpenNI interface enumerates all possible production chains according to the registered modules. The application can then choose one of these chains, based on the preference for a specific brand, component, or version, and create it. Secondly, OpenNI enables the application to use a single node, without being aware of the production chain beneath this node. And last but not least, an application can also be non-specific, and request the first enumerated production chain from OpenNI.

2.3.2. Libfreenect Software (OpenKinect Project)

OpenKinect is an open community, which started in early November 2010, of over 2000 members, contributing on free, open source libraries that will enable the Kinect to be used with Windows, Linux and Mac.11

The main goal of OpenKinect community is to develop the libfreenect software which includes all the necessary code to activate, initialize and communicate data

---

11 http://openkinect.org
with the Kinect hardware. The libfreenect project is covered under a dual license, Apache20 and GPL2. As yet, it has interfaces for the following languages / platforms: C, C++, .NET (C#/VB.NET), Java (JNI and JNA), Javascript, Python, C Synchronous Interface, Actionscript, and Lisp. The general installation requirements needed for all platforms are for the driver (libusb-1.0, Cmake) and for the glview sample (OpenGL, glut, pthreads), which are attended by detailed instructions for each platform.

The libfreenect software is the core library for accessing the Microsoft Kinect USB camera. Currently, the library supports access to: RGB and Depth Images, Motors, Accelerometer, and LED. Until now, the Kinect audio core is under development and it is not yet integrated into the project source tree.

OpenKinect uses git on github for source control. Git is a distributed version control system (DVCS). The official repositories to develop a contribution respecting the policies are managed by the project lead and designated maintainers. Before a contribution is accepted into an official repository, it must pass the Contribution Criteria.

Several technical issues could be taken into consideration. One of the technical issues is that libfreenect library does not have a skeleton tracking feature, because libfreenect is actually a low-level driver within OpenKinect, while skeleton tracking is higher-level than drivers. The library can only provide the raw data and then a skeleton-tracking solution can be built. Another technical issue is that RGB and IR data can not be streamed simultaneously, which can only be available as different settings for the same isochronous stream. However, it is possible to stream at the same time, the RGB and the depth data.

Kinect has three USB devices connected through its USB cable, known as Generic USB Hub: Xbox NUI Audio, Xbox NUI Motor, Xbox NUI Camera. The devices do not conform to any standard USB class such as HID, camera, or audio devices. This indicates that using it as a plug-and-play webcam or microphone is probably not possible. The Audio device provides combined audio from Kinect's four microphones. It also provides hardware noise cancellation by subtracting the TV's game audio and accounting for the 3D layout of the room. The Motor device controls the actual motor for tilting/panning the Kinect. It also controls power to the rest of the Kinect devices. After the driver for the Motor is installed / activated, Kinect displays a

---

12 https://github.com/OpenKinect/libfreenect
flashing green light and the other devices connect to the internal USB hub. And the Camera provides both an RGB and a depth map image.

To sum up, the libfreenect library, developed by OpenKinect community, provides the audio, depth and motor raw data for the Microsoft Kinect camera.

2.3.3. PrimeSensor NITE Middleware

PrimeSense, an Israeli startup, is the leader in sensing and recognition solutions, and its product portfolio includes the PrimeSensor Reference Design hardware, a 3D data generation unit and the PrimeSense NITE Middleware.

The PrimeSensor Reference Design (Figure 2.3) is a low-cost, plug-and-play USB device. This solution enables a device to perceive the world in 3D and to translate these perceptions into a synchronized depth image, in the same way that humans do. Basically, the Reference Design generates real time depth, color and audio data of the scene.

![Figure 2.3: PrimeSensor Reference Design hardware](http://spong.com/article/21059/Microsoft-Natal-Based-On-Low-Cost-Plug-and-Play-Usb-Powered-Device)

The 3D data generation unit is used for the 3D sensing technology for Kinect Camera device. It is a motion-control system that lets the players control the interface through full-body gestures.

The NITE Middleware is for developing NI applications and it represents the perception component of the PrimeSensor Reference Design end-to-end solution. Beside the perception component, which represents the brain that comprehends the

---

13 [http://www.primesense.com](http://www.primesense.com)

user interaction with the surroundings, the end-to-end solution also contains a sensor component, which observes the scene with the users and their surroundings.

The NITE Middleware enables NI through Control by Gesture application, which facilitates users to interact with different consumer electronics in a natural and intuitively way, having no device in their hands. But, NITE Middleware, combined with PrimeSensor Reference Design hardware, offers a full-body entertainment platform, called Games for All application. The set of controls implemented by NITE middleware enables the social entertainment developers, and not only, to freely use the depth and image processing software to create NI applications.

In conclusion, I would like to say that PrimeSense, who is the leader in NI and 3D depth sensing solutions, launched and cofounded the non-for-profit organization OpenNI to create the OpenNI standards and to implement the OpenNI framework. The OpenNI framework is developed under an open source license and is designed to work with PrimeSense SDK which provides a fully documented API.

2.3.4. Openni_kinect library (ROS OpenNI Project)

Robot Operating System (ROS) is an open-source robot operating system, providing a software framework for the development of robot applications. ROS provides standard operating system services (hardware abstraction, device drivers, message-passing, package management, and more), but also libraries and tools for obtaining, building, writing, and running code across multiple computers.\(^\text{15}\) ROS is a framework of processes that are grouped into packages that are organized into sets of stacks.

ROS OpenNI is an open-source project focused on the integration of the PrimeSense sensors (especially for Microsoft Kinect sensor device) with ROS.

Kinect and OpenNI libraries for ROS are available in the openni_kinect stack. The ROS drivers are the openni_camera and the openni_tracker. The openni_camera driver for OpenNI, publishes raw depth, RGB, and IR image streams, while the openni_tracker driver broadcasts the OpenNI skeleton frames. The ROS driver is compatible to the the Microsoft Kinect, but also with the PrimeSense devices. The

\(^{15}\) http://www.ros.org
nite package offers the PrimeSense NITE middleware, which provides skeleton tracking, hand-point tracking and gesture recognition.

The openni_kinect stack only works with the ROS Diamondback, which was released on March 2, 2011 and is the third ROS distribution release, after ROS C Turtle. Diamondback contains over 120 ROS stacks, including support for the Microsoft Kinect.

To conclude, openni_kinect package focuses on the integration of Microsoft Kinect with ROS and it is supported on Ubuntu Linux, while other variants such as Fedora and Mac OS X are not yet stable.

2.3.5. FAAST Toolkit

The Flexible Action and Articulated Skeleton Toolkit (FAAST) is built on top of PrimeSensor NITE Middleware and relies upon the software drivers from OpenNI and PrimeSense. FAAST toolkit tracks the user's motion using the PrimeSensor hardware or the Microsoft Kinect sensor.16

FAAST middleware includes the body movements and the motion of 24 different body skeleton joints, and it streams the entire skeleton for the first calibrated user that is currently visible to the sensor.

Among other features, the FAAST engine includes a Virtual Reality Peripheral Network (VRPN) server which is used to send data from a Microsoft Kinect sensor device, including the pose for each joint of a user. The Virtual Reality (VR) applications can use any VRPN client. Another interesting feature is that FAAST can map keyboard inputs with different body postures and gestures, facilitating the use of depth sensors with VR applications and games.

In conclusion, FAAST provides tracking of wholes skeletons and is free to use and distribute for research and noncommercial purposes. However, the current version of FAAST is available for Windows only.

16 http://projects.ict.usc.edu/mxr/faast/
2.3.6. Microsoft Kinect SDK (Beta)

Microsoft Research released the non-commercial **SDK for Kinect sensor device** in spring 2011. This programming toolkit was released for the academic research and enthusiast communities to develop NUI applications.

The Microsoft Kinect SDK, which is in a beta status, includes drivers, for using Kinect sensor devices on a computer that is running Windows 7 OS, with the specification that applications cannot run on a virtual machine. The Kinect SDK also includes rich APIs and device interfaces, together with technical documentation and source code samples for developers.¹⁷

There are several important **hardware requirements** for Microsoft Kinect SDK that are mentioned. One of the requirement is that the computer should have a dual-core, 2.66-GHz or faster processor. Then, another requirement is that the Windows 7 OS should support Microsoft DirectX capabilities and the computer requires 2-GB RAM, but 4-GB RAM is recommended.

Interested researchers should be familiar with the **Visual Studio 2010** development environment and **C#, C++** or **Visual Basic languages**.

This beta SDK provides three important features. The first feature offers access to **raw data streams** from the depth sensor, color camera sensor, and four-element microphone array. The second feature enables **skeletal tracking** to track the skeleton image of maximum two persons who are moving in the Kinect sensor’s field of view. Last but not least, the Microsoft SDK provides advanced **audio capabilities** for a four-element microphone array.

Furthermore, I would like to add that the Microsoft SDK enables skeletal viewing for **standing scenarios only**, not seated figures, and the figures should stand between 4 and 11 feet from the sensor.

The **NUI API**, as component of Beta SDK and the core of the Kinect for Windows API, is a set of APIs that retrieves data from the image sensors and controls the Kinect devices. NUI API supports access to the Kinect sensors that are connected to the computer and also to image and depth data streams from the Kinect image sensors. NUI API delivers as well a processed version of image and depth data to support skeletal tracking.

All in all, the long waited Microsoft Kinect SDK generated a lot of research in a short period of time. Even though the beta SDK provides audio capabilities, does not need a calibration pose and supports multiple Kinect sensors on a single computer, it lacks features such as hand only tracking, a gesture recognition system, PrimeSense sensors compatibility and is available only for Windows 7 OS.

2.4. Summary and Discussion

This chapter describes Kinect technology and its applications, as well as a review of the available software development tools. Microsoft Kinect is a motion-tracking device that was first designed to provide control free entertainment through a natural user interface, using just gestured and spoken commands. Scientists and researchers created open source tools that enabled the development of amazing applications in many different areas.

The OpenNI framework defines a device-independent API for writing applications using natural interfaces and offers raw data processing. This library was chosen due to its availability of the documentation, ease of installation and also because it provides, through additional libraries, the ability to track the user skeleton.
Chapter 3

Selection and Feedback Strategies

My project investigates different strategies for pointing and clicking from a distance using only the human hand. The user interacts with a gesture-based interface, having no physical device in his hands, but being tracked by Kinect sensor device at a certain distance and in a certain area.

The selection strategies developed in my project allow the users to perform freehand gestures to accomplish a task, while the feedback strategies compensate the lack of kinesthetic feedback while completing the task.

In order to maximize the task performance, but at the same time to compensate for the limitations of human kinesthetic capacities, I started my investigation by analyzing the standard point and click device, which is the mouse. At a glance, to click on a target while using the mouse, the user has to select the target, perform the action of push and release, and then receive a sensory feedback. Analyzing the human body posture, all these actions take place while the user sits on a chair and he moves his hand in a horizontal plane. In my scenario, the user has to perform free hand 3D space gestures in a vertical plane. I did another investigation on touchscreen technology, and I noticed that the user interacts directly with his finger or a stylus, when he is up close to the screen. In my project, the user interacts with the technology with a bare hand and being away from the display.

Relevant to my work, with the goal to provide a natural transition from the use of one technology to another, are the natural gestures that the user has to perform while selecting and clicking on a target, the hand postures and movements required while having no physical device in hand, the compensation of kinesthetic feedback and the elimination of fatigue in freehand interaction.

Below I present the selection and feedback strategies that are relevant to my project, but also the chosen and implemented strategies.
3.1. Selection Strategies

With regard to my project, I brainstormed different strategies that the user can apply to perform a bare hand click on a graphical interface while he is tracked by the Kinect sensor device.

Therefore, several strategies to accomplish a select & click action could be:
- select the item with one hand and click with the other hand;
- select the item by setting the mouse cursor on the target and then remain with the hand until the click is performed during a certain amount time;
- select the item and then press, push the hand forward, as a depth movement;
- select the item with the hand or voice (saying the word “select”) and then say the word “click” to accomplish the click;
- select and click the item doing different hand gestures (e.g.: doing a fist / turn the hand / using just one or two fingers etc).

Furthermore, other features like select & click, like double-click, right-click or drags, could also be taken into consideration. These features can be accomplished by doing different gestures with your hands, by drawing the shape of the target, or why not voice recognition.

I developed and implemented two different selection strategies in my project to perform a click on the graphical interface through the Kinect sensor device: temporal and dart. The dart strategy has four approaches: Hand to Shoulder Absolute (HSA), Hand to Kinect Relative 1 (HKR1), Hand to Kinect Relative 2 (HKR2), Hand to Kinect Absolute (HKA), and a diagram of the strategies is represented in Figure 3.1:

![Figure 3.1: Selection Strategies](image-url)
3.1.1. Temporal Strategy

In the temporal strategy the user holds the cursor, which is actually the user’s hand, for a certain amount of time in the interior of the target (Figure 3.2).

![Figure 3.2: Selection Temporal Strategy](image)

When the measured time passes a certain value, a click is triggered. The time measured is reset to zero when the hand moves over a certain distance. This is to avoid difficulties in the user holding the hand still and to improve certainty of clicking. Otherwise, a small movement of the hand, such a shaking would reset the timer.

3.1.2. Dart Strategy

In the dart strategy, the user has to keep his hand, the left one or the right one, in the interior of the target, and to move it towards the Kinect to perform a click.

Two main approaches were developed in the dart strategy. In the first approach (Figure 3.3), the distance between the user’s hand and shoulder must exceed a pre-set value in depth \(d\). Once the user pushed in depth sufficiently enough, he has to release the hand.

![Figure 3.3: Selection Dart Strategy (Hand to Shoulder)](image)
The second dart approach (Figure 3.4) is similar to the first approach, but we use the distance between the user’s hand and Kinect itself.

![Figure 3.4: Selection Dart Strategy (Hand to Kinect)](image)

Exploiting all the modules that a Kinect sensor device offers us (raw data streams, skeletal tracking and audio capabilities), I could have implemented all the brainstormed strategies, but the goal of my project is to maximize the task performance. Therefore, basic selection strategies were taken into consideration, exploiting just the raw data streams and the skeletal tracking of the Kinect device.

### 3.2. Feedback Strategies

One of the challenges of my project was to improve the user’s performance and sensation of realism while interacting with a virtual environment. In the system that I designed, the presence and the movements of users are detected and tracked at distance by the Kinect device while they are interacting on the graphical interface.

The most important senses in HCI are the audible, the visual and the tactile senses. They are commonly used and recommended to be used in interface design to create a better, more natural and intuitive interaction.18

Taking into consideration the software and hardware resources available, I decided to implement different visual feedback strategies for my system. To indicate

---

the feedback for the user’s existence in the graphical interface, I used geometrical figures and colors (Figure 3.5).

[Image]

**Figure 3.5:** Design of the user’s hand

As we can see in **Figure 3.5**, the hand is represented in the graphical applications by a hexagon, while the arm is represented by a straight line. The circles that appear on the screen represent the targets that the user has to click on. Moreover, to indicate the movements of the user and the completion of tasks I also used, besides the geometrical figures and colors, the variation of shapes (Figure 3.6).

[Image]

**Figure 3.6:** Variation of shapes

For each of the two interaction strategies that I implemented in my project, I developed different visual feedbacks, as following:

I. In the Temporal Strategy, the size of the hand is set to have a radius in an interval. The hand maintains the maximum size as long as it is in motion. When the user wants to click a target, the hand size steadily decreases until it reaches the minimum value, which signifies a successful click. Moreover, after the click
was performed, the user’s hand changes the color in a dark one and then quickly increases. If the hand is moved before the click to occur, the size changes to the maximum value (Figure 3.7).

![Figure 3.7: Visual Feedback in the Temporal Strategy](image)

II. In the Dart Strategy, the hand’s size decreases while the user’s hand approaches the Kinect device. (Figure 3.8)

![Figure 3.8: Visual Feedback in the Dart Strategy](image)

If the user clicked, the size of the hand is set to a fixed number of units and the color changes, from lighter to darker.

In the design process of the visual feedback strategies, I have taken into consideration the following characteristics: the interface should be intuitive, the gestures should be remembered easily, without hesitation, users can easily see what
the gestures are for, and the interface shouldn’t physically be stressful when used often. The gesture accuracy and the system performance can be improved by adding audio feedback.

3.3. Summary and Discussion

This chapter introduces different strategies in which a user can do selection using the Kinect device, and in which visual feedback can be provided. The selection strategies that were of our interest are two, the temporal one and the dart one. The temporal strategy requires the user to hold still over the target to trigger a click. The dart strategy requires the user to move his hand towards Kinect device to perform a click. The dart strategy can be additionally divided into two approaches, depending on which distances were measured: the hand to shoulder distance or the hand to Kinect distance.

In the design of the visual feedback I used geometrical figures and colors. The hand is represented in the graphical applications by a hexagon, while the arm is represented by a straight line. The circles that appear on the screen represent the targets. The variation of shapes as visual feedback is used to indicate the movements of the user and the completion of tasks. Different colors indicate the clicking state, analogues to a mouse button being pushed and released.
Chapter 4

In this chapter a system was implemented to evaluate the performance of different pointing and selection strategies using the Kinect device. Two selection strategies were implemented, the temporal one and the dart one. The temporal strategy takes into consideration the time parameter, while the dart strategy considers the distance parameter.

Implementation

The “OpenNI_Hand” Project is implemented in Linux OS using the OpenNI framework and NITE middleware. The OpenNI framework offers raw data processing, while NITE middleware provides with skeleton tracking. The project is developed on Eclipse Platform, specifically C++ programming language.

The project provides a GUI which allows the user to click on a fixed number of targets and generates statistics about user interaction efficiency.

At the start of the application, the GUI displays the raw depth data from the kinect sensor device (Figure 4.1).

![Figure 4.1: Raw depth data from the Kinect Sensor Device](image)

Once the user is detected by the application, he is calibrated using a fixed pose (Figure 4.2).
After the calibration is completed, the user is represented on the screen by a wireframe skeleton (Figure 4.3).

The user can now easily track his movements by watching the hand simulations on the screen. The hands are represented in the graphical applications by a hexagon, while the arms are represented by a straight line. The goal is to click on the targets, which are actually the circles that appear on the screen (Figure 4.4). This is accomplished either by hand movement or by using another input device, such as a mouse or touchpad.
The application stops and generates the output statistics once the user successfully clicked on the required number of targets.

**Details of implementation**

The starting point of my project was a sample source code provided by NITE middleware package, called “OpenNI_Hand”. The application consists of the `SceneDrawer.cpp` file which is a collection of methods used to draw the scene, and also the `Main.cpp` file which contains callbacks to initialize the video output and the user input (mouse, keyboard, and movement). Furthermore, it uses the data from the initialization file, provided from the command line, and it initializes the user tracking and calibration.

I have created a class named `User` in the `User.cpp` file to track and update the user movement, generate clickable objects (targets), calculate accuracy statistics and manage user interaction (movement, keyboard or mouse).

Every time the application is stopped, an output file is generated containing useful information regarding the last run.

Besides for the NITE middleware, the other libraries used in the application are the GL and `glut` to draw the user interface and the `config++` library to configure the application parameters using an external file.

In order to achieve the desired user interaction, I have implemented two main strategies as follows: **temporal strategy** and **dart strategy**. The **temporal strategy** is...
designed for the user to perform clicks by holding the cursor, which is actually the user's hand, for a certain amount of time in the interior of the contour of the geometrical figures. In the **dart strategy**, the user has to keep his hand in the interior of the geometrical figure, and to move it towards the Kinect sufficiently enough in depth to perform a click. Once the user pushed in depth sufficiently enough, he has to release the hand.

The temporal and the dart strategies were used to test how the OpenNI library can be used to implement user interaction. They also determine an efficient way to achieve the interaction by measuring Fitts's coefficient and altering the configuration data for the application.

The input data is read from the configuration files for each of the temporal and dart strategies. The common parameters for all the methods, set in the configuration files, are: the `clickMethod` – which chooses one of the implemented methods to click, the `targetSize` – which represents the size of the circle (target), `targets` – which is the array with the coordinates of the targets, and the last common variable is `stopAfter` – which sets the number of the targets that need to be clicked.

For the Temporal Method, I added the parameter `framesToClick` in the configuration file. It represents the number of frames the user has to keep a hand still in order to perform a click. For the Dart Methods, a common parameter is `scale` – which represents the magnification factor. This factor acts in the application as a zoom in. Another common parameter for the Dart Methods is the `distanceToClick` – which represents the variation in depth of movement required in order to initialize a click. It has different meanings for each dart algorithm.

I designed the hand as a structure of the left and the right hand consisting of the current x, y, z coordinates of the hand on the screen (`pos`), the initial position of the hand (`initPos`), the distance between two 3D points (`distance`), the variable `idle` which indicates that the hand is still (in the temporal method) or within clicking region (in the dart method) and the list of floats `pastData` where I store the previous positions of the hand.

```cpp
struct hand {
    XnPoint3D pos;
    XnPoint3D initPos;
    float distance;
    unsigned int idle;
    std::list<float> pastData;
} left, right;
```
I implemented the click in the `click()` function and it is considered as a hit when it is performed in the interior of the geometrical figure, which is actually a circle. Therefore if the distance between the current position of the hand and the coordinates of the center of the target is smaller than the radius of circle, then I consider it as a hit. Otherwise, I consider a miss and I do not take into consideration the misses performed by the hand which is farther than the clicking hand. When it is a hit, I calculate the Index of Performance (IP) in the `clickHit()` method based on the Index of Difficulty (ID) which is implemented in the `fittsLaw()` function. Once the click was a hit, I generate a new target.

The target is generated by the `generateTarget()` method. As long as there are still elements in the `TARGETS` array, the position of the next target is read from the array, otherwise it is randomly generated. The method `getTarget()` returns the position of the current target.

Each time the user moves, the `updateHand()` function is called for each hand. It draws a simulation of the hand and varies the size using the `getHandSize()` function based on the current position and action; stationary or moving, in the process of clicking or not.

For the application I have also implemented the feature to click with the mouse as an alternative for body movements. The purpose is to have the possibility to compare how quickly pointing can be done using different clicking methods. The click performed with the mouse is implemented in the `externClick()` function. If the click occurred in the interior of the geometrical figure, then I call the `clickHit()` function to compute the IP which counts as a right hand click and I change the hand coordinates to the current position (x,y). If the click was done in the exterior of the figure, then I increment the number of misses. The mouse-click is initialized in the `glutMouse()` function in the `Main.cpp` file. The click occurs when a mouse button is pressed and released. The mouse is activated when the first click is performed and it is assigned to user with the 99 ID, to avoid interference with a user tracked by Kinect.

The application stops when the `updateHand()` function detects that all the targets were successfully clicked.
4.1. Temporal Strategy

The temporal strategy is designed for the user to perform clicks by holding the cursor, which is actually the user’s hand, for a certain amount of time in the interior of the contour of the geometrical figures.

The idea of the temporal method is to hold the hand still for a fixed number of frames which counts as a click. In other words, a **successful click** is a click triggered in the interior of the geometrical figure that happens after a fixed number of frames that passed. A **missed click** is a click that is performed in the exterior of the target and the fixed number of frames passed. To improve the temporal method and not to have a big number of misses, I do not count the misses performed by the hand which is farther away than the other hand. In the implementation, farther away means actually a distance that exceeds the distance between hand and target. For example, if the user has both of his hands on the screen, but he wants to perform a click with the right hand on a target that appeared on the right corner, the missed performed with the left hand is not counted. Naturally, the left hand should be on the left side of the screen (and the distance between the left hand and the target is bigger than the distance between the right hand and the target).

If the **idle** counter has not reached the required number of frames and the hand is moved, then I set the counter to zero. Otherwise I increment the counter until it passes the selected threshold.

```plaintext
if (d > FRAMES_CLICK) {
    hand.idle = 0;
}
hand.idle++;
if (hand.idle == FRAMES_CLICK) {
    click(hand, joint.position);
}
```

The **idle** counter is reset to zero when the hand moves over a certain distance (d>FRAMES CLICK). This is to avoid difficulties in the user holding the hand still and to improve certainty of clicking. Otherwise, a small movement of the hand, such a shaking would reset the timer.

In the configuration file for the temporal method I set the coordinates of the targets and the number of frames that count as a click, **framesToClick** parameter.
The size $s$ of the hand that performs a click is set to have the radius between 0 and 15.0. The initial size is 15.0 and it remains the same as long as the hand is moving. When the hand is idle, as the user wants to click a target, the hand size steadily decreases until it reaches value 0, which signifies a successful click. If the hand is moved before the click to occur, the size changes to the initial value of 15.

\[
s = 15.0 \times \frac{\text{FRAMES\_CLICK} - h->\text{idle}}{\text{FRAMES\_CLICK}};
\]

### 4.2. Dart Strategy

Dart is the popular game where darts are thrown by users at a circular target fixed to a wall (Figure 4.5). The dart game involves a specific board design and a set of rules.

![Figure 4.5: The Dart Game](image)

In the dart strategy that I designed (Figure 4.6), the specific board is the graphical interface and the darts are the Kinect Sensor device interactions.

![Figure 4.6: The Dart Strategy](image)

The rule of the dart strategy that I implemented states that once the user clicks on a circle, he has to immediately click on the next circle that appears on the user
interface. In order to finish the "game" the user has to successfully click on the required number of targets. To measure if the user clicked on a target, I used the distance variation between the user and the Kinect Sensor device.

I have implemented three different dart approaches, taking into consideration different distance variation in depth of movement between the user and the Kinect Sensor device. The three approaches are **Hand to Shoulder Absolute Distance (HSA)**, **Hand to Kinect Relative Distance (HKR)** and **Hand to Kinect Absolute (HKA)**.

A common feature for the dart algorithm that I have implemented is the *zoom in* feature in the `project()` function. This function returns the scaled current position of the hand. After the coordinates are received from the Kinect device and transformed into two dimensional coordinates, the function increments their values by the scale factor.

### 4.2.1. Hand to Shoulder Absolute Distance Approach

The idea of the Hand to Shoulder Absolute distance approach is that the user has to move the hand, the left one or the right one, towards the Kinect, sufficiently enough to perform a click. The distance that the user has to cover is set in the configuration file as the `distanceToClick` parameter, and it represents the distance between the user’s shoulder and his hand. It is considered to be the absolute value of the distance, to assure that the user’s hand is not situated in the –Z axis, but on the +Z axis, facing the device.

The Hand to Shoulder Absolute distance algorithm states that the distance between the user’s shoulder and hand (d), must exceed a value in depth, given by the `DIST_CLICK` variable, and that the hand should move towards the device (the `idle` variable is not 0), in order to perform a click.

```java
    d = abs(joint.position.Z - jshoulder.position.Z);
    hand.distance = d;
    if (d > DIST_CLICK && !hand.idle) {
        hand.idle = 1;
        click(hand, joint.position);
    } else if (d < DIST_CLICK && hand.idle) {
        hand.idle = 0;
    }
```
If the user has not extended his hand sufficiently enough (the pre-set distance), the algorithm does not initialize the click (Figure 4.7).

![Figure 4.7: Hand to Shoulder Absolute Distance Approach – graphical representation](image)

As a feedback for the user, the size \( s \) of the hand in the graphical application varies and it informs the user if he already performed an action or not. If the user clicked, the size of the hand is set at 1.0 unit. The same size is also used when the hand is stationary.

```c
if (h->idle)
    return 1.0;

s = 50 * (DIST_CLICK - h->distance) / DIST_CLICK;
```

However, while the user clicks, the hand size is large at the beginning of the action, reaching a maximum of 50 units, and decreases directly proportional to the distance between the hand and shoulder as the hand extends.

**4.2.2. Hand to Kinect Relative Distance Approach**

The aim of the Hand to Kinect Relative Distance approach is to initiate the click once the user moves his hand sufficiently enough in depth towards the Kinect Sensor Device in a certain amount of time.

The numerical value of the distance in depth is set in the `distToClick` parameter from the configuration file. The numerical value of the necessary frames (`framesToClick`) that need to pass to accomplish a click is also set in the configuration file.
I have implemented two Hand to Kinect Relative Distance approaches. The first approach (Figure 4.8) takes into consideration the distance that the user has to cover with his hand on the Z axis to perform a click, while the second approach (Figure 4.8) considers the closest distance on the Z axis to the Kinect sensor device that was reached by the users’ hand during the frame interval. For both methods, I introduce the previous positions of the hand in the array pastData.

The first Hand to Shoulder Relative Distance algorithm (Figure 4.8) that I implemented states that I am approaching to the Kinect sensor if I have enough elements in the pastData list, which means that enough frames passed (FRAMES_CLICK), and also if the value of the distance between the user’s hand and Kinect sensor device (d), is smaller than the position of the hand when the click action began (pd). The click is accomplished when the user moved the hand sufficiently enough in depth (DIST_CLICK) at each frame.

```plaintext
d = joint.position.Z;
hand.pastData.push_front(d);
if (hand.pastData.size() >= FRAMES_CLICK) {
    float pd = hand.pastData.back();
    hand.pastData.pop_back();
    if (pd > d) { //approaching to kinect
        if (!hand.idle && pd - d > DIST_CLICK) {
            hand.idle = 1;
            click(hand, joint.position);
        }
    } else { //going away from kinect
        if (hand.idle)
            hand.idle = 0;
    }
}
```

---

**Figure 4.8:** Hand to Kinect Relative Distance Approach (1) – graphical representation
The second Hand to Kinect Relative Distance algorithm (Figure 4.9) states that I am approaching to Kinect sensor device when the distance \( (pd) \), that is measured between the hand’s position when the click action began, exceeds the minimum distance reached in the frame interval. The movement counts as a click if the difference between the two distances is greater or equal than the required value set in the configuration file \( (\text{DIST\_CLICK}) \). The minimum distance represents the lowest value in the array, calculated at each frame.

```cpp
......
    it= hand.pastData.begin();
    it++;
    minimum = *it;
    it++;
    for ( ; (it != hand.pastData.end()); it++)
    {
        if(*it < minimum)
            minimum = *it;
    }
    if (pd > minimum) { //approaching to kinect
        if (!hand.idle && pd - minimum > DIST_CLICK) {
            hand.idle = 1;
            click(hand, joint.position);
        }
    } else { //going away from kinect
......
```

**Figure 4.9:** Hand to Kinect Relative Distance Approach (2) – graphical representation
As a feedback for the user, for both Hand to Kinect Relative Distance algorithms, the size $s$ of the hand decreases when the user comes closer to the Kinect sensor and increases when he goes farther. The user can see that he performed a click when the color of the hand becomes darker and the radius of the hexagon is very small, which is actually set to 0.1.

$$s = h->pastData.back() - h->pastData.front();$$
$$s = 20 \times (DIST\_CLICK - s) / DIST\_CLICK;$$

For example, in the first Hand to Kinect Relative Distance approach, I have the $pastData$ array whose dimension is equal to $FRAMES\_CLICK$ set in the configuration file. For the following example its value is 15. The $DIST\_CLICK$ parameter is 10. The steps showing the user feedback for the Hand to Kinect Relative Distance approach are as follows:

User starts to move his hand in front of the Kinect. I register in $pastData$ all the distances between the user’s hand and the Kinect, as follows:

$pastData$: 14
$pastData$: 13.9, 14
$pastData$: 13.8, 13.9, 14

$pastData$: 12.5, 13, 13.1, 13.5, ....13.8, 13.9, 14
$pastData$: 12, 12.5, 13, 13.1, 13.5, ....13.8, 13.9, 14

After the array $pastData$ is full of elements, the size of the hand (the radius of the hexagon) starts to change its dimension and it shows the user if he is approaching or going farther from the Kinect.

Step 1:
$pastData$: 11, 12, 12.5, 13, 13.1, 13.5, ....13.8, 13.9, 14
$$s = 20 \times (10 - (14-11)) / 10 = 14$$ start a click

Step 2:
$pastData$: 10.8, 11, 12, 12.5, 13, 13.1, 13.5, ....13.8, 13.9
$$s = 20 \times (10 - (13.9-10.8)) / 10 = 12.6$$ process of clicking (the radius of the hexagon is decreasing $\Rightarrow$ user is approaching the Kinect)

...  
Step 14:
$pastData$: 3, ...., 10.8, 11
$$s = 20 \times (10 - (11-3)) / 10 = 4$$

Step 15
Even if the value of the user’s hand is negative, for the purpose of the feedback the radius needs to have a positive value and is set to 1.0.

For example, in the second Hand to Kinect Relative Distance approach, the only difference is that I always compare with the minimum distance reached in the frame interval (\(pd_{\text{minimum}}\)).

As we can see, the feedback becomes more intuitive for the user because it follows the user movements more closely.

### 4.2.3. Hand to Kinect Absolute Approach

The goal of the Hand to Kinect Absolute method is to perform a click by passing a hand through an imaginary boundary (a threshold) between the user and the Kinect Sensor Device.
A click is initialized if the distance $d$ between the Kinect device and the user’s hand on the Z axis, reaches the necessary threshold, which is the distance $DIST\_CLICK$, set in the configuration file. Another condition to do a click is that the user’s hand should not be still, so the flag $\text{idle}$ is set to 1.

$$d = \text{joint.position.Z};$$
$$\text{hand.distance} = d;$$
$$\text{if} \ (d < DIST\_CLICK \ &\ & \text{!hand.idle}) \ {\text{hand.idle}} = 1;$$
$$\text{click(hand, joint.position);}$$
$$\text{else if} \ (d > DIST\_CLICK \ &\ & \text{hand.idle}) \ {\text{hand.idle}} = 0;$$

If the user did not pass the boundary, as he is in the exterior of the threshold area, the click is not performed (Figure 4.10).

![Figure 4.10: Hand to Kinect Absolute Approach – graphical representation](image)

The user knows that he did not moved the hand on the screen as he sees the feedback of the hand size set at 1.0 unit. When the user starts to move the hand, the size of the hand is directly proportional with the current distance of the hand ($\text{distance}$) and the distance of the boundary (the invisible wall), which is $DIST\_CLICK$. The hand’s size decreases while the user’s hand is approaching to the Kinect device.

```
if (h->idle) //idle==1
    return 1.0;

s = (h->distance - DIST\_CLICK)/50;
```
4.3. Summary and Discussion

A system was implemented to evaluate the performance of different pointing strategies using the Kinect. The system runs in Linux OS using the OpenNI framework and NITE middleware. The project was developed on Eclipse Platform, specifically C++ programming language.

The project provides a GUI which allows the user to click on a fixed number of targets and generates statistics about user interaction efficiency.

The application supports different ways of interacting with Kinect device to perform clicking. Using a configuration file, the user can choose which clicking strategy to use, as well as various parameters for configuring it.

I introduce two selection strategies, the temporal one and the dart one. The temporal strategy takes into consideration the time parameter, while the dart strategy considers the distance parameter.

The Hand to Kinect Absolute approach is a Kinect oriented strategy, while all the other methods are user oriented. The Hand to Kinect Absolute approach is a Kinect oriented strategy because it considers the distance between the Kinect and the user. The user performed the click if he reached with his hand the necessary threshold (the invisible wall). All the other dart methods are user oriented because the distance that the user has to cover to perform a click is the distance between the user’s shoulder and his hand.
Chapter 5

Evaluation

In this chapter, I introduce Fitts’s Law, I present the design of the evaluation and then I layout the results of the three tested strategies (Temporal, Hand to Shoulder Absolute Distance, and Hand to Kinect Relative Distance (2)).

5.1. Fitts’s Law

This project explores the use of Fitts’s Law, proposed by Paul Fitts in 1954, as a performance model for HCI and ergonomics\textsuperscript{19}, which is a law based on the Shannon's Theorem. The Shannon formulation expresses, in information theory, that the effective information capacity (bits/s or, simply written, bps) of a communication channel can be transmitted at a specific bandwidth and in the presence of noise.\textsuperscript{20} Following the work of Shannon, in Fitts’s Law, the realization of movement is similar to the transmission of "information" in electronic systems. Movements are assigned an index of difficulty, in "bits", and in carrying out a movement task the human motor system is said to transmit so many "bits of information". If the number of bits is divided by the time to move, then a rate of transmission in "bits per second" can be ascribed.\textsuperscript{21}

Fitts’s Law is a psychological model of human movement used to measure user performance in the design of user interfaces.\textsuperscript{22}

Fitts’s Law predicts that the time required to rapidly move to a target area is a function of the distance to the target and the size of the target. In the Fitts’s Law description of pointing, the parameters of interest are: the time to move to the target ($MT$), the distance (amplitude) of movement from start to the target center ($D$) and the

target width \((W)\). According to Fitts’s Law, the time to move and point to a target is a logarithmic function:

\[
MT = a + b \log_2 \left( \frac{D}{W} + 1 \right) \quad (1)
\]

In equation (1), \(a\) is the start/stop time of the device (intercept) and \(b\), is the inherent speed of the device (slope). The \(a\) and \(b\) parameters are empirically determined constants, that are device dependent.

The logarithm in Fitts's Law is called the index of difficulty \(ID\) for the target, measured in units of bits and describes the difficulty of the motor tasks. We can rewrite the law as:

\[
MT = a + bID \quad (2)
\]

An index of performance \(IP\) (also called throughput \(TP\)), measured in bits/time, can be defined to characterize how quickly pointing can be done. There are two ways to define the \(IP\), as mentioned in ISO 9241-9, the final draft international standard (FDIS) of “Ergonomic requirements for office work with visual display terminals – Part 9: Requirements for non-keyboard input devices”. One way of defining the \(IP\) is:

\[
IP = 1/b \quad (3)
\]

In equation (3), \(b\) is measured in second/bits, thus the unit of \(IP\) is bps and has the disadvantage of ignoring the effect of \(a\).

The other way of defining the \(IP\), which incorporates also the \(a\) constant, is:

\[
IP = \frac{ID_{avg}}{MT_{avg}} \quad (4)
\]

Equation (4) has the disadvantage of depending on the mean \(ID\) and it also depends on the task parameters, such as number, sizes and distances of targets used in measuring the \(IP\). As mentioned in ISO 9241-9, the \(IP\) in equation (4) is an ill-defined concept.

In my project, I used Fitts’s Law to measure the \(IP\) of Kinect input device, by modeling the act of pointing at distance. The user has to point and click on a fixed
number of targets on the designed graphical interface, while he is tracked at distance by Kinect sensor device. Considering that the targets that I designed are circles, the \( ID \) that I used is:

\[
ID = \log_2 \left( \frac{d}{2R} + 1 \right)
\]  

In equation (5), \( R \) is the radius of the target, while \( d \) represents the distance of movement from starting point to center of the circle, measured along the axis of motion.

The IP formula that I used in my project is:

\[
IP = \frac{ID_{avg}}{MT_{avg}}
\]

In conclusion, Fitts’s Law is used to assist in the design of user interfaces and in interface evaluation, but it also helps to study and compare input devices with respect to their pointing capability. Therefore, there are several consequences to be taken into consideration while designing graphical interfaces. For example, one consequence is that buttons and other graphical controls should be a reasonable size, taking into consideration that it takes longer to hit targets that are further away and smaller. Another consequence to be taken into consideration is that the edges and corners of the computer monitor are reached by the cursor while using the mouse, touchpad or trackball, but are not acquired with touchscreens or Kinect device.

### 5.2. Design of the Evaluation

The goal of the evaluation is to measure the efficiency, accuracy and satisfaction of the designed system.

Therefore, I scheduled 6 users (5 men and 1 woman) to interact with the graphical interface, while being tracked by Kinect. Each user had to test one time the three conditions (Temporal – T, Hand to Shoulder Absolute Distance – HSA and Hand to Kinect Relative Distance 2 – HKR 2) that I implemented, for a fixed number of targets, as represented in the following table:
<table>
<thead>
<tr>
<th>User</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>T</td>
<td>HSA</td>
<td>HKR 2</td>
</tr>
<tr>
<td>U2</td>
<td>T</td>
<td>HKR 2</td>
<td>HSA</td>
</tr>
<tr>
<td>U3</td>
<td>HSA</td>
<td>T</td>
<td>HKR 2</td>
</tr>
<tr>
<td>U4</td>
<td>HSA</td>
<td>HKR 2</td>
<td>T</td>
</tr>
<tr>
<td>U5</td>
<td>HKR 2</td>
<td>T</td>
<td>HSA</td>
</tr>
<tr>
<td>U6</td>
<td>HKR 2</td>
<td>HSA</td>
<td>T</td>
</tr>
</tbody>
</table>

**Table 1:** The balance conditions to remove bias during experiment

All the users had to click on 30 round targets (circles) with radius of 30 or 40 units. In the configuration files, I have varied the positions on the screen and the widths of the circles. The same set of 30 targets was used for each of the algorithm. The tasks to be accomplished by users were balanced (Table 1) in order to avoid bias, therefore, the tasks does not have an influence on performance. The last 20 targets were taken into consideration to calculate the IP. The first 10 targets represent the training set that each user had to click on for each algorithm to get used with the system.

At the end of the experiment each user received a questionnaire to rank the three tested methods, respecting the order of testing. In the design of the questionnaire I asked the users to answer to three questions. According to Fitts’s Law, only untrained movements are described, therefore to get a valid IP I needed users that had no experience with the Kinect device. Thus, the first question was checking the previous experience of the users. The second question was asking users to rank the efficiency of the tested strategies on a scale of Very good, Good, Neither good nor bad, Bad, Very bad. The last question was asking for suggestions how to improve the developed strategies.

**5.3. Results and Interpretation**

The results of the evaluation with real users provides information regarding the efficiency of the system by measuring the IP using the Fitts’s Law, the accuracy of
the strategies by measuring the number of misses, and the satisfaction of the users by the questionnaire results.

According to the IP obtained results (Graph 1), the Hand to Kinect Relative Distance (2) strategy has the best performance because we obtained an IP average of 1.42, while for the other two strategies we obtained 1.15 for the Temporal one and 0.93 for the Hand to Shoulder Absolute Distance strategy.

![Graph 1](image1.png)

**Graph 1:** The average IP for the three tested strategies

Plotting the performance on each individual target (Graph 2); we can see that the average performance gets better with time. This shows the effect on training of the device, and this is why we do not consider the first 10 targets in the calculation of the IP. We can see that the temporal method is the fastest to learn, because it has the least difference in the performance before and after the training period.

![Graph 2](image2.png)

**Graph 2:** The performance per target for the three tested strategies
A one-way within-subject ANOVA test, in which all users test all the selection strategies, was conducted to measure the effect of the selection strategies over the IP (Table 2). Three conditions were used to make the comparisons, as following: (T vs. HSA), (T vs. HKR 2) and (HSA vs. HKR 2). There is a high statistically significant effect of the selection strategies on the IP, as we obtained p=0.004, so p < 0.01. Three paired samples t-tests were used to make post-hoc comparisons. No significant difference (0.13991479) was found for the (T vs. HSA) pair, and statistical significant difference (p<0.05) for the (T vs. HKR 2) and (HSD vs. HKR 2) pairs, for which we obtained 0.01209377 and 0.01123922. We can say that HKR 2 is a better strategy than the T strategy and also than the HSA strategy.

<table>
<thead>
<tr>
<th>Performance Strategy</th>
<th>Errors/task</th>
<th>IP (bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T – Temporal</td>
<td>0.02</td>
<td>1.15</td>
</tr>
<tr>
<td>HSA – Hand to Shoulder Absolute</td>
<td>0.41</td>
<td>0.93</td>
</tr>
<tr>
<td>HKR 2 – Hand to Kinect Relative (2)</td>
<td>0.44</td>
<td>1.42</td>
</tr>
</tbody>
</table>

**Table 2**: Pointing efficiency with three different performance algorithm

Looking at the amount of errors per click (Table 2), we can see that the Temporal strategy is the most accurate because it has the least frequency of errors per click. The Temporal strategy is 10 times more accurate than the other two strategies.

The average performance on screen for the three tested strategies was calculated, dividing the screen on 9 equal areas. The last 20 targets of the test were distributed on the 9 equal areas and then counted (Table 3).

```
2 2 2
1 3 3
3 3 1
```

**Table 3**: The total number of targets on each area tested for all the three strategies

The average performance on screen for the three tested strategies is represented in **Figure 5.1**. The average IP is higher on the right side of the screen. An explanation could be because all the users tested the strategies using their right hand. On the left side we can see that especially on the corners, the performance is lower. A good performance is obtained in the center area of the screen, due to the easiness of interacting with Kinect device. The Hand to Shoulder Absolute and the Hand to
Kinect Relative Distance (2) got the best IP in the center of the screen, while the Temporal strategy is almost constant in all the areas.

![Figure 5.1: The average performance on screen for the three tested strategies](image)

This evaluation, on the average performance on the screen, was not planned from the beginning; therefore, better statistical information could be obtained with more targets equally distributed on the screen.

Regarding the qualitative evaluation, I assigned numbers to the possible replies to be able to calculate the average (Table 4). The Very good option got a 5, while the Very bad answer got a 1. The following table shows the average ranking of each strategy:

<table>
<thead>
<tr>
<th>Performance Strategy</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>T – Temporal</td>
<td>4.6</td>
</tr>
<tr>
<td>HSA – Hand to Shoulder Absolute</td>
<td>2</td>
</tr>
<tr>
<td>HKR 2 – Hand to Kinect Relative (2)</td>
<td>3.33</td>
</tr>
</tbody>
</table>

**Table 4:** Average user ranking of the three tested strategies
The users consider the Temporal method as the best one, followed by the Hand to Kinect Relative Distance (2). They all agreed that the Hand to Kinect Relative Distance (2) is *Bad*. An interpretation of the result could be due to fatigue or easiness. Even if people ranked the temporal strategy as the best one, the best IP we obtain is for the Hand to Kinect Relative strategy. A possible reason of the obtained result could be that people are considering what is easy to use (user perception); however the Hand to Kinect Relative Distance (2) is the one that gives the best efficiency.

5.4. Summary and Discussion

In this chapter, I present the results of the three tested strategies (Temporal, Hand to Shoulder Absolute Distance, and Hand to Kinect Relative Distance (2)) using 6 users and 30 targets. The first 10 targets represents the training period. The IP shows that the Hand to Kinect Relative Distance (2) strategy has the best performance. The ANOVA test confirms that the Hand to Kinect Relative Distance (2) is statistically better than the other two strategies. On the other hand, the temporal strategy has the best accuracy than the other two strategies. The users graded the temporal strategy as the best one, followed by the Hand to Kinect Relative Distance (2). Users think that the most accurate strategy is the one that also performs best.
Chapter 6

Conclusions and Possible Research Directions

The project introduces different strategies for pointing and selecting from a distance using only the human hand, by being tracked by the Microsoft Kinect sensor device.

Two selection strategies were designed, the temporal one and the dart one. The temporal strategy requires the user to hold still over the target to trigger a click. The dart strategy requires the user to move his hand towards Kinect device to perform a click. The dart strategy can be additionally divided into two approaches, depending on which distances were measured: the hand to shoulder distance or the hand to Kinect distance. Then I introduce the design of the visual feedback which is based on geometrical figures and colors with the goal to show the user how the system recognizes the pointing and selections. A system was implemented to evaluate the performance of the pointing strategies using the Kinect sensor device. The system provides a graphical interface which allows the user to click on a fixed number of targets. Three implemented performance strategies were tested with real users with a defined selection of targets stored in the configuration files. In the end of the project, I provide an evaluation on performance of the implemented pointing strategies using the Fitts’s Law. A detailed statistical evaluation is presented for the three tested strategies, using an ANOVA test. The calculated index of performance shows that the Hand to Kinect Relative Distance (2) strategy has the best performance. The ANOVA test confirms that the Hand to Kinect Relative Distance (2) is statistically better than the other two strategies. The higher the index of performance, the better is the strategy.

As possible research directions, the Relative Distance Hand to Shoulder approach could be implemented to perform a bare hand click on a graphical interface. The gesture accuracy and the system performance could be improved by adding audio feedback. Another future development could be to explore the use of various visual feedbacks. Finally, another research direction could be the evaluation of selection strategies with two hands.
References


[5] D. Vogel, R. Balakrisham, *Distant Freehand Pointing and Clicking on Very Large, High Resolution Displays*, Department of Computer Science, University of Toronto


[8] https://github.com/OpenKinect/libfreenect


Appendix A

The attached CD contains the folder OpenNI_hand. This project is implemented in Linux OS using the OpenNI framework and NITE middleware. The project is developed on Eclipse platform, specifically C++ programming language.

In this folder, the source files are: Main.cpp, SceneDrawer.cpp, User.cpp, Timer.cpp, and the headers are: SceneDrawer.h, User.h, Timer.h.

The folder contains the configuration files: config_dart_ad_str.txt, config_dart_rd2_str.txt, config_dart_rd_str.txt, config_dart_th_str.txt, config_temp_str.txt.

The folder also contains the output files: out_dart_abs.txt, out_dart_rel.txt, out_dart_rel.diff.txt, out_dart_th.txt, out_temp.txt.

Besides for the OpenNI library and the NITE middleware, the other libraries used in the application are the GL and glut to draw the user interface, but also the config++ library to configure the application parameters using an external file.
Appendix B
Example of configuration files

Config_dart_ad_str.txt

```
targetSize: 40;
targets: ([0.5, 0.5, 40.0], [0.9, 0.1, 30.0], [0.9, 0.2, 30.0], [0.7, 0.9, 40.0],
[0.8, 0.7, 30.0], [0.3, 0.2, 40.0], [0.5, 0.7, 30.0], [0.9, 0.9, 30.0],
[0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.5, 0.8, 30.0], [0.8, 0.1, 30.0],
[0.4, 0.6, 40.0], [0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.7, 0.5, 40.0],
[0.6, 0.3, 30.0], [0.2, 0.3, 30.0], [0.7, 0.4, 40.0], [0.3, 0.9, 30.0],
[0.2, 0.4, 30.0], [0.7, 0.6, 40.0], [0.8, 0.1, 40.0], [0.2, 0.7, 40.0],
[0.9, 0.7, 30.0], [0.6, 0.4, 40.0], [0.5, 0.1, 40.0], [0.5, 0.9, 30.0],
[0.3, 0.1, 30.0], [0.6, 0.8, 40.0]);
```

```
clickMethod: 1;
distanceToClick: 325;
stopAfter: 30;
scale: 300;
```

Config_dart_rd2_str.txt

```
targetSize: 30;
targets: ([0.5, 0.5, 40.0], [0.9, 0.1, 30.0], [0.9, 0.2, 30.0], [0.7, 0.9, 40.0],
[0.8, 0.7, 30.0], [0.3, 0.2, 40.0], [0.5, 0.7, 30.0], [0.9, 0.9, 30.0],
[0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.5, 0.8, 30.0], [0.8, 0.1, 30.0],
[0.4, 0.6, 40.0], [0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.7, 0.5, 40.0],
[0.6, 0.3, 30.0], [0.2, 0.3, 30.0], [0.7, 0.4, 40.0], [0.3, 0.9, 30.0],
[0.2, 0.4, 30.0], [0.7, 0.6, 40.0], [0.8, 0.1, 40.0], [0.2, 0.7, 40.0],
[0.9, 0.7, 30.0], [0.6, 0.4, 40.0], [0.5, 0.1, 40.0], [0.5, 0.9, 30.0],
[0.3, 0.1, 30.0], [0.6, 0.8, 40.0]);
```

```
clickMethod: 4;
distanceToClick: 25;
framesToClick: 5;
stopAfter: 30;
scale: 300;
```

Config_dart_rd_str.txt

```
targetSize: 30;
targets: ([0.5, 0.5], [0.9, 0.1], [0.9, 0.2], [0.7, 0.9], [0.8, 0.7], [0.3, 0.2],
[0.5, 0.7], [0.9, 0.9], [0.1, 0.9], [0.5, 0.5], [0.5, 0.8], [0.8, 0.1],
[0.4, 0.6], [0.1, 0.9], [0.5, 0.5], [0.7, 0.5], [0.6, 0.3], [0.2, 0.3],
[0.7, 0.4], [0.3, 0.9]);
```

```
clickMethod: 2;
```
distanceToClick: 10;
framesToClick: 15;
stopAfter: 20;
scale: 300;

**Config_dart_th_str.txt**
targetSize: 40;
targets: ([0.5, 0.5], [0.9, 0.1], [0.9, 0.2], [0.7, 0.9], [0.8, 0.7], [0.3, 0.2], [0.5, 0.7], [0.9, 0.9], [0.1, 0.9], [0.5, 0.5], [0.5, 0.8], [0.8, 0.1], [0.4, 0.6], [0.1, 0.9], [0.5, 0.5], [0.7, 0.5], [0.6, 0.3], [0.2, 0.3], [0.7, 0.4], [0.3, 0.9]);
clickMethod: 3;
distanceToClick: 1000;
stopAfter: 20;
scale: 300;

**Config_temp_str.txt**
targetSize: 20;
targets: ([0.5, 0.5, 40.0], [0.9, 0.1, 30.0], [0.9, 0.2, 30.0], [0.7, 0.9, 40.0], [0.8, 0.7, 30.0], [0.3, 0.2, 40.0], [0.5, 0.7, 30.0], [0.9, 0.9, 30.0], [0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.5, 0.8, 30.0], [0.8, 0.1, 30.0], [0.4, 0.6, 40.0], [0.1, 0.9, 40.0], [0.5, 0.5, 30.0], [0.7, 0.5, 40.0], [0.6, 0.3, 30.0], [0.2, 0.3, 30.0], [0.7, 0.4, 40.0], [0.3, 0.9, 30.0], [0.2, 0.4, 30.0], [0.7, 0.6, 40.0], [0.8, 0.1, 40.0], [0.2, 0.7, 40.0], [0.9, 0.7, 30.0], [0.6, 0.4, 40.0], [0.5, 0.1, 40.0], [0.5, 0.9, 30.0], [0.3, 0.1, 30.0], [0.6, 0.8, 40.0]);
clickMethod: 0;
framesToClick: 30;
stopAfter: 30;
scale: 300;