

TouchCU

Texas Christian University
April 23, 2014



[DEVELOPER GUIDE]

Version 4.1

©2013-2014 Computer Science Department, Texas Christian University

Table of Contents

I.	Revision Signatures	III
II.	Revision History	IV
1	Introduction	1
1.1	Purpose	1
1.2	Overview	1
2	System Overview.....	2
2.1	System Components	2
2.1.1	TouchCU Standalone Application	2
2.1.2	Microsoft Kinect for Windows	2
2.1.3	Standard Projector	2
2.2	Environment.....	2
3	Development Getting Started	3
3.1	Microsoft Kinect.....	3
3.2	Standalone Application Development	3
3.2.1	Creating an Application.....	3
4	Standalone Application Development	4
4.1	System Overview.....	4
4.1.1	Kinect Interactions	4
4.1.2	Touch Inputs.....	4
4.1.3	System Tray.....	4
4.1.4	Debug Menu.....	4
4.2	Classes.....	5
4.2.1	Touch.....	5
4.2.2	Speech.....	6
4.2.3	Debug.....	7
4.2.4	Application	8
4.3	The GUI	10
4.3.1	Calibration Window	10
4.3.2	Settings Window	10
4.3.3	Debug Menu.....	10
4.3.4	Speech Visualizer	11

4.3.5 System Tray Menu	11
4.4 Kinect Integration	12
4.4.1 Discovering and Initializing the Kinect	12
4.4.2 Frame Listeners	13
4.5 Deriving Touch Injection Parameters.....	15
5 Glossary of Terms.....	17
6 Appendix	18
6.1 Appendix A: Gesture Tables	18
6.2 Appendix B: Voice Command Tables	19
6.3 Appendix C: Use Case Diagram	20
6.4 Appendix D: Use Case Scenarios	21

I. Revision Signatures

By signing this document, the team member is acknowledging that he/she has read through this document thoroughly and has certified that the information within this document is accurate and satisfies all requirements.

<u>Name</u>	<u>Signature</u>	<u>Date Signed</u>
Trenton Bishop		
Yizhou Hu		
Blake LaFleur		
Thales Lessa		
Matthew Spector		

II. Revision History

<u>Version</u>	<u>Changes</u>	<u>Edited</u>
v1.0	Initial Documentation Draft	04 February 2014
v2.0	Additional development information added	18 February 2014
v3.0	Added math explanation and updated based on current project state	28 February 2014
v4.0	Iteration 4	25 March 2014
v4.1	Final Draft	23 April 2014

1 Introduction

1.1 Purpose

The purpose of this document is to provide developers with the appropriate information so that they can modify the TouchCU source code to meet their needs and requirements. Each major component of the application will be broken down and outlined.

1.2 Overview

This document includes the following sections.

Section 2 – System Overview: Covers the main components of the system.

Section 3 – Development Getting Started: Covers an introduction to programming with the Microsoft Kinect.

Section 4 – Standalone Application Development: Covers the classes and programming of the TouchCU Application.

Section 5 - Glossary of Terms: Includes a list of abbreviations and their technical terms and their associated definitions.

2 System Overview

2.1 System Components

TouchCU is comprised of the standalone application, a computer running Windows 8 or newer, the Microsoft Kinect for Windows, and a standard projector.

2.1.1 TouchCU Standalone Application

The TouchCU standalone application is used to initially calibrate the system for use. Once the user has calibrated the system, TouchCU runs in the background. The application's settings, screen calibration, and debug mode can all be accessed from the system tray menu.

2.1.2 Microsoft Kinect for Windows

The Kinect is used in all aspects of our system. It uses a proprietary USB cable for power and connection to the computer. The Kinect tracks 20 joints while in Default Mode and captures at a rate of 30Hz. Our system utilizes the AudioStream, ColorStream, DepthStream, and SkeletonStream.

2.1.3 Standard Projector

TouchCU is compatible with any projector. A resolution of 1920x1080 is recommended for best performance, if 1920x1080 is not possible, choose a resolution as close to this as possible. The projector can use either analog or digital input as long as it supports the required resolution.

2.2 Environment

TouchCU will be a software application that can be installed on any computer running Windows 8 or higher. The Kinect Driver v1.8.0 must be installed along with the .NET Framework 4.5 (included in Windows 8).

The computer must meet the minimum requirements for using the Microsoft Kinect, which are:

- minimum of a dual-core processor @ 2.66 GHz
- 2 GB of RAM
- dedicated USB 2.0 port

3 Development Getting Started

3.1 Microsoft Kinect

In order to use the Kinect for development, a Kinect for Windows must be used. The user must also download the Kinect for Windows SDK and Developer Toolkit from <http://www.microsoft.com/en-us/kinectforwindowsdev/start.aspx>. The Kinect for Windows SDK will install the driver, and the required APIs and device interfaces used for Kinect development. The Kinect for Windows Developer Toolkit includes Human Interface Guidelines, sample applications (with full source code), Kinect Studio, Face Tracking SDK, along with other development resources.

- Kinect for Windows SDK
- Kinect for Windows Developer Toolkit

TouchCU was developed and tested using v1.8.0 of the SDK and Developer Toolkit.

3.2 Standalone Application Development

The standalone application was written in Microsoft Visual Studio 2012 and used C# along with the .NET Framework 4.5. It is necessary to use this IDE when developing for the Microsoft Kinect since the Kinect for Windows SDK is integrated into Microsoft Visual Studio. This IDE can be purchased from Microsoft and can be found at <http://msdn.microsoft.com/en-US/vstudio>.

Note: If you are a student, contact your professor or department for access to DreamSpark. This will allow you to obtain Visual Studio at either little or no cost. Visit the following link for more details.

<https://www.dreamspark.com/Student/Default.aspx>

3.2.1 Creating an Application

1. Create a new C# WPF project and give it a name.
2. Within your Visual Studio project, right-click on solution and click **Add Reference**.
3. Select the tab **.NET** from the window that appears.
4. Verify that the **Microsoft.Kinect** listing is present.
 - a. If there is no reference to **Microsoft.Kinect**, select the *Browse* tab.
 - b. Navigate to *C:\Program Files\MicrosoftSDK\Kinect\V1.8\Assemblies\Microsoft.Kinect.dll*.
 - c. Select **OK** to add the reference to your project.

The developer can now add **using Microsoft.Kinect** in the import section of their code to access the Kinect in their application.

4 Standalone Application Development

4.1 System Overview

4.1.1 Kinect Interactions

The KinectControl class controls various aspects of the Kinect hardware, and sets up the different streams for use with the TouchCU application. The KinectFrameListeners class contains the EventListeners that are triggered by the Kinect when specific frames are ready to be processed. There are currently two listeners that have been created. One for calibration mode, and the other for TouchAnalyser.

4.1.2 Touch Inputs

The TouchAnalyser class takes in Kinect input points, convert them into pixel locations on the screen. It then detects the number of touch points in action and construct touch points for the Windows 8 Touch Injection API.

4.1.3 System Tray

The System Tray class contains the ability to be able to open all windows associated with the TouchCU application, as well as exit the application completely.

4.1.4 Debug Menu

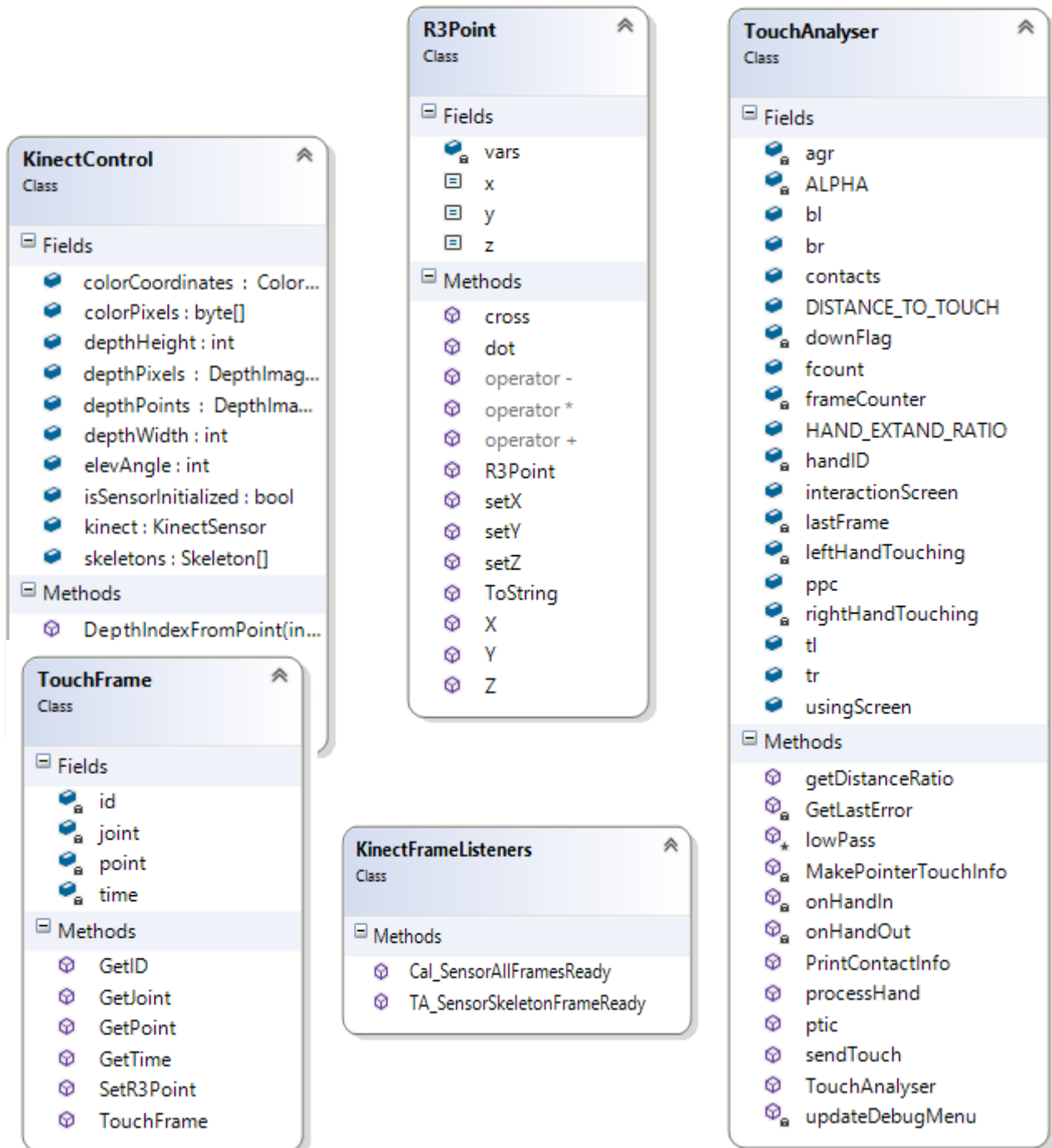
The debug menu consists of three WPF windows: the main debug window, the touch tracker, and the manipulation mode. It takes input values from 1) output of TouchAnalyser and 2) Windows Message loop. These values are displayed side-by-side in the main debug window, which allows for the debugging of our touch injection.

The touch tracker window visualizes the users touch points on the canvas by allowing the users to draw on the canvas by touching it. This allows verification of the location of touch injection as well as the performance of the smoothing parameters.

The manipulation window allows the user to manipulate the image on the screen using one & two-handed gestures. The image on the manipulation window can be dragged, rotated, and zoomed. After the user successfully manipulations the image, the manipulation data is displayed in the main debug window.

4.2 Classes

4.2.1 Touch



4.2.2 Speech

SpeechCommander

Class

- Fields
 - actionChoices
 - command
 - commandChoices
 - e
 - grammarIndex
 - readyTimer
 - sensor
 - settingsWindow
 - speechRecognizer
 - startListening
- Properties
 - touchKeyboardProc
- Methods
 - CloseDebugMenu
 - CloseKeyboard
 - CloseSettings
 - CloseStartMenu
 - CloseWindow
 - FindWindow
 - getIndex
 - GetKinectRecognizer
 - InitializeSpeechRecognition
 - OpenDebugMenu
 - OpenKeyboard
 - OpenMyDocs
 - OpenSettings
 - OpenStartMenu
 - PostMessage
 - ReadyTimerTick
 - SpeechCmdNotifier
 - SpeechCommander
 - SreSpeechHypothesized
 - SreSpeechRecognitionRejected
 - SreSpeechRecognized
 - StartSpeechRecognition
 - switchGrammar

SpeechVisualizer

Class

- Window
- Methods
 - SpeechVisualizer

4.2.3 Debug

The image displays five screenshots of class inspection windows from a development tool, showing the internal structure of various classes:

- DebugMain** (Class, Window):
 - Fields: `_TouchCollection`, `interactionScre...`, `manipulation`, `screenWorking...`, `touchTracker`
 - Properties: `TouchCollection`
 - Methods: `CloseDebug_Cli...`, `DebugMain`, `Manipulation_C...`, `PointTracker_Cl...`
- TouchData** (Class):
 - Properties: `Event`, `Pointer`, `PointerID`, `X`, `Y`, `Z`
- Debug_TouchTracker** (Class, Window):
 - Fields: `debugMain`, `dragStartedP1`, `dragStartedP2`, `holdCounter`, `isHold`, `previousPT1`, `previousPT2`, `primaryTouchID`, `primaryTouchTimer`, `secondaryTouchID`, `secondaryTouchTim...`
 - Methods: `Debug_TouchTracker`, `DrawingCanvas_Tou...`, `DrawingCanvas_Tou...`, `DrawingCanvas_Tou...`, `drawLine`, `PrimaryTouchUp`, `SecondaryTouchUp`
- Debug_Manipulation** (Class, Window):
 - Fields: `debugMain`, `mControlTimer`, `mData_Control`, `previousPT1`, `primaryTouchID`, `primaryTouchTimer`, `secondaryTouchID`, `secondaryTouchTim...`
 - Methods: `Debug_Manipulation`, `mControlDisplayStart`, `Preview_TouchDown`, `Preview_TouchUp`, `PrimaryTouchUp`, `SecondaryTouchUp`, `Window_InertiaStar...`, `Window_Manipulati ...`, `Window_Manipulati ...`, `Window_Manipulati ...`
- Debug_ManipulationData** (Class, UserControl):
 - Methods: `Debug_ManipulationData`

4.2.4 Application

Settings
Sealed Class
→ ApplicationSettingsBase

- Fields
 - defaultInstance
- Properties
 - Correction
 - Default
 - DisableAirGestures
 - DisableVoiceCmd
 - DistanceToTouch
 - EnableAnnotationMode
 - HandExtendRatio
 - JitterRadius
 - MaxDeviationRadius
 - Prediction
 - RunAtWinStart
 - Smoothing
- Methods
 - SettingChangingEventH...
 - Settings
 - SettingsSavingEventHan...

Settings
Class
→ Window

- Methods
 - correctionSlider_ValueChanged
 - distance_to_touch_Slider_ValueChanged
 - hand_extension_Slider_ValueChanged
 - jitterRadiusSlider_ValueChanged
 - maxDeviationRadiusSlider_ValueChanged
 - predictionSlider_ValueChanged
 - Settings
 - SettingsBtn_Click
 - smoothingSlider_ValueChanged

SystemTray
Class

- Fields
 - menulitem_Calibration
 - menulitem_Debug
 - menulitem_Exit
 - menulitem_Settings
 - trayIcon
 - trayMenu
- Methods
 - menulitem_Calibration_Click
 - menulitem_Debug_Click
 - menulitem_Exit_Click
 - menulitem_Settings_Click
 - SystemTray
 - trayIcon_DoubleClick

PointPlaneConversion
Class


- Fields
 - inverse
 - n
 - Pbl
 - Pbr
 - Ptl
 - Ptr
 - q
 - r
- Methods
 - getX
 - getY
 - getZ
 - PointPlaneConversi...

SplashScreen
Class
→ Window

- Fields
 - cw
 - dm
 - dm2
 - sc
 - st
 - startDebug
 - sv
 - ta
- Methods
 - initSplash
 - initSystemTray
 - SplashScreen
 - startDebugMenu

ComboxBoxItem
Class

- Properties
 - ScreenName
 - Value
- Methods
 - ComboxBoxItem
 - ToString

CalibrationWindow 


Class
→ Window

Fields

- cornerCounter
- corners
- polygon
- wBmp
- worker

Methods

- acceptButton_Mous...
- adjElevDown_Click
- adjElevUp_Click
- CalibrationWindow
- Clamp
- DrawLine
- DrawPolygon
- el_MouseLeftButton...
- el_MouseLeftButton...
- el_MouseMove
- elevSlider_DragCom...
- elevSlider_MouseLe...
- getColorRange
- getSPfromEl
- HSL2RGB
- HsvToRgb
- MouseUp_Listener
- moveElevationDown
- moveElevationUp
- OnClosing
- resetButton_Mouse...
- screens_SelectionCh...
- setElev
- worker_DoWork
- worker_RunWorker...

ColorRGB 

Struct

Fields

- B
- G
- R

Methods

- ColorRGB
- explicit operator Co...
- implicit operator Co...

4.3 The GUI

For the development of the TouchCU application, we tried to make the user experience as intuitive as possible. The GUI is composed of the following interfaces:

- Calibration Window
- Settings Window
- Debug Window (main window, point tracker window, manipulation window)
- Speech Visualizer
- System tray menu

4.3.1 Calibration Window

This window, as the name suggests, allows the user to calibrate its projected screen in relation to the Kinect's view. The Calibration window is ALWAYS initiated at program startup, so that TouchCU can be calibrated. It contains a "Canvas" that displays a view of the Kinect's camera, a slider for adjusting the Kinect's vertical angle, a dropdown menu for deciding which display to use when the application has been calibrated, and buttons for resetting calibration points or accepting the calibration points (only enabled after 4 points are chosen). The "Canvas" object with the Kinect's view allows the user to select the 4 corners of the projection screen; these points can be dragged to adjust the calibration area.

4.3.2 Settings Window

This window allows the user to adjust smoothing parameters and enable/disable specific features of the application. The settings window allows the users to adjust the following options:

- Disable air gestures
- Disable voice commands
- Smoothing
- Correction
- Prediction
- Jitter radius
- Max deviation radius
- Distance to consider a touch
- Hand extension
- Save button (must be clicked to save changed settings)

4.3.3 Debug Menu

This window allows the user to debug and test the application. The Debug Window is composed of a main window and two sub-windows, point-tracker and manipulation. The main window provides data feedback for the user. It contains ellipses that change color when either pointer is interacting with Windows. In the middle, there is a list-view that is populated with recognized touch events as they occur, along with the X,Y,Z coordinates from Windows & TouchCU for each pointer. X and Y are pixel coordinates and Z is meters from the projection surface. Lastly, the main window contains buttons to open/close the point-tracker and manipulation windows, as well as close the Debug Window itself.

The point-tracker and manipulation sub-windows fill the remaining part of the screen below the main window. These windows which are activated through the pressing of buttons on the main window, allow for user interaction.

4.3.3.1 Point Tracker Window

This is an extension of the debug menu (opens under it) and is a blank purple screen. Whenever it sees a touch, it follows the user's hand and draws a green path until the user removes its hand from the interaction zone.

4.3.3.2 Manipulation Window

This window is an extension of the debug menu. It contains a graphic of the TCU logo. It allows the user to interact with/manipulate the object on the screen; that is, it allows the user to drag, zoom in (enlarge), zoom out (shrink), and rotate the image displayed on the screen.

4.3.4 Speech Visualizer

This window was designed to make it easier for the user to use voice commands while using TouchCU. Whenever the user says "Addie" (the word that initializes all voice commands), a full screen window opens up with the commands available to the user. They're separated in two ListBox objects; one contains the command words ("Open", "Close") and the other the action words ("Start Menu", "Window", "My Documents", "Settings", "Debug"). At the top of the screen, there are Label objects, containing the selected commands; i.e. when the user says "Open", the word open shows up, then when it says "Debug", the word debug shows up and the command is fired.

4.3.5 System Tray Menu

Our system tray menu is a ContextMenu contained inside System.Windows.Forms and is accessed by right clicking on the TouchCU icon on the system tray. When open, it shows 4 menu items: "Calibration", "Debug", "Settings", and "Exit". The first 3 open their respective windows and "Exit" closes the program.

4.4 Kinect Integration

4.4.1 Discovering and Initializing the Kinect

All of the code for initializing the Kinect can be found in the `InitSensor` function within the `KinectControl` class of the solution.

```
public static void InitSensor()
{
    if (KinectSensor.KinectSensors.Count < 1 )
    {
        MessageBox.Show("No Kinect sensor was found! TouchCU will now close.", "Error");
        Environment.Exit(0);
    }

    if (KinectSensor.KinectSensors.Count > 0)
        foreach (var sensors in KinectSensor.KinectSensors) //Check list of sensors
        {
            if (sensors.Status == KinectStatus.Connected) //If sensor is detected, set it to the sensor we will use
            {
                kinect = sensors;
                break;
            }
        }
}
```

Figure 1

After the Kinect sensor has been chosen it is tested to make sure that it is fully operational, and if so we want to start the sensor using the built in `Start()` function. After the Kinect sensor has been started we need to set the resolution of the streams that we plan to use. (DepthStream, ColorStream, and SkeletonStream) A few arrays to hold the values of each stream, and the skeletons being tracked.

```
KinectControl.kinect.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
KinectControl.kinect.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
KinectControl.kinect.DepthStream.Range = DepthRange.Default;
KinectControl.kinect.SkeletonStream.TrackingMode = SkeletonTrackingMode.Default;
KinectControl.kinect.SkeletonStream.EnableTrackingInNearRange = true;
// this.kinect.SkeletonStream.AppChoosesSkeletons = true;
KinectControl.depthWidth = KinectControl.kinect.DepthStream.FrameWidth;
KinectControl.depthHeight = KinectControl.kinect.DepthStream.FrameHeight;

skeletons = new Skeleton[KinectControl.kinect.SkeletonStream.FrameSkeletonArrayLength];
colorPixels = new byte[kinect.ColorStream.FramePixelDataLength];
depthPixels = new DepthImagePixel[kinect.DepthStream.FramePixelDataLength];
depthPoints = new DepthImagePoint[kinect.DepthStream.FramePixelDataLength];
```

Figure 2

4.4.2 Frame Listeners

Cal_SensorAllFramesReady – Calibration Frame Listener

```

public static void Cal_SensorAllFramesReady(object sender, AllFramesReadyEventArgs e)
{
    if (e == null)
        return;
    bool colorRecv = false;
    bool depthRecv = false;

    using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
    {
        if (null != depthFrame)
        {
            depthFrame.CopyDepthImagePixelDataTo(KinectControl.depthPixels);
            depthRecv = true;
        }
    }
    using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
    {
        if (null != colorFrame)
        {
            colorFrame.CopyPixelDataTo(KinectControl.colorPixels);
            colorRecv = true;
        }
    }

    if (colorRecv && depthRecv)
    {
        KinectControl.kinect.CoordinateMapper.MapColorFrameToDepthFrame(ColorImageFormat.RgbResolution640x480Fps30, DepthImageFormat.Resolution640x480Fps30, KinectControl.depthPixels, KinectControl.depthPoints);
        CalibrationWindow.wbmp.WritePixels(new Int32Rect(0, 0, CalibrationWindow.wbmp.PixelWidth, CalibrationWindow.wbmp.PixelHeight), KinectControl.colorPixels, CalibrationWindow.wbmp.PixelWidth * sizeof(int), 0);

        if (SplashScreen.cw.IsActive && SplashScreen.cw.cornerCounter > 3) //Max of 80"
        {
            double ld = Canvas.GetLeft(SplashScreen.cw.bottomRightMarker);
            int li = (int)ld;
            //MessageBox.Show("Double: " + ld + "\nInt: " + li);
            int depth = KinectControl.depthPixels[KinectControl.DepthIndexFromPoint(((int)Canvas.GetLeft(SplashScreen.cw.bottomRightMarker), (int)Canvas.GetTop(SplashScreen.cw.bottomRightMarker)))]-Depth;

            if (depth < 1300 || depth > 3000)
                SplashScreen.cw.polygon.Fill = System.Windows.Media.Brushes.Red;
            else
                SplashScreen.cw.polygon.Fill = new SolidColorBrush(CalibrationWindow.getColorRange(depth));
        }
    }
}

```

Figure 3

When the calibration window is opened this frame listener is added to the Kinect Listeners' Action Event. Depth frame and color frame information is then loaded into the appropriate arrays, and resources are then given back to the Kinect. Then we want to map the depth frame to the color frame to align the two streams together to make sure that the depth of the pixel that the user clicks on is correct for both streams. Once the streams are mapped we can display the updated color stream bitmap on the window.

TA_SensorSkeletonFrameReady – Touch Analyzer Frame Listener

```
public static void TA_SensorSkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
    if (e == null)
        return;

    bool skelRecv = false;
    using (SkeletonFrame skel = e.OpenSkeletonFrame())
    {
        if (skel != null)
        {
            skel.CopySkeletonDataTo(KinectControl.skeletons);
            skelRecv = true;
        }
    }

    if (skelRecv)
    {
        if (KinectControl.skeletons.Length > 0)
        {
            foreach (Skeleton skel in KinectControl.skeletons)
            {
                if (skel.Joints[JointType.HandLeft].TrackingState == JointTrackingState.Tracked)
                {
                    SplashScreen.ta.processHand(skel.Joints[JointType.WristLeft], skel.Joints[JointType.HandLeft], 11);
                }
                if (skel.Joints[JointType.HandRight].TrackingState == JointTrackingState.Tracked)
                {
                    SplashScreen.ta.processHand(skel.Joints[JointType.WristRight], skel.Joints[JointType.HandRight], 22);
                }
                if (skel.Joints[JointType.HandRight].TrackingState == JointTrackingState.Tracked || skel.Joints[JointType.HandLeft].TrackingState == JointTrackingState.Tracked)
                {
                    SplashScreen.ta.sendTouch();
                }
            }
        }
    }
}
```

Figure 4

Once calibration has been completed we switch the frame listener from Cal_SensorAllFramesReady to TA_SensorSkeletonFrameReady because the DepthStream and ColorStream have been turned off at this time. This allows us to increase the frame rate of the Kinect from 12FPS to its maximum 30FPS.

Similarly to the previous frame listener, we want to copy our data into the appropriate arrays to give the resources back to the Kinect. Once completed, we want to traverse through our list of skeletons to find the first one that has all joints successfully tracked. The wrist joint, hand joint, and ID are passed to the TouchAnalyzer in order for it to compute the hand location in terms of the calibrated projection area. Lastly we call sendTouch() to process each hand and inject the data into the Windows TouchInjection API.

4.5 Deriving Touch Injection Parameters

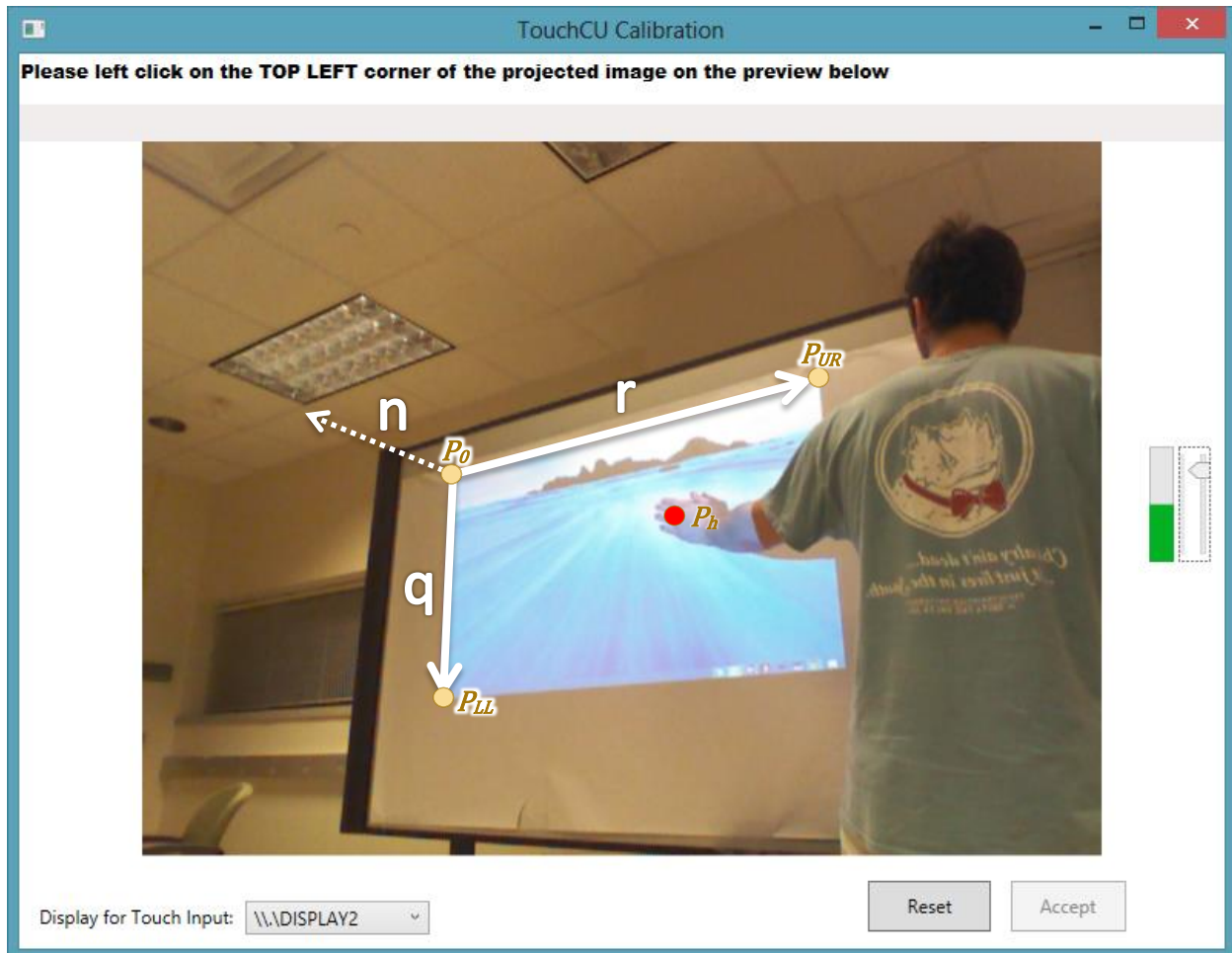


Figure 5

Every point in the space that the Kinect can see is represented by three Cartesian coordinates (x,y,z) . To identify the “touch” surface, these Cartesian coordinates are used to represent a vector from the Kinect’s origin (it’s camera/depth detector) to that particular point as $\vec{u}, \vec{v} \in \mathbb{R}^3$.

Note: Figure 2 displays the view as seen from the Kinect, please note this is a mirrored image.

The upper left corner P_0 is used as a base point. Then, two vectors: \vec{r} from the upper left corner to the upper right corner and \vec{q} from the upper are calculated. \vec{r} and \vec{q} determines the plane the projected image is in. For any pixel in the image, it can be represented as $(a\vec{r} + b\vec{q})$ from P_0 , with $a, b \in \mathbb{R}, 0 \leq a, b \leq 1$.

$$\vec{r} = \overrightarrow{P_{UR}} - \overrightarrow{P_0}$$

$$\vec{q} = \overrightarrow{P_{LL}} - \overrightarrow{P_0}$$

The cross product, \vec{n} , of \vec{r} and \vec{q} is then taken. \vec{n} points vertically out from the plane.

$$\vec{n} = \frac{\vec{r} \times \vec{q}}{|\vec{r} \times \vec{q}|}$$

For any hand position in the 3D space, the different between the input point and the UL corner point can be represented by a linear combination of \vec{r} , \vec{q} and \vec{n} .

$$\vec{P}_h - \vec{P}_0 = x\vec{r} + y\vec{q} + z\vec{n}$$

The coefficients x, y and z are determined by this formula:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = (\vec{r} \quad \vec{q} \quad \vec{n})^{-1}(\vec{P}_h - \vec{P}_0)$$

- Screen Injection $(x_0, y_0) = (width \cdot x, height \cdot y)$
- Distance from hand to screen $d = z$

5 Glossary of Terms

- **Audio Stream** – Data stream from the Kinect that allows for high-quality audio capture that can be used for voice commands.
- **Color Stream** – Data stream from the Kinect that acts as an RGB web camera.
- **Depth Stream** – Data stream from the Kinect that creates a depth map from the distance of the sensor to the object.
- **EOD** - End of Day.
- **HDMI** - High-Definition Multimedia Interface.
- **IDE** - Integrated Development Environment.
- **Kinect** - A motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with PC through a natural user interface using gestures and spoken commands. The Kinect recognizes 20 joints on the human body at a capture rate of 30 Hz.
- **Kinect Studio** - Debugging tool for the Kinect sensor that allows the depth and color streams to be viewed in real time.
- **NTASC** - North Texas Area Student Conference.
- **Rich Interaction** – Supports multi-touch.
- **SDK** - Software Development Kit.
- **Skeletal Stream** – Data stream from the Kinect that tracks up to 20 joints on the human body.
- **SRS** - Student Research Symposium.
- **TBD** - To Be Determined.
- **TCU** - Texas Christian University.
- **TouchCU** - The team & project name for the 2013-2014 senior design team.
- **USB** - Universal Serial Bus.

6 Appendix

6.1 Appendix A: Gesture Tables

Single Hand Gestures		
Name of Gesture	How it's Performed	What it's Used For
Tap (GR 1)	Tap an item on the screen once.	Simulates a left-click from a mouse.
Double-Tap (GR 2)	Tap an item on the screen twice.	Simulates a double left-click from a mouse.
Hold (GR 3)	Tap an item on the screen and hold.	Simulates a right-click from a mouse.
Drag (GR 4)	Tap and hold the screen while moving in any direction.	Simulates moving an object on the screen.

Two Hand Gestures		
Name of Gesture	How it's Performed	What it's Used For
Zoom (GR 5)	Both hands will be placed on the screen and move either farther or closer apart.	Simulates making an object larger or smaller on the screen.
Rotate (GR 6)	Both hands will be placed on the screen to emulate a clockwise or counter-clockwise motion.	Simulate moving the object around a center point.

Air Gestures		
Name of Gesture	How it's Performed	What it's Used For
Swipe Left (GR 7)	One hand in mid-air will move a short distance to the left.	Simulates using the left arrow on the keyboard.
Swipe Right (GR 8)	One hand in mid-air will move a short distance to the right.	Simulates using the right arrow on the keyboard.

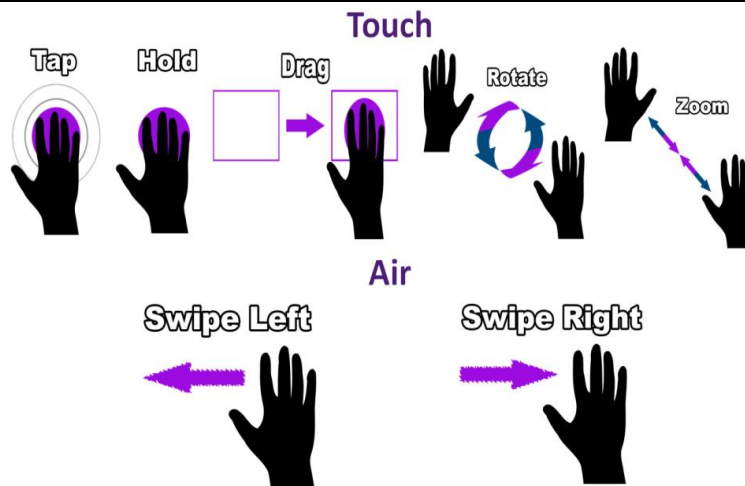


Figure 6

6.2 Appendix B: Voice Command Tables

Trigger Word		
Name of Command	How it's Performed	What it's Used For
Addie (VCR 1) [ad-ee] ['ædi:]	User will say "Addie" aloud followed by a command word + action word.	Initiates the voice recognition process.

Command Words		
Name of Command	How it's Performed	What it's Used For
Open (VCR 2) [oh-puh n]	User will say "Open" aloud followed by an action word.	Used to open the following action word.
Close (VCR 3) [kloh-z]	User will say "Close" aloud followed by an action word.	Used to close the following action word.

Action Words		
Name of Command	How it's Performed	What it's Used For
Start Menu (VCR 4) [stahrt men-yoo]	User will say "Open/Close Start Menu" aloud.	Opens or closes the Windows Start Menu.
Window (VCR 5) [win-doh]	User will say "Close Window" aloud.	Closes the active window.
My Documents (VCR 6) [mahy dok-yuh-muh nts]	User will say "Open My Documents" aloud.	Opens the user's Documents.
Settings (VCR 7) [set-ings]	User will say "Open Settings" aloud.	Opens the TouchCU settings menu.
Debug (VCR 8) [dee-buhg]	User will say "Open/Close Debug" aloud.	Opens or closes the TouchCU debugging overlay.

6.3 Appendix C: Use Case Diagram

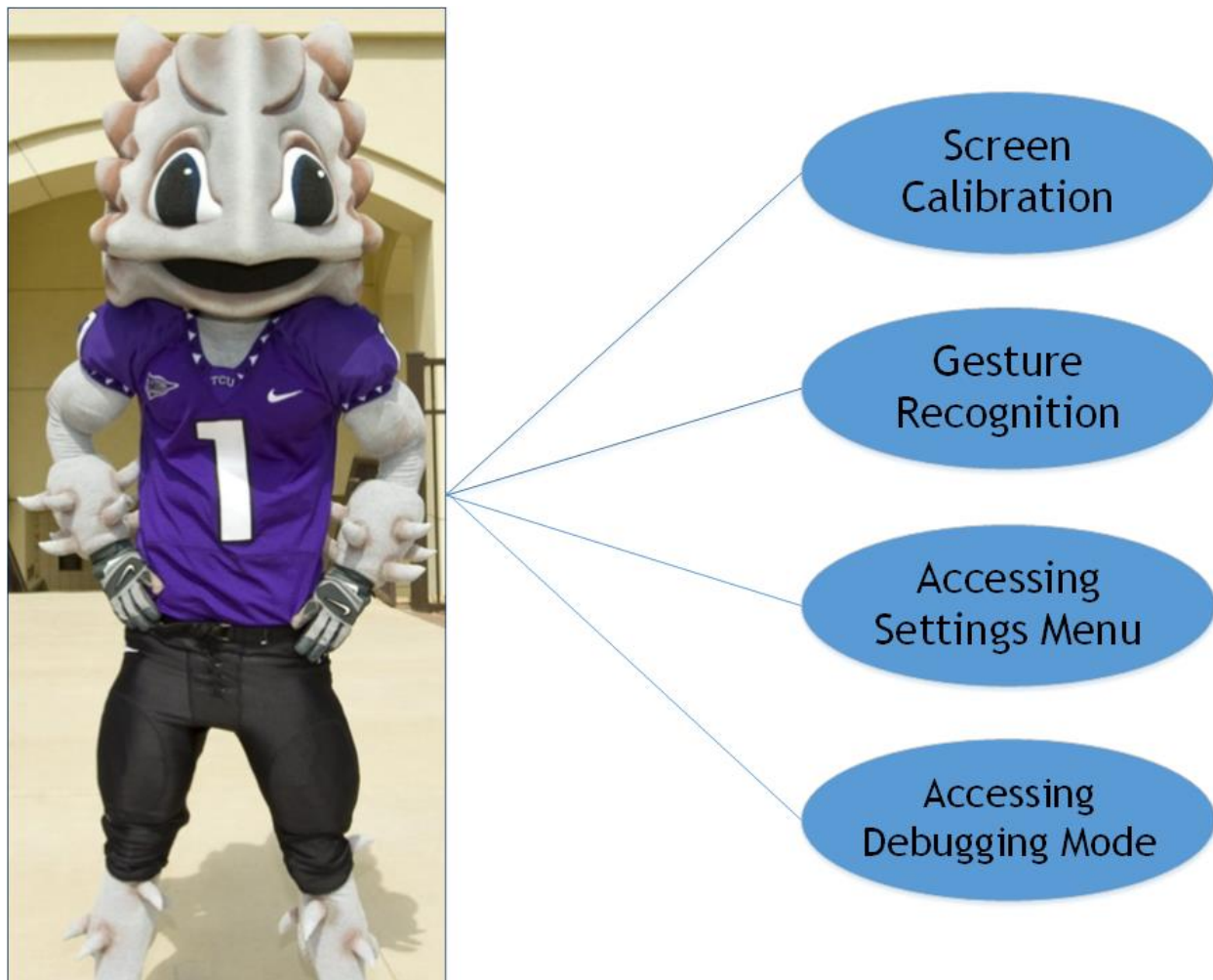


Figure 7

6.4 Appendix D: Use Case Scenarios

Screen Calibration	
Actors	User
General Goal	Determine the size and position of the screen on the surface that is being used and calculate proper screen coordinates.
Pre-conditions	<ul style="list-style-type: none"> • TouchCU application must be running. • One Microsoft Kinect for Windows is connected to the computer via USB and facing the surface that will be used. • Windows Desktop is projected onto the screen surface.
Triggers	The application is started on a computer for the first time or the user has decided to re-run the calibration wizard.
Course of Events	<ol style="list-style-type: none"> 1. Manual Calibration (Multi-point Calculation Method) <ul style="list-style-type: none"> • The application will prompt the user to put their right hand on each of the four screen corners. • Takes the coordinates and saves the data in a local file for future use. 2. The application will minimize and will begin to take input from the Kinect.
Alternate Paths	Application will use existing calibration settings.
Post-Conditions	Application moves onto Monitoring State.

Gesture Recognition (Monitoring State)	
Actors	User
General Goal	Recognize user movement and determine which gesture was performed.
Pre-conditions	Application is in the Monitoring State, successful post-calibration.
Triggers	The data stream input to the Microsoft Kinect for Windows contains movement from either one of the hands being tracked.
Course of Events	<ol style="list-style-type: none"> 1. Application determines if a gesture was received. 2. Application moves into active state. 3. Application generates the appropriate OS input command based on gesture. 4. Application sends input command to the OS.
Alternate Paths	If the hand does not generate a recognizable gesture, nothing will be done and application will go back to the Monitoring State.
Post-conditions	When one gesture is successfully processed and sent to the OS, the application will go back to the Monitoring State.

Accessing Settings Menu	
Actors	User
General Goal	Provide a screen within the application that shows the user a menu where they can select certain settings to change how the application runs.
Pre-conditions	System is up and running. Application is currently running and in either the Listening or Monitoring State.
Triggers	User will right-click the TouchCU icon from the System Tray and select settings or the user will say <i>"Addie + Open + Settings"</i> .
Course of Events	<p>The settings frame appear with the following checkboxes:</p> <ul style="list-style-type: none"> • Disable Air Gestures • Disable Voice Commands <p>The settings frame also allows the user to set the following smoothing parameters:</p> <ul style="list-style-type: none"> • Smoothing • Correction • Prediction • Jitter Radius • Max Deviation Radius • Distance to consider a touch • Hand Extension
Alternate Paths	The settings menu will not be present unless the icon is clicked on in the System Tray or the user says <i>"Addie + Open + Settings"</i> .
Post-conditions	Application will be running regularly.

Accessing Debugging Mode	
Actors	User
General Goal	Provide a screen within the application that shows the user detailed information on the input streams (IR, Depth, Color, Sound) the Kinect is receiving.
Pre-conditions	System is up and running. Application is currently running and in either the Listening or Monitoring State.
Triggers	User will select the option from the settings menu or the user will say "Addie + Open/Close + Debug".
Course of Events	The debug frame appear with the following: <ul style="list-style-type: none"> • Screen overlay showing current touch data • Coordinates of left and right hands <p><i>NOTE:</i> This does not prevent the application from running.</p>
Alternate Paths	The debugging screen will not be present unless the option is selected from the settings menu or the user says "Addie + Open/Close + Debug".
Post-conditions	Application will be running regularly with the debugging frame as an overlay.