D-BUG

Detailed Design Document

DUYGU ARALIOĞLU
BEDİA ACAR
ZÜLFÜ ULAŞ ŞAHİN
GÜLNUR NEVAL ERDEM

MIDDLE EAST TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
2011

Table of Contents

1. INTRODUCTION

This Detailed Design Document for TSL Kinect project sponsored by INNOVA provides the complete description of the system. This document also mostly follows the functionalities identified in the SRS document of the project, however some changes has been made in the structural arrangement of the system and all these changes were stated under the related subsections.

1.1. Problem Definition

In this project our aim is to solve the communication problem between speech-impaired people and others. When someone with speech impairment is unable to express himself/herself, it can be frustrating. Since almost most of the people do not know sign language and cannot understand what speechless people mean by their special language, tasks such as shopping, settling affairs at a government office are so difficult that speech-impaired people cannot handle by their own. Our team intends to help alleviate the frustration that speech-impaired people face by using assistive technology. This project proposes a method that translates sign language in a manner that other people can also understand. More precisely, the final product of the project will get the sign language gestures and give the meaning of them in text format.

1.2. Purpose

This SDD provides the design details of TSL Kinect in the scope of Middle East Technical University Computer Engineering Department Graduation Project and aims to provide a guide to a design that could be easily implemented by any designer reading this document. Throughout the document design architecture and procedure, constraints, interface design and implementation strategies will be explained in detail.

1.3. Scope

The scope of this SDD is to provide information about the design procedure of the system. The design constraints, data design, architectural design and user interface design will be elucidated in the scope of this document. Also chosen helper libraries and complete time planning of the system will be included. The intended audience is the ones who will implement the project, hence it is aimed that this document to be a guideline for those developers.

1.4. Overview

This SDD is divided into ten sections in order to provide a complete and understandable perception about the system to the target readers. First section is mostly about the scope and purpose of the document. This section also includes the definition of the problem that is intended to be solved.

In the second part, system overview, a general description of the software system including its functionality and matters related to the overall system and its design is provided. The intended goals, objectives and benefits of the project are stated in this section, too.

The third section states the design considerations and consists of two parts. In the first part, design assumptions, dependencies and constraints of the system are defined. In the second part, design goals and guidelines are given in terms of reliability, usability, portability and extensibility of the system.

The organization of the data structures of the system is explained in section four. Subsequently, a data dictionary is provided in order to provide a detailed description of the system major data, including data objects, their attributes and methods.

In the fifth section, the architectural design of the application and detailed description of modules are elaborated. In this section, it is also provided a sequence diagram for each module.

Sixth section is all about the user interface design. In this section the functionality and expected features of the user interface is given. In addition, some possible screenshots showing the interface from the user's perspective are provided and purpose of the screen objects is explained.

In the seven part of the document, the details of each component are explained deeply by following classification, definition, responsibilities, constraints, composition, interactions, resources, processing and exports issues.

In the eighth part, information about the libraries and tools that our project depended on is given.

In section nine, time planning and scheduling issues will be demonstrated by a Gantt chart.

The last part covers the summary of the whole document.

1.5. Definitions, Acronyms and Abbreviations

Definitions, acronyms and abbreviations are listed in the below table.

HMM	Hidden Markov Model
GUI	Graphical User Interface
WPF	Windows Presentation Foundation, is graphical subsystem based on XAML to develop user interfaces in Windows-based applications
TSL	Turkish Sign Language
SDD	Software Design Document
SDK	Software Development Kit
XAML	Extensible Application Markup Language

1.6. References

[1] IEEE Std 1016-1998: IEEE Recommended Practice for Software Design Descriptions

[2] Software Requirements Specification for TSL-Kinect, it was prepared according to IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications

[3] A Revealing Introduction to Hidden Markov Models
http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf

[4] A. B. S. Hussain, G. T. Toussaint, and R. W. Donaldson. Results obtained using a simple character recognition procedure on Munson's handprinted data. IEEE Transactions on Computers, 21:201–205, February 1972.

[5] J. Yang, Y. Xu, “Hidden Markov Model for Gesture Recognition”, The Robotics Institute, Carnegie Mellon University, pp. 10, 1994. Retrieved from http://www.ri.cmu.edu/pub_files/pub3/yang_jie_1994_1/yang_jie_1994_1.pdf

[6] Getting started with Microsoft Kinect SDK, http://www.i-programmer.info/programming/hardware/2623-getting-started-with-microsoft-kinect-sdk.html?start=1

[7] http://blogs.msdn.com/b/eternalcoding/archive/2011/08/20/10174036.aspx and (from readme of Kinect SDK)

[8] http://en.wikipedia.org/wiki/Kinect

[9] Coding4Fun Kinect Toolkit http://channel9.msdn.com/coding4fun/projects/Coding4Fun-Kinect-Toolkit

2. SYSTEM OVERVIEW

Our software needs a Kinect camera and a personal computer with Windows 7 operating system to work. Kinect camera gathers real world images with depth information and transfers it to computer via USB. Microsoft Kinect SDK gathers these images, processes them and detects skeletal-position information of the user who stands in front of the camera. Software uses position information of the body parts for both interface control and gesture recognition.

3. DESIGN CONSIDERATIONS

3.1. Design Assumptions, Dependencies and Constraints

3.1.1. Assumptions and Dependencies

While designing this project, we needed to make some assumptions related to software, hardware and the environment. First of all, our program is intented to run on Windows 7 OS. Working platform cannot be changed since the Microsoft official SDK for Kinect has such a big restriction that it can be used only on Windows 7 platform.

Related to hardware, our program will run on computer systems, so the user is expected to have a computer with dual-core,2.66-GHZ or faster processor and minimum of 2 GB of RAM . Moreover, the graphic card support in Windows for Direct X 9.0c is also expected. The most important requirement as a hardware is the user must have Kinect for Xbox 360 sensors.

Related to environment, since our program is completely about skeleton tracking, the environment that the user stands should be clear as possible in order to have a properly working SDK and maintain accuracy. We assume that Kinect is set up properly according to its manual and the user stands between 1.8 to 2.4 meters away from the Kinect sensors. In our environment, it is also assumed the room that the Kinect is set up is large enough and there are no moving objects around.

There are no restricted features about the user, however; since our system tracks only one person’s skeleton user should stand alone in front of the sensors.

3.1.2. Design Constraints

3.1.2.1. Time Constraints

Both the design and implementation parts of the project should be completed within eight months. At the end of the fall semester of 2011-2012 academic year a prototype needs to be implemented, so in order to achieve this all group members should strictly follow the specified schedule. The detailed schedule can be found in the Gantt charts in the section 8 of this detailed design report.

3.1.2.2. Performance Constraints

The system performance strongly depends on the environment that the Kinect is set up. When the environment is so noisy or there are moving objects around the recognition of the skeleton cannot be done as desired by SDK. Moreover, the system should recognize the gesture as fast as possible and return an answer to the user. In order to satisfy these constraints, the process running on the background of the application needs to be modeled with the fastest algorithm as possible.

3.1.2.3. Financial Constraints

Project is financially supported by Innova Bilişim Çözümleri A.Ş in terms of Kinect sensors.

3.2. Design Goals and Guidelines

3.2.1. Reliability

Our main purpose is to make the system run without any problems or bugs. In order to maintain the reliability any performed gesture will be correctly recognized by the implemented HMM and correct answer will be published if the gesture is in the pre-determined gesture list. We will use various testing strategies while designing the

project in order to improve the performance and decrease the number of errors that will occur.

3.2.2. Usability

Since this project intends to simplify speech-impaired people's lives it should be simple and functional at the same time. The user interface is kept simple so that the pure functionality is gathered without being bothered with lots of menus or buttons. It should also be stated that not only the sign language translation part but also switching between menus and sub-menus are maintained by the user's movements. In other words kinetic interface is implemented hence the user does not have to use any input device (namely mouse and keyboard) to control the program once it is started.

3.2.3. Portability

The system will be implemented only into the pc environments having Windows7 operating systems which makes the system low-portable.

3.2.4. Extensibility

This project is designed to be extensible in terms of the number of gestures to be recognized. For the time being the system will recognize 10 pre-defined TSL gestures, however, it will be possible to add new gestures to the programs' memory by gathering experimental data of gestures and training new HMMs.

4. DATA DESIGN

4.1. Data Description

Every major module and sub-components in the system will be implemented as classes. Our software basically performs a "function" (sign language recognition), and both main

and sub-components will be created to compose a functional program. There will be no database system because the numbers of the components are pre-determined, their creation will be handled in initialization, their names and duties are restricted and there will be no structured information transfer between classes. Information transfers will be in simple vectors of primitive types. Also, predetermined gestures will be stored as Hidden Markov Model matrices in a simple text file, and transfer of this file to main memory will be held in a simple initialization function. So, we decided to remove Database module that we defined in the previous requirements specification document.

There will be four main classes of our system: InputHandler, Recognizer, InterfaceHandler, and HMM. First three of them will have only one instance, and functional structure of the system is built by them. For every sign language gesture that introduced to the system, there will be one or two HMMs (this depends on gesture type) in the Recognizer module. HMMs are also initialized at the beginning of the program and never be released.

There will be three sub-classes of our system: InterfaceMode, SpaceZoningMatrix, and Button. Button class will contain information about buttons of InterfaceModes, so this classes will be elements of InterfaceModes at the beginning. InterfaceModes will contain specific buttons of all user interface menus of the software (there are four of them), and they will also be created by InterfaceHandler at the beginning of the application. SpaceZoningMatrices will be created and deleted every time when InputHandler repeated its functional routine.

In the section 4.2, a short explanation of attributes and helper methods of each class will be provided. The fundamental functional structures of main components, and the way of assisting attributes, helper methods and subclass objects in these structures will be explained together with algorithmic details in section 5.

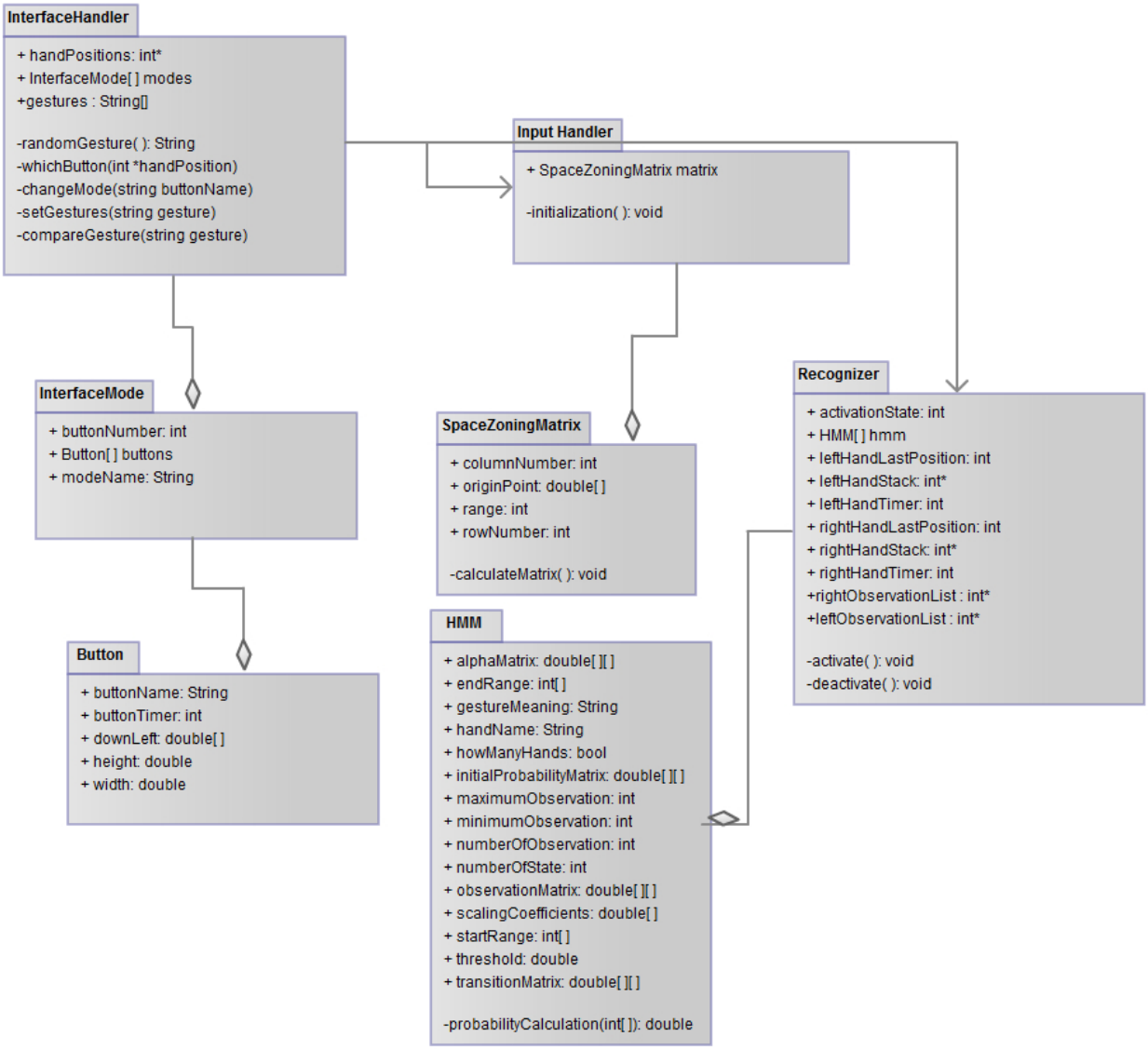


Figure 1: Class Diagram of the System

4.2. Data Dictionary

InputHandler:

Attributes:

- matrix (spaceZoningMatrix): This object will be created and initialized with every new input comes to InputHandler.

Methods:

- initialization(): void — This method initializes all other components and creates pre-defined class instances.

Recognizer:

Attributes:

- activationState (boolean): This variable determines whether the Recognizer is active or not.
- hmm (HMM[]): This array holds all HMMs in the system.
- leftHandLastPosition (int): This variable contains the last position of left hand that comes from InputHandler.
- leftHandStack (int*): This stack holds different consequent positions of left hand.
- leftHandTimer (int): If left hand does not change its position, this variable increments.
- leftObservationList(int *): the list that keeps observation sequence for left hand which will be sent to HMM

- rightObservationList(int *): the list that keeps observation sequence for right hand which will be sent to HMM
- rightHandLastPosition (int): This variable contains the last position of right hand that comes from InputHandler.
- rightHandStack (int*): This stack holds different consequent positions of right hand.
- rightHandTimer (int): If right hand does not change its position, this variable increments.

Methods:

- activate():void This method sets activationState true.
- deactivate():void This method sets activationState false.

HMM:

Attributes:

- alphaMatrix (double[][]): This matrix contains scaled alpha values of each state calculated at each pass.
- endRange (double[]): This array contains possible end positions of the gesture as SpaceZoningMatrix indices
- gestureMeaning (string): Holds the corresponding meaning of the gesture.
- handName (string): This variable determines which hand's movement will be recognized by HMM (left or right).
- howManyHands (boolean): This variable tells if related gesture is composed of one or both hands movements.

- initialProbabilityMatrix (double[][]): This matrix contains probabilities which determine the initial state.
- maximumObservation (int): This variable determines the maximum number of matrix indices that gesture can be represented by.
- minimumObservation (int): This variable determines the minimum number of matrix indices that gesture can be represented by.
- observationMatrix (double[][]): This matrix matches each states and each discrete observations with probability values.
- observationNumber (int): This variable holds the number of observations in the model.
- scalingCoefficients (double[]): This array contains scalars calculated for each pass of forward algorithm.
- startRange (int[]): This array contains possible start positions of the gesture as SpaceZoningMatrix indices
- stateNumber (int): This variable holds the number of states in the model.
- threshold (double): This value determines minimum occurrence probability of the gesture that HMM recognizes.
- transitionMatrix (double[][]): This matrix contains probabilities of state transitions. Also known as A matrix.

Methods:

- probabilityCalculation(int []):double This method calculates occurrence probability of given observation list.

InterfaceHandler:

Attributes:

- handPositions(int *): This holds the positions of both hands

- InterfaceMode[] modes: The array holding the instances of the InterafceMode object
- gestureNames (string[]): This array holds gesture meanings for randomGesture function.
- educationCounter (int): Counts execution cycles in order to detect seconds for education mode of program.
- gestures(string[]): This array holds the last 4 gesture meanings coming from Recognizer to illustrate user.

Methods:

- randomGesture(): String Randomly chooses a gesture name from gestureNames array.
- displayVideo(video *stream): void This method displays the image of the user on the screen
- changeMode(string buttonName):InterfaceMode This method changes the mode to “Eğitim” or “İletişim” mode.
- setGestures(string gesture): void This method sets the meaning of the new gesture name it the communication mode .
- compareGesture(string gesture):void This method compares the string it gets with the expected gesture’s name in education mode.
- whichButton(int *handPosition):button This method returns the button that the hand is on

SpaceZoningMatrix:

Attributes:

- columnNumber (int): This variable contains how many column that matrix has.

- originPoint (double[]): This array holds the x and y coordinates of the origin of matrix.
- range (int):This variable contains length of one square of matrix.
- rowNumber (int): This variable contains how many row that matrix has.

Methods:

- calculateMatrix():void Calculates data members of this class.

InterfaceMode:

Attributes:

- buttonNumber (int): This variable is the number of buttons in InterfaceMode.
- Button[] buttons: This is the array holding the buttons present in an interface mode
- modeName (string): Name of the interface mode

Button:

Attributes:

- downLeft (double[]): This array holds x and y coordinates of a button with respect to the upper left corner of the screen box.
- width (double): This variable holds the width of a button.
- height (double): This variable holds the height of a button.
- buttonTimer (int): This variable holds a counter determines the number of the cycles that a hand cursor is on this button.
- buttonName (string): hols the names of buttons

5. SYSTEM ARCHITECTURE

5.1. Architectural Design

System will consist of four major components. These components designed as functional units and will be implemented as classes. InputHandler can be thought as the core component of the system. It will operate while program is on, and it repeatedly arranges Kinect input that SDK serves to the system. All the other components will use these inputs and performs their responsibilities with the rhythm of InputHandler's repetitive functional routine. HMMs are responsible for detecting pre-determined gestures. Each gesture has its own HMM, and the part of the system that manages HMM modules, their inputs and decides whether an input sequence corresponds to a pre-defined gesture meaning is called Recognizer. InterfaceHandler is the module that arranges menus, buttons, video displays, and also handles functional requirements of each mode of the user interface. Functionalities of education and communication mode will be provided by InterfaceHandler.

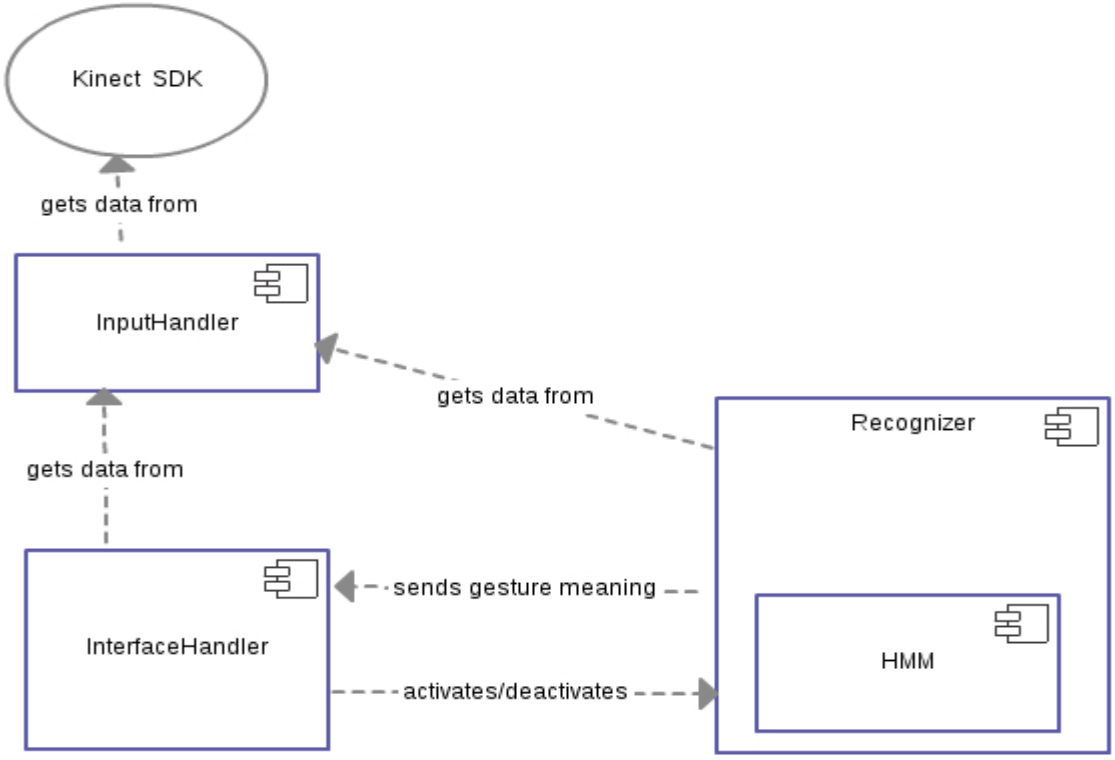


Figure 2 : Component Diagram of the System

5.2. Description of Components

5.2.1. InputHandler

This component basically manipulates Kinect inputs and serves them to other components.

5.2.1.1. Processing narrative for InputHandler

InputHandler component is the core component of the software. It processes Kinect inputs that SDK supplies to the system, and sends them to related components while program is on.

Two kinds of input manipulation are held by this module. First, it calculates hand positions ,scaled with respect to the size of the screen box, for interface controlling via gestures (explained in 5.2.1.3). Second, it matches hand positions to a number in the space zoning matrix.

5.2.1.2. InputHandler interface description

This module takes video stream and skeleton joints information generated by SDK as input. It sends scaled hand positions and video stream to InterfaceHandler component, and the matrix indices of both hands to Recognizer component.

5.2.1.3. InputHandler processing detail

This component repeats its functional routines every time a new input is generated by SDK (30 times per second).

First, it scales x- y coordinates of the hands with the size of screen box, a rectangle on user interface. This calculation is transformations of 3d positions of both hands into 2d positions, and the results will be used to move hand cursors which will be explained in section 5.2.4.

Second, InputHandler creates an abstract matrix called SpaceZoningMatrix. It is used for detecting the positions of the hands with respect to its indice labels, and those indice labels will be sent to Recognizer. Every time SDK provides a raw input of skeleton joints, a new SpaceZoning Matrix is created and hand positions are reduced to single integer value.

SpaceZoningMatrix is created by calculating the length between the user's hip and shoulder centers. This length is divided to a pre-determined integer and result is stored in a variable called range. Then, hip center is considered to be lying on the down-middle of the matrix, and originPoint is determined by taking "*a* times range left, *b* times range up" from hip center. Origin is the upper-left corner of the matrix. Matrix squares are labeled with integers from left to right and then up to down (for example, squares of 4x5 matrix will be labeled from upper-left to bottom-right as 1,2,…,19,20 with this method). Then , the squares that hands lie on are computed and their labels are sent to Recognizer as single integer values for each hand. Origin, range, a, b, x values and matrix square labels can be seen in the *Figure 3*.

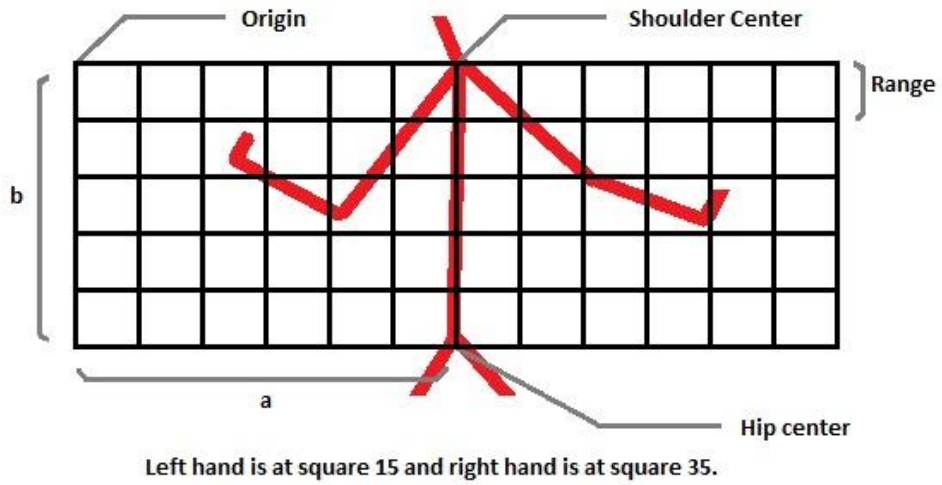


Figure 3: Space zoning matrix

Exact values of these variables will be determined experimentally in the implementation phase.

5.2.1.4. Dynamic behavior InputHandler

InputHandler has interaction with Kinect SDK, InterfaceHandler and Recognizer. It takes raw input from SDK whenever it is ready, processes some of these raw inputs and passes them to both InterfaceHandler and Recognizer.

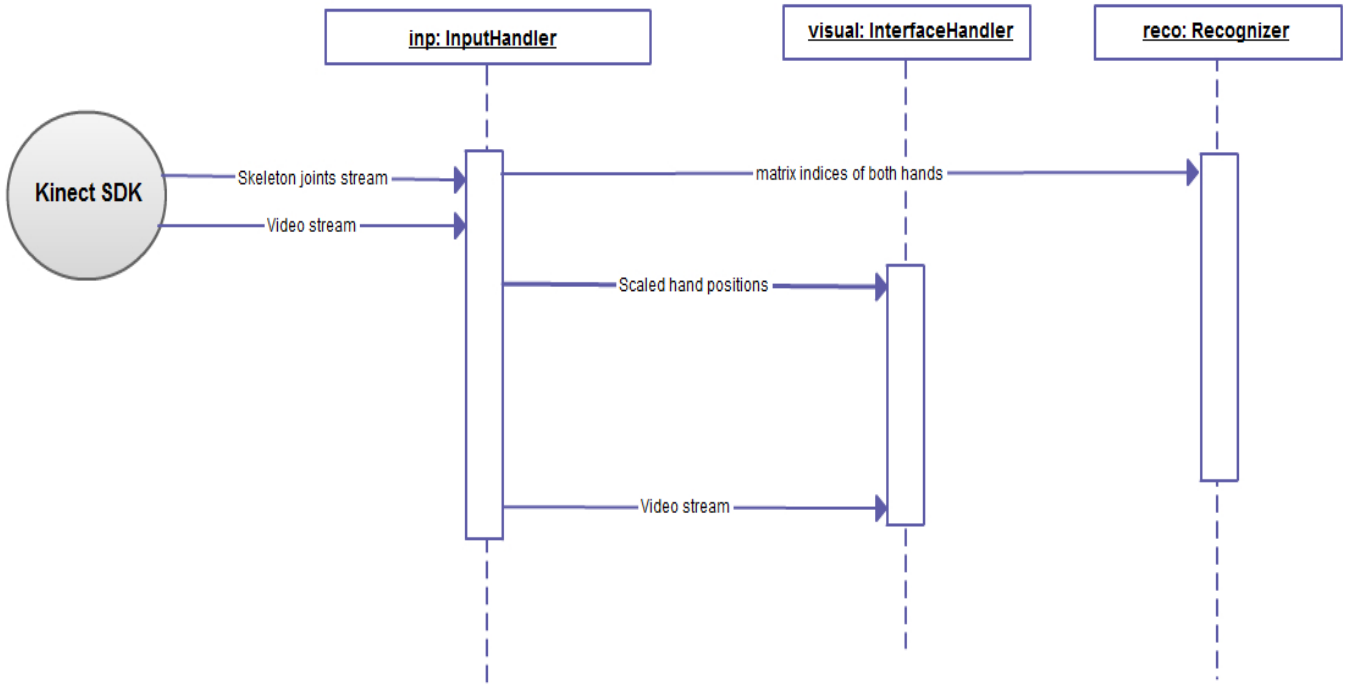


Figure 4: Sequence Diagram of InputHandler

5.2.2. Recognizer

This component essentially performs gesture recognition.

5.2.2.1. Processing narrative for Recognizer

Recognizer will use HMMs to detect performed gesture; also it will arrange, store and deallocate observation stacks. It will try to match collected observations with pre-determined gesture meanings by using corresponding HMMs which are created for each gesture.

5.2.2.2. Recognizer interface description

Module takes position values of both hands from InputHandler as input. Its activation is triggered by InterfaceHandler. It gives the meaning of the related gesture to InterfaceHandler as output.

5.2.2.3. Recognizer processing detail

While Recognizer module is in active state, it stores observation inputs of both hands that come from InputHandler on related hand stacks. So, one of the major tasks of this module is eliminating inputs and releasing these right/leftHandStacks whenever it is necessary.

This module takes two integer values for each hand, and compares them with the previous ones. If they are not different, they will not be inserted into right /leftHandStacks. Same successive positions that come from InputHandler can imply that hand is standing firm. There are two counters for each hands detecting the cycle numbers of the same values come from InputHandler. If a counter reaches a pre-determined upper-limit, it means that hand is not moving, so related right /leftHandStack will be released to prevent unnecessary memory allocation.

If a new position value comes from InputHandler to this module, then it will be put into related hand stack as first element, and Recognizer checks first *n* elements of related hand stack with each HMMs assigned to this hand. The value of *n* changes with respect to HMM (every number between max. and min. observation number of HMMs are checked). When Recognizer checks an HMM for first *n* element, it first compares first element of the stack with HMM's endRange, and *n*.th element of the stack with HMM's startRange. If both values are matched, then HMM module calculates logarithmic probabilities for these *n* observations. If the result is bigger than the threshold value of HMM, it means that recorded movement is the gesture to which HMM is assigned. Then, hand stack that contains this *n* position values will be released.

If a gesture performed with two hands, Recognizer checks both hand stacks in this way. After this process, if recognition does not occur, Recognizer keeps inserting new position values into hand stack.

5.2.2.4. Dynamic behavior Recognizer

When Recognizer is activated by InterfaceHandler, it starts to perform gesture recognition via HMM sub-components and transfers the result to the InterfaceHandler until it is deactivated.

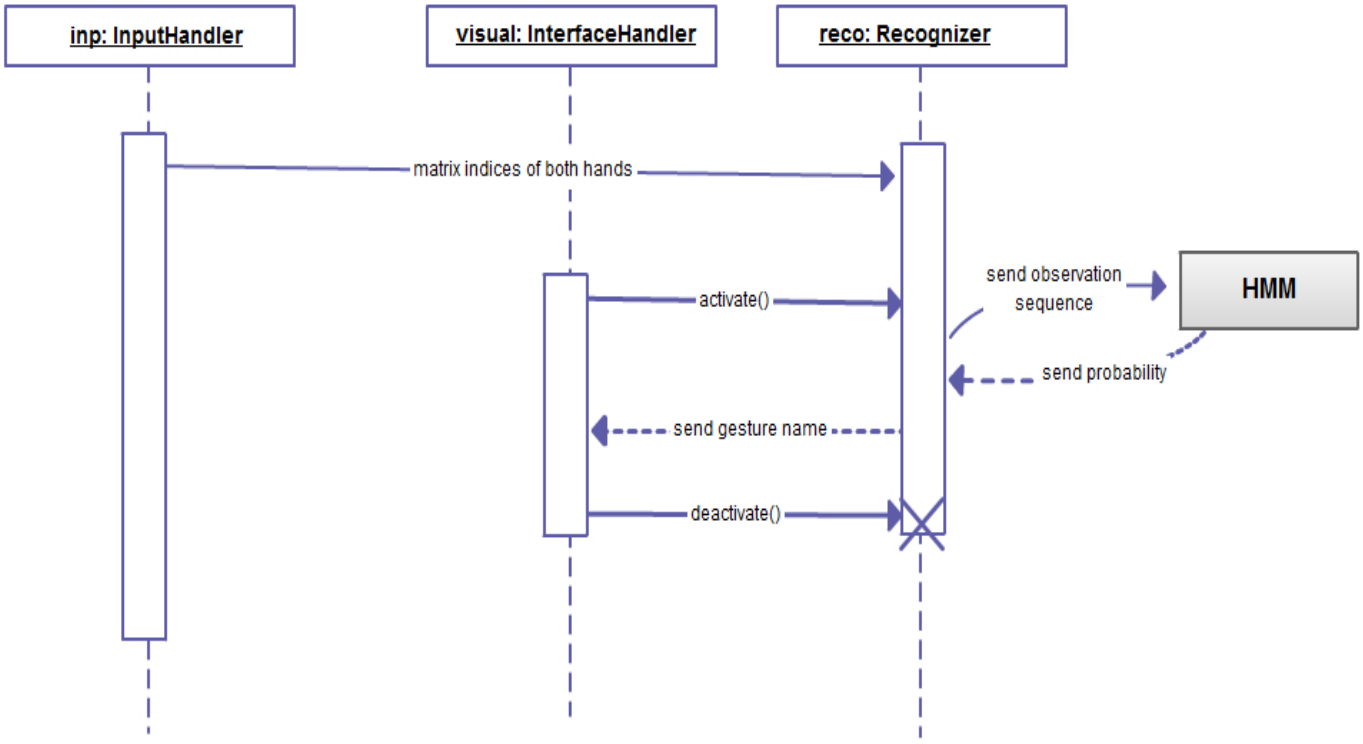


Figure 5: Sequence Diagram of Recognizer

5.2.3. HMM

HMMs are not independent components, they are sub-components of Recognizer module. On the other hand, instances of this class play an important role in gesture recognition process. They can be considered as main functional units of the software just like others mentioned in this section. Before giving information about the processing narratives or interface description of this module, one should understand what HMM stands for. Although their functional characteristics are very sophisticated

and worth to be explained in detail, we will provide a brief explanation about HMM to let the reader have an understanding about it.

HMM (Hidden Markov Model) is a probabilistic model created with a state set and an observation set. Model is based on the assumption that a visible observation sequence (in our software, patterns will be represented with observation sequences) is triggered by unobservable (hidden) states of the model. Model is defined by a transition matrix (a matrix that shows probability of each state transitions), an initial distribution matrix (a matrix that shows starting probabilities of each state) and an observation matrix (a matrix that matches hidden states and visible observations with a probability value).

If a model is given, an observation sequence's occurrence probability for this model can be calculated with forward algorithm. Forward algorithm recursively calculates occurrence probabilities of first *t* elements of an observation sequence for each state by using probabilities of first *t-1* elements. These values called *alpha* value of state *i* at time *t*, and they are collected to calculate next *alpha* values until *t* reaches to the end of the sequence. When *t* is equal to the length of the sequence, the observation sequence's occurrence probability with each state is calculated.

Also, a model can be trained with an observation sequence set by repeatedly re-estimating its matrix values. Training aims to maximize occurrence probability of a specific kind of observation sequence. If a model is trained with parallel observation sequence set, then it will give higher probability to the sequences that has same/alike properties with the ones in training set.

In our system, one or two HMMs will be trained for each gesture. Minimum probability that HMM calculates for its training sequences will be stored as its thresholds. Also, minimum and maximum number of observations that will be used for training HMMs will be held in this unit. There will be more explanations about HMMs, their kinds and

topologies, calculating and training principles, formulas and algorithms in Detailed Design Report.

5.2.3.1. Processing narrative for HMM

These functional units will be employed by Recognizer. It will make calculations for an input sequence in order to determine whether the given sequence matches with the training gesture of HMM. It will also hold gesture related information, such as gestures minimum occurrence probability in this model, its maximum, minimum length as a sequence, and its possible starting and ending points.

5.2.3.2. HMM interface description

HMMs take observation sequences from Recognizer as input. It gives gesture related information and calculated occurrence probability of a sequence with respect to the model as output to Recognizer module.

5.2.3.3. HMM processing detail

When an HMM takes an observation sequence, it uses forward algorithm to calculate probability. During execution of algorithm, it computes *alpha* values and scales them with the sum of *alpha* values which belong to same pass. Scaled *alpha* values for each state in each pass are collected in alphaMatrix, and scalars of each pass are collected in scalingCoefficients array. After passes reach to the end, logarithmic sum of scalars of each pass will be equal to the logarithm of actual occurrence probability of the set.

5.2.3.4. Dynamic behavior HMM

When Recognizer module is active, it takes pre-determined values of an HMM, then if it is necessary, it passes portions of related hand stacks to HMM. An HMM will perform its calculation and both the result of this calculation and HMM's threshold value will be passed to Recognizer.

As it is stated before, HMMs are actually sub-components of Recognizer module. Since an HMM performs its calculation only when Recognizer requests, and it communicates only with Recognizer, it is decided not to put a special sequential diagram for HMM.

5.2.4. InterfaceHandler

InterfaceHandler is responsible for the arrangement of the user interface. It also changes interface modes according to the scaled hand positions information that comes from InputHandler.

5.2.4.1. Processing narrative for InterfaceHandler

InterfaceHandler manages the user interface, moves hand cursors (they are used as mouse pointers) and changes interface modes with respect to hand movements of the user, and also implements user interface mode functionalities. There will be four user interface mode: main menu, tutorial, education and communication mode. For each of them, InterfaceHandler creates an InterfaceMode class instance at the beginning of the program. Each InterfaceMode object will contain different buttons and some of them will have textboxes.

5.2.4.2. InterfaceHandler interface description

InterfaceHandler takes video stream and scaled hand position vectors from InputHandler. While communication or education mode is active, it takes the string from Recognizer that shows the meaning of the performed gesture. It activates and deactivates Recognizer component whenever interface mode changes.

5.2.4.3. InterfaceHandler processing detail

No matter which menu is active, back stage of user interface remains unchanged. InterfaceHandler takes video stream from InputHandler, and uses library functions to render and display video on back stage. Also, this module uses another library function to move hand cursors to new positions on the screen box in each cycle.

There are four InterfaceMode objects in this component. These modes will be activated by hand cursors. Buttons of the program is actually not real buttons (i.e., mouse event handler will not be assigned to them) , they are represented by coordinates of a rectangle in screen box and an icon. If one of the hand cursor stands on the rectangle of a button, InterfaceHandler increments counter of the button, and if this counter reaches to an upper-limit, then current mode is deactivated and related user interface mode of the button will be activated.

When a new interface mode is activated, related textboxes and button images will be shown on front stage. All this modes have different functional properties, and InterfaceHandler implements these functionalities.

If main menu or tutorial is opened, program will not perform gesture recognition. So, InterfaceHandler deactivates Recognizer module and displays the related buttons on front stage. There is not a special function of these two user interface modes.

If the communication mode is on, InterfaceHandler activates Recognizer. Every time recognizer detects a gesture, it passes its name to InterfaceHandler. InterfaceHandler will put these strings in an array, and content of this string array will be displayed in the textbox.

If education mode is activated, a back button and a textbox that shows requested gesture's name and time left will be displayed on front stage. randomGesture function chooses a random gesture name in every ten seconds, and waits user to perform the chosen gesture. Ten seconds will be decremented by the help of educationCounter variable, this counter will be incremented every time that InterfaceHandler main routine is executed. When educationCounter reaches an upper limit, the timer on the screen will be decremented. If user performs correct gesture, Recognizer will transfer the expected gesture name and InterfaceHandler stops counting down process. Textbox shows a success message for a while and another gesture will be selected randomly to repeat this process. If user cannot perform expected gesture on time, a failure message appears on text box for a while and another gesture will be selected randomly to repeat this process.

5.2.4.4. Dynamic behavior InterfaceHandler

InterfaceHandler module is cooperating with both InputHandler and Recognizer. Activation and deactivation of Recognizer depends only on this module.

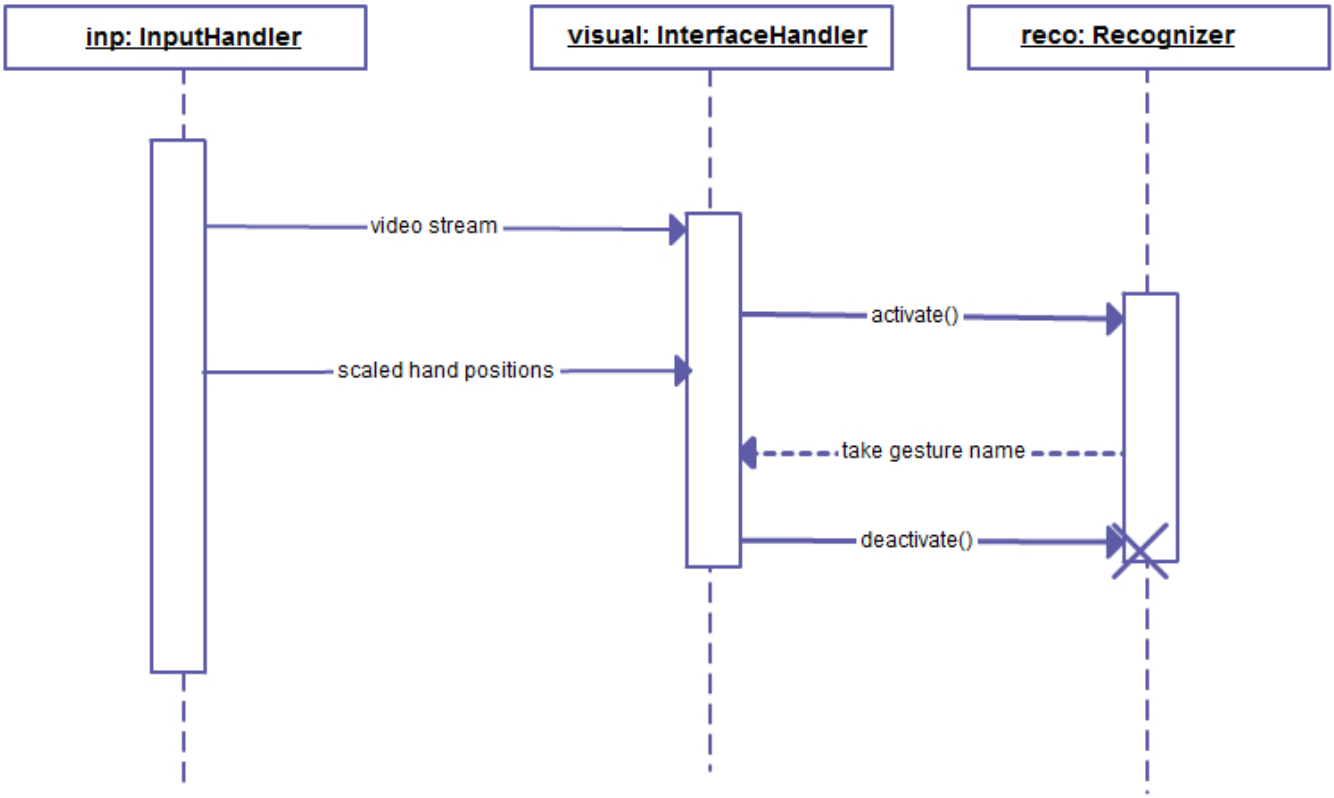


Figure 6: Sequence Diagram of InterfaceHandler

5.3. Design Rationale

We decided to conduct our system operations with four main components, and all these parts have a crucial role. Since both recognizing function and interface needs processed inputs, the best decision is to assign a module to handle Kinect SDK's raw inputs and provide processed inputs to other modules. An alternative method could be making every module to handle Kinect SDK inputs by themselves. But as mentioned before, recognizing algorithm needs discrete observations as inputs, and putting these complex calculations into Recognizer module will complicate its functional operation unnecessarily. While taking Kinect SDK inputs via a time dependent event handler, InputHandler also determines the rhythm of the system and enforces other modules to

execute their routines repeatedly. There should be a component that takes this role for the system even if there is no InputHandler.

Recognizer module recognizes sign language gestures by using HMMs. Since sign language recognition is the main purpose of the system, an independent module is assigned for this function. The most important advantage of collecting functional units of recognition process into one module is that: communication between InterfaceHandler and this module can be arranged in a simple, elegant way. Recognizer module can be activated and deactivated by one command and it transfers recognized gesture into a simple output.

InterfaceHandler contains all user interface related functionalities. In SRS document, we assigned independent modules for education and communication modes. But these user interface modes need very little functionality except arranging interface changes. So, we decided to remove them and assign their functionality to InterfaceHandler. With this arrangement, all interface related functions of the system are collected and user interface mode information is wrapped with InterfaceMode structures.

5.4. Traceability of Requirements

In this section, we will trace our design implementation to our software requirement specification. The aim is to ensure that, we have kept the design simple enough not to implement additional functionality that users did not wanted . On the other hand, we have added new sub-components, data structures and eliminate some of the functionalities explained in the previous software requirements specification document. Since the software that we develop is mostly an interface based one, in the requirements specification document mostly the functional requirements of user interface are mentioned. Because of this property, most of the functional requirements are related to the InterfaceHandler module.

Below, the requirements versus modules in a tabular form to associate the requirements with modules to show which module is responsible for which requirement.

SRS REQUIREMENTS	SDD MODULES
3.2.1.	5.2.2.,5.2.4.
3.2.2.	5.2.2.,5.2.4.
3.2.3.	5.2.4.

6. USER INTERFACE DESIGN

6.1. Overview of User Interface

This project can serve both speech-disordered people and the others. Yet it is designed especially for the speech-disordered people's convenient use. Since most of the speech-disordered people are not capable of hearing neither any sound-depended input-output is included. All the functionalities of the program is maintained with gestures and movements performed by the user. More precisely, clicking any of the buttons related to an action is done by the user putting any hand on the related button for a certain amount of time.

In this program there are four main menus: main menu, tutorial menu, communication menu and education menu. In all menus the user can see the video image of himself/herself. The hand positions of the user are shown with hand images on the screen. The user sees the positions of these images and can select the buttons accordingly. Also it helps the user during the communication and education modes.

In the main menu the user can switch to education or communication mode. Also it is possible that the user switch to the tutorial menu form this menu. The exit from the program is possible only from this menu.

Tutorial menu is a simple text based menu that the user can read how to use the program. The user can go only to the main menu from this menu.

Communication menu is where the conversion from Turkish Sign Language to text is done. The user performs the gestures in front of the Kinect and the program tries to recognize the gestures. The gestures are performed one after another and as the pre-defined gestures are recognized they are displayed on the screen. If the movements do not correspond to any gesture nothing is displayed. The user can go only to the main menu from this menu.

The education menu is for the user to practice Turkish Sign Language. It is like a game asking the user to perform gestures that is displayed in a text box on the screen. Every gesture should be done in a certain amount of time and the user can see the remaining time. After the specified time elapses the user is informed with a message appearing on a text box which informs the user whether s/he has performed the gesture correctly or not. The user can go only to the main menu from this menu.

6.2. Screen Images

Figure 7: ScreenShot of Main Menu

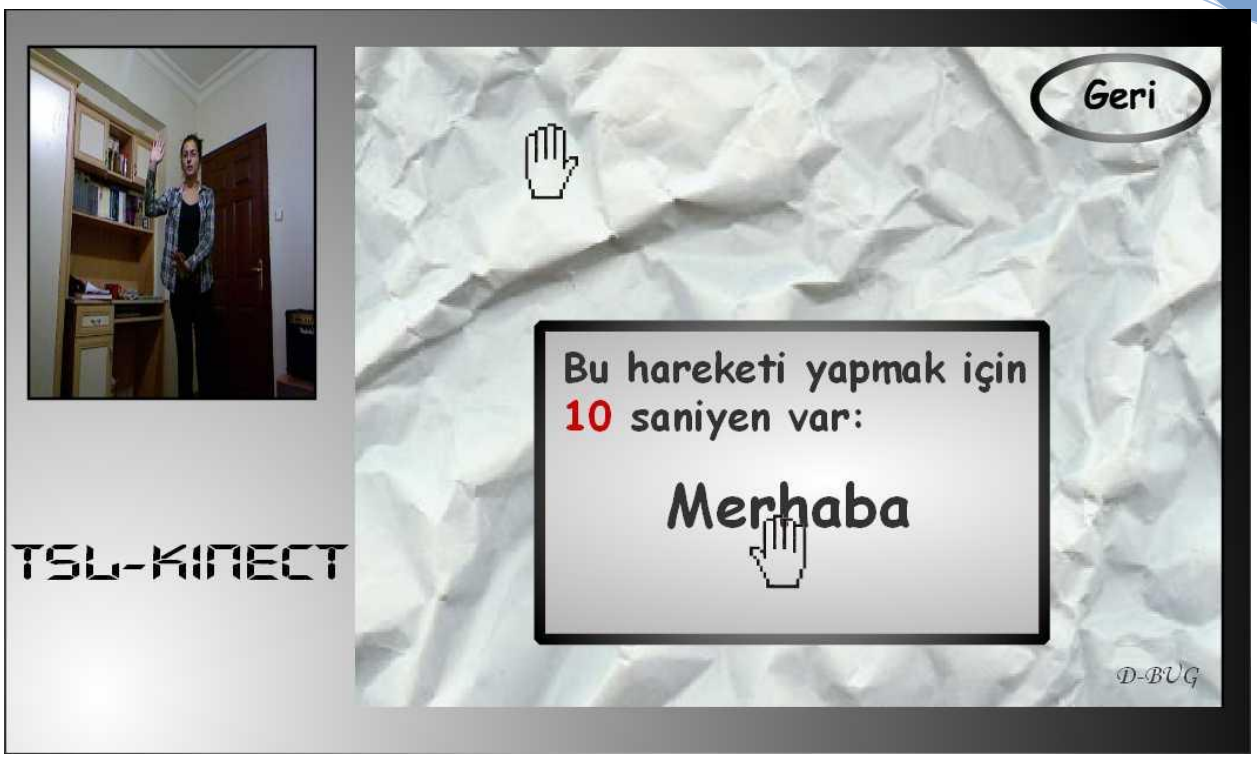


Figure 8: ScreenShot of Education Menu

Figure 9: ScreenShot of Communication Menu

6.3. Screen Objects and Actions

Since there are only four menus and the program interface is kept simple there are not many interface components. The general structure of the user interface is as follows:

- There exists a video box showing the image of the user simultaneously. This helps the user see herself/himself whether s/he is in the line of sight of the Kinect.
- There is the screen box on the right half of the program interface which consists of different buttons, or text boxes according to the active mode of the program.
- The hand positions of the user is shown with hand images on the screen box.

The components of the screen box are dependent on the active mode. In this program there are four menus: main menu, tutorial menu, communication menu and education menu.

In the main menu it is possible to choose either of the communication or education modes of the program by selecting the corresponding buttons. “Nasıl Kullanılır?” button is for the tutorial menu to appear. The “Çıkış” button is to exit from the application.

The tutorial menu has a text box in which the helping text is written and a button labeled “Geri” for switching back to main menu.

In the education menu there is a text box which the meaning of gestures displayed also the remaining time is displayed in this text box. The information about the trueness of the gesture performed is given in a text box again. There also exists a “Geri” button to switch back to main menu.

In the communication mode the recognized gestures’ meanings are displayed in a text box where the last word is at the top and emphasized. There also exists a “Geri” button to switch back to main menu.

7. DETAILED DESIGN

7.1. InputHandler Component

7.1.1. Classification

This component is a class with attributes and methods.

7.1.2. Definition

InputHandler is responsible for the input manipulation that comes from Kinect. This component is in interaction with both Recognizer and InterfaceHandler.

7.1.3. Responsibilities

The main role of the InputHandler component is getting data from Kinect. InputHandler will be on process as long as the application is working, in other words, it serves the manipulated input 30 times per second to InterfaceHandler and Recognizer. Firstly, it calculates hand positions, scaled with respect to the size of the screen box for the controlling user interface with gestures feature of the application. Also the video stream coming from Kinect is to be posted to InterfaceHandler by InputHandler. Secondly, Recognizer receives the hand positions as a number in the SpaceZoningMatrix from the InputHandler. Furthermore, another significant responsibility of this module is to initialize all other components at the beginning of the application.

7.1.4. Constraints

There are some constraints about the environment that the user stands. For this component to do its task properly, Kinect should be set correctly. As a requirement of this, the environment that the user stands should be as clear as possible, otherwise Kinect cannot process the input. The room should be large enough and there should not be any moving objects.The application can recognize only one person's skeleton joints, thus there

should not be another person around. Also the distance between the user and the Kinect is another constraint that Kinect requires. This distance should be between 1.8 to 2.4 meters. The user should stand straight since the distance between the hip center and the shoulder center is used to produce the SpaceZoningMatrix. If the user bends forward the distance between hip center and shoulder center will decrease and relative miscalculations occur. It is also same for bending backward, right or left.

7.1.5. Compositions

There is only one sub-component of the InputHandler which is SpaceZoningMatrix class. Although the existence of SpaceZoningMatrix class was not obligatory, it is integrated into the system since the matrix properties such as the height and weight of the matrix are determined experimentally and so if there needs to be any changes in these attributes it will be done easily.

7.1.6. Uses/Interactions

The two modules that the InputHandler interacts are the Recognizer and InterfaceHandler. As long as the application is active the InterfaceHandler is served by the InputHandler in order to manage the user interface with gestures. While the Recognizer module is active, InputHandler sends the hand positions to this module.

7.1.7. Resources

For this component the main external resource is obviously Kinect and the input it serves. Also, the libraries used for this component NUI and Coding4Fun should be stated. The positions of the skeleton joints of the user are given by the help of NUI library. The video stream, which is accessed by the help of Coding4Fun library, is another resource for this component although it directly sends this input to InterfaceHandler. Coding4Fun is also used to obtain the hand positions scaled with respect to the screen.

7.1.8. Processing

The first task that InputHandler accomplishes is to determine the hand positions on the screen, in other words, the transformations of 3d positions of both hands into 2d positions on the screen. The results will be used to represent hands on the screen box which is a rectangle on the user interface. To do this, it scales x- y coordinates of the hands with the size of screen box. Then it passes these computed values to InterfaceHandler module.

Another production of this module is the SpaceZoningMatrix. This is an abstract matrix created every time SDK provides the input of skeleton joints positions. By the help of this matrix hand positions are reduced to single integer value that is to be sent to Recognizer. For this, module calculates the length between the user's hip and shoulder centers. This length is divided to a pre-determined integer and result is stored in a variable called range. It is assumed that hip center is considered to be lying on the down-middle of the matrix. Assuming so the originPoint , which is the upper-left corner of the matrix, is determined by taking "a times range left, b times range up" from hip center. The indices of the matrix is determined with integers from left to right and then up to down (for example, squares of 4x5 matrix will be labeled from upper-left to bottom-right as 1, 2,…, 19, 20 with this method). After the matrix is computed the squares that the hands lie on are determined and the labels (integer values) are passed to the Recognizer.

7.1.9. Interface/Exports

The SpaceZoningMatrix is the data produced by this module. It is not directly sent to any other module, instead the matrix is used by this module itself. By using SpaceZoningMatrix, the matrix indices that the hand is on are determined and sent to the Recognizer component. The other production of InputHandler is the hand positions with respect to screen box that is to be sent to InterfaceHandler. The video stream is also transfered to InterfaceHandler by this module.

The initialization task of the other modules is a function without any return value. By this function the application has its components and classes initialized and ready to be used during the whole running time of it.

7.2. Recognizer Component

7.2.1. Classification

Recognizer module is a class of the system.

7.2.2. Definition

The role of this class is to determine the performed gesture by comparing them with the pre-defined gestures. The main functionality of this component is delivered by the help of HMM sub-component.

7.2.3. Responsibilities

Since it receives the hand positions of the movements from InputHandler for 30 times per second, an elimination of unnecessary matrix indices is required. Only the changing positions of the both hands are to be kept in the stacks. This task is handled by the Recognizer.

This component is also responsible for releasing the stack(s) when the hand(s) stand still for memory considerations.

The most significant role of the module is determining which gesture is performed. To do this the Recognizer manipulates the input, constructs the right/leftObservationList and sends it to HMM instances defined for each gesture.

7.2.4. Constraints

The Recognizer module will be active when the application is in either communication or education modes. The activationState is set/reset by InterfaceHandler. For any gesture to be recognized, it should start/end in the correct region of squares of SpaceZoningMatrix.

7.2.5. Compositions

Recognizer module involves only one sub-component namely HMM. It stores HMM instances for each gestures in an array called hmm.

7.2.6. Uses/Interactions

This module interacts with three classes namely InputHandler, InterfaceHandler and HMM. Recognizer receives the matrix indices of both hands from InputHandler. By manipulating this input, it creates the observationList and sends it to HMM instances. According to output of HMMs, the determined gesture name is sent to InterfaceHandler. InterfaceHandler also activates/deactivates the Recognizer according to chosen interface mode.

7.2.7. Resources

There are no external entities directly used by Recognizer component.

7.2.8. Processing

The first functionality of the Recognizer module is eliminating unnecessary matrix indices for performance considerations by following the changes in squares of SpaceZoningMatrix. If the position of a hand on SpaceZoningMatrix is not changing, it means that the input

coming from InputHandler is the same with the previous one. So, it compares the new hand position with the top element of the related stack, if these values are the same, new hand position is not pushed to stack. This prevents filling the stacks unnecessarily due to the fact that Kinect produces output 30 times per second which means the same number coming from InputHandler while the hand is moving along the same square of the SpaceZoningMatrix. Thus, only the changing positions of the both hands are kept in the stacks.

Determining gesture meaning by using HMMs is the main responsibility of Recognizer. To be able to do this, Recognizer constructs the rightObservationList/leftObservationList from the first n elements of the each hand stacks. The length of the observation list (n) is determined by the maximum/minumum observation number of the HMMs. According to the value of n, Recognizer makes comparison between the first element of the stack and HMM's endRange, and also between nth element of the stack and HMM's startRange. The reason of these comparisons is to increase the performance by sending observationList to less HMM instances. If these values match, observationList is sent to the related HMM to calculate the occurrence probability of this observation sequence and then HMM returns the probability of observing the given sequence to Recognizer. If this probability is bigger than the threshold value of HMM, this means that HMM's gestureMeaning corresponds to the performed gesture and this meaning is sent to the InterfaceHandler component. Whenever a gesture is recognized, these n elements of the related hand stack are released.

Releasing the stacks when the hands stand still is the another duty of the Recognizer. While performing the gesture the user should move his/her hand(s) over a square of the matrix in a time less than a second. If the position of a hand does not change for such a time, the right/leftHandStack should be released for using memory efficiently. This is done by two counters for each hands. This counter is incremented each time the Recognizer receives the matrix indices from InputHandler. Since the input comes 30 times per second when the counter reaches 30 it means that the user is not moving his/her hands for almost a second.

Thus, the right/leftHandStacks can be released. By doing so, there will not be any missed gesture since the previous observations are already checked for gesture recognition.

7.2.9. Interface/Exports

The observationList and the recognized gesture name are the main productions of this module. The observationList is a list constructed for an HMM and holds appropriate number of observations for this HMM. The number of observations is determined with the HMM's maximum and minimum observation numbers. There are two observation lists for two hands. The gesture name is the string that is the movements passed to InterfaceHandler module to be presented on the screen.

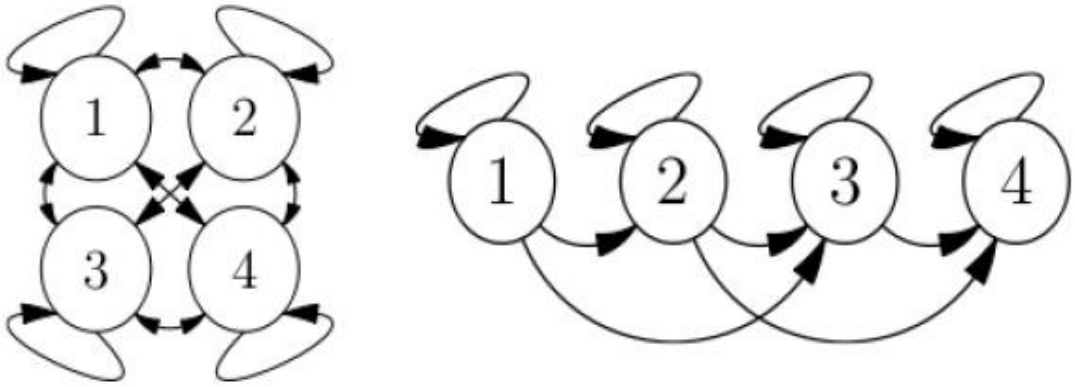
7.3. HMM Component

As explained in 5.2.3, each HMM sub-component will contain a Hidden Markov Model, and also they will have some additional values that are not a part of a model, but related to its gesture. A brief description of the model was provided in section 5.2.3 and in this section usage of these extra attributes and a more detailed explanation of the model will be stated.

System uses one or two first-order, left-right discrete Hidden Markov Models for every gesture. Since observation set of each HMM will be one dimensional and discrete values (single integers), model is called discrete. First-order means that an observation at time t will be emitted only by the hidden state at time t. So, observation matrix of a first order Hidden Markov Model has one probability value that matches a state with an observation. Left-right implies a special topology of the model. In a left-right model, a state q_i has transition probability different from zero with itself, q_{i+1} and q_{i+2} states only. Also, initial distribution matrix enforces model to start with the first state q_0 in this left-right topology.

These restrictions make the model more powerful, because successively ordered state sequences are more likely to represent changes over time.

Standard topology of an HMM (ergodic model) and left-right model can be seen in Figure 10.



(a) (b)

Figure 10: Standard HMM(a) Left-right HMM(b)

This sub-component will hold model related matrices and a function that applies forward algorithm. Training of the model, in other words, getting effective values of the matrices will be done externally, and training functions will not be added into the system. Before training, different observation sequences of each gesture will be collected as training set by repeatedly performing gestures in front of the Kinect. The characteristic values of a gesture (possible starting and ending points, average length of a gesture etc.) that HMM component holds will be extracted from training set manually. In the first stage of the training of a model, matrices will be initialized. Experiments show that initializing matrices with approximately equal values (but not with uniform values, they should be slightly different) gives the best training results. State number of each model will be set equal to the average length of the sequences in training set. State transition matrix will be initialized according to the constraints of left-right model.

Figure 11 illustrates the state transition matrix of a left-right model with four states. Except the last states, transition values can be nearly ⅓ for achieving best results.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}$$

Figure 11: State transition matrix

In second stage, initialized model will be trained iteratively. While training, Baum-Welch algorithm for multiple observation sequences will be applied on the model. Algorithm calculates probability of each sequence of the training set with respect to the current matrices of the model, then compares the products of these probabilities with the products of old probabilities. If new probability product is bigger than the older one, values of each matrix will be re-estimated. Re-estimation can be executed while there is a significant growth in probability products. Although there is no analytical approach to create the best model, it is known that Baum-Welch algorithm raises the likelihood of an observation sequence which is similar to the ones in training set.

After training completed, occurrence probability of each observation sequences in the data set will be calculated and the minimum value will be set as threshold of the model. Validation of these threshold values will be tested during implementation. If it is necessary, for example if a correct performance of the gesture cannot pass the threshold, these values can be changed later. When all these training stages successfully completed and necessary information gathered, HMM component will be added to the system.

7.3.1. Classification

This sub-component will be implemented as a class with data members and methods, and its instances will be employed by Recognizer component.

7.3.2. Definition

HMM sub-components will be used to determine the likelihood probability of a set of numbers which is extracted from the movements of hands. Probability is calculated with respect to HMM's training set.

7.3.3. Responsibilities

Every HMM object in the system is assigned to a different gesture. They are responsible for containing name, and characteristic values of its gesture, and providing them to Recognizer. Also, instances of this class are capable of calculating occurrence probability of an observation sequence which is composed and provided by Recognizer.

7.3.4. Constraints

There are some constraints for the HMM component related to the start, end position of a gesture and the length of the observation sequences.

In order to make HMM component to recognize performed gesture more precisely and effectively, there must be a range for the start and end positions of each gesture. For example, in a 4x12 SpaceZoningMatrix, for the gesture "Goodbye" right hand should start its movement from the chest level within the squares of matrix in a range of <19,20,21>. Also, end position of the right hand is restricted in that manner. These numbers will be precisely determined after gathering several experimental values. In addition to starting and ending positions, the length of a gesture sequence is also restricted. Possibility calculation

will be executed only if the length of the sequence is in between pre-determined range. For instance, the observation sequence of the gesture “Goodbye” could have a length of either 4, 5 or 6 but not 7. In order to cover all different starting-ending points and maximum-minimum lengths of a gesture, it is important to collect different performances of a gesture as much as possible while creating training set.

These constraints are necessary to make the comparisons more efficiently since there will be at least one HMM for each gesture and the system should determine the best-fit HMMs for the performed gesture by looking at the starting and ending states and also the length of the sequence.

7.3.5. Compositions

Since HMMs are already sub-components, all functional responsibilities of them are covered by their data members and related methods that works on these members. In other words, there are no sub-components of an HMM.

7.3.6. Uses/Interactions

Being a sub-component of Recognizer and required only for Recognizer, HMM component has an interaction only with the Recognizer component. This interaction appears whenever Recognizer is in the active state, that is, every time a gesture is performed to be recognized. After InputHandler component gets the input and Recognizer makes necessary comparisons and arrangements with this input, it sends it to the related HMMs for probability calculations.

7.3.7. Resources

There are no directly used external resources for the HMM component. There exists some open-source libraries for Hidden Markov Models with discrete observation sequences,

however in TSL-Kinect instead of manipulating existing codes for our purpose; we decided to write our own HMM implementation.

7.3.8. Processing

The main functionality of an HMM will be calculating the logarithmic probability of a given sequence. Another responsibility of this sub-component which is collecting and providing gesture-related values will not need any special process. All these values are hold as public attributes of this class, and can be directly reached by Recognizer.

When an observation sequence extracted by Recognizer from a hand stack and is given to HMM, it executes forward algorithm (other name of this algorithm is alpha pass) as explained in 5.2.3.

Alpha value for a state i and at time t is the occurrence probability of first t observation of the observation sequence, and also occurrence probability of state t. For an observation sequence with length T, alpha values of each state at time t=0 will be initialized with initial distribution matrix and related observation matrix value for each state q=0,1,2....N-1. For observations at time t=1,2,3,...,T-1, alpha values can be calculated recursively by using alpha t-1 values. Calculation can be summarized as follows:

1. Let $\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$, for $i = 0, 1, \ldots, N-1$
2. For $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, compute

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right] b_i(\mathcal{O}_t)$$

After alpha values calculated until T-1, the probability of sequence is equal to the sum of alpha T-1 for each state.

Although this calculation is mathematically valid, since it deals with very little probabilities while T grows, and alpha values go underflow eventually. The solution to this underflow problem is to scaling numbers. With scaling, in each stage of the pass, sum of the alpha values of each state is recorded to the scaling coefficients array, and each alpha values of time t is divided by this sum. Occurrence probability of the sequence can be calculated by inverted products of the scale values. Again, we can avoid underflow problem by taking logarithms of scaling factors and summing them. Then, logarithm of probability will be equal to negative of logarithmic sum of each scale value for t=0,1,2,...T-1. All of these calculations will be held in probabilityCalculation function of HMM component, and return value of this function will be the logarithmic probability value.

7.3.9. Interface/Exports

Gesture related information is kept as public variables and they can be reached externally. Each HMM holds name of the gesture in gestureMeaning string variable, how many hands gesture has in howManyHands variable, name of the hand (left or right) that HMM is assigned in handName string variable, possible starting and ending points as arrays of integers in endRange, startRange variables, minimum and maximum lengths as integers in minimumObservation and maximumObservation variables, and a threshold probability value of a gesture as double in threshold variable.

They take data from Recognizer and executes scaled forward algorithm with calculateProbability function. This function returns result as double.

7.4. InterfaceHandler Component

7.4.1. Classification

InterfaceHandler component is also a class with several attributes, methods and two subcomponents.

7.4.2. Definition

InterfaceHandler is the component that manipulates user interface modes. In TSL-Kinect project there will be four interface modes: Main Menu, Tutorial, Education and Communication Modes. The main functionality that InterfaceHandler takes hand is arranging the displays namely text-boxes, buttons and labels for each user interface modes by checking the switches between modes.

7.4.3. Responsibilities

This component is responsible for all the necessary user interface functionalities. Basically, it changes the screen contents according to the user interface mode changes. The visibilities, fonts or sizes of the screen images are also decided by InterfaceHandler while the application is on process. InterfaceHandler is also responsible for activating and deactivating the Recognizer component whenever a change occurs in user interface modes from education/communication modes to main menu mode or vice versa.

7.4.4. Constraints

The functionality of this component is completely depends on the Microsoft SDK's image processing ability. Since Microsoft Official SDK is not good enough to recognize the hand positions when two skeleton joints overlap, it is assumed that controlling the user interface

is performed by only one hand and the initial position of the hand is away from the body in order to be sure that there is no overlapping between two skeleton joints.

7.4.5. Compositions

InterfaceHandler is composed of two sub-components which are InterfaceMode and Button. InterfaceHandler includes an array of InterfaceMode instances called “modes” and InterfaceMode sub-component has an array of Button instances called “buttons”. InterfaceMode class exists for storing the positions of every button displayed in the active user interface mode. This information is required to compare the scaled hand positions, with respect to the screen, with the button positions in order to perform the “click” event. As it can be understood from the name Button class is for holding the information about all the buttons used in the user interface. An instance of Button class provides the height, width, text and position information of a rectangular button image.

7.4.6. Uses/Interactions

Apart from sub-components (explained in section 7.4.5.) InterfaceHandler is in interaction with both InputHandler and Recognizer.

The interaction with InputHandler appears every time InputHandler repeats its functional routines, i.e., whenever a new input is generated by SDK (30 times per second) to display the video stream on the screen and make comparison for interface mode changes.

On the other hand, the interaction with Recognizer only appears when the system user selects the education or communication mode or returns to the main menu from these modes.

7.4.7. Resources

For this component there are no directly used resources. However, since it is in interaction with the InputHandler to get the video stream and scaled hand positions, InterfaceHandler indirectly requires Coding4Fun toolkit and NUI library. The provided functionalities of these libraries are explained in the section 7.1.7.

7.4.8. Processing

As is the case in the other components of the system an InterfaceHandler object is created by the initialization() function placed in the InputHandler component.

Two main functionalities are handled by this component: changing the user interface modes and displaying the gesture meanings on the screen.

Since there are four user interface modes of the system, InterfaceHandler creates an array of InterfaceMode object with size four. The transitions between these modes are decided by checking whether the hand position is in the area of an activation button belonging to the related mode. All the buttons of the user interface are represented by rectangular areas and a click event is checked by whichButton() function of the InterfaceMode class. This function takes the scaled hand positions 30 times per second and it finds the associated button by checking whether the hand position lies on the rectangular area of a button. After finding the related activation button a button counter, which is the buttonTimer attribute of a Button object, is incremented by one and if this counter equals to a pre-determined upper limit, it means that hand remains its position within the border of the button and so a click event is performed. After checking the click event of a button, changeMode() function in InterfaceHandler is called and it rearranges the view of the user interface by manipulating the visibilities of the related interface mode buttons and textboxes if any.

When the application is started, user will be confronted with the main menu. User will be able to switch between tutorial, communication and education modes from the main menu. When a click event is detected for the tutorial mode, a video will be displayed to teach the use of the system visually.

When a click event is detected for communication mode, InterfaceHandler activates Recognizer. Recognizer sends the name of the performed gesture to InterfaceHandler whenever the recognition process is succeeded. The name of the performed gesture coming to InterfaceHandler is of string type and if the gesture could not be recognized the content of the string is empty. These strings are stored in the “gestures” attribute of InterfaceHandler which is of type string array with size four in order to display the content of gestures in the textbox. The displayed content of the textbox is updated by this way: Whenever a new no-null string arrives to InterfaceHandler, it is placed in the first position of the array and current value is placed in the second position , second string goes to the third position and third one goes to the fourth position.

When a click event is detected for education mode, InterfaceHandler again activates Recognizer for recognition process. In this mode user will be asked to perform a gesture which is determined randomly by the randomGesture() function. The user will have ten seconds to perform the asked gesture and the timer on the screen will be decremented until user performed the correct gesture. In each time user makes a movement Recognizer sends a string to the InterfaceHandler and this string will be compared with the randomly chosen one. If these two are decided to be equal, at the end of simple string comparison process, then counting down process is stopped and a success message is shown on the textbox. If these two strings are never the same within the 10 seconds duration, a failure message is displayed for a while and another gesture is selected randomly to repeat the same process.

Both communication and education mode interfaces contain a “Back” button in order to

return to the main menu. When a click event is detected for the “Back” button, InterfaceHandler deactivates Recognizer so the system will not perform any gesture recognition until one of the education or communication mode is selected again by the user.

7.4.9. Interface/Exports

InterfaceHandler has interaction with both InputHandler and Recognizer. The displayVideo function takes video stream from InputHandler to display the image of the user on the screen in every modes of user interface. whichButton function in InterfaceMode sub-component takes the scaled hand position vectors from InputHandler and finds the associated button. changeMode function takes the name of the button and if the name of the button is “Education” or “Communication” Recognizer component is activated by setting its activationState attribute to an integer value for gesture recognition. If the name of the button is “Communication” Recognizer sends the meaning of the gesture via setGestures function. If the name is “Education”, Recognizer sends the meaning of the gesture via compareGesture function. If the name of the button is “Back” InterfaceHandler deactivates Recognizer by setting its activationState attribute to 0.

8. LIBRARIES AND TOOLS

8.1. Hardware

8.1.1. Kinect

8.1.1.1. Description

Kinect for Xbox360,or simply Kinect (originally known by the code name Project Natal), is a motion sensing input device by Microsoft for the XBox 360 video game console. Based around a webcam-style add-on peripheral for the Xbox 360 console, it enables

users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken command.

8.1.1.2.Usage

Kinect handles the image processing part of the project.It support the positions of the joint coordinates of the body in the space where the Kinect is at the center of the space. Here is a diagram representing the joints (Figure 12):

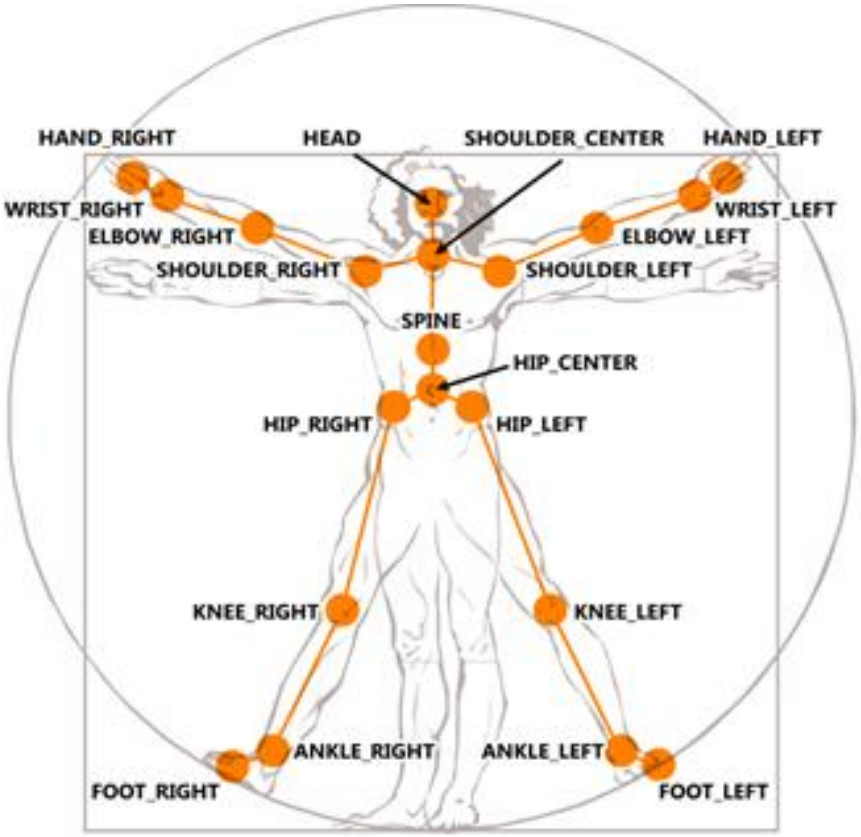


Figure 12: Skeleton joints of Kinect SDK

8.2. Software

8.2.1. Kinect SDK

The Microsoft Kinect for Windows SDK provides the native and managed APIs and the tools you need to develop Kinect enabled applications for Windows. This SDK provides support for the features of the Kinect sensor (color images, depth images, audio, skeletal data, etc.).

Basically, the Kinect sensors will send a set of three streams:

Image stream can be displayed like with any other camera (for example to do augmented reality). The Kinect video sensor can return a stream with 2 resolutions: one at 640x480 (at 30 frames per second) and one at 1280x1024 (but at 15 frames per second).

The depth stream is the determining factor in our case. It will indeed add to each pixel a depth defined by the sensor. So in addition to the 2D position of each pixel (and color) we now have depth. This will greatly simplify the writing of shapes detection algorithms.

A third stream is sent from the sensor: it is the audio stream from the four microphones. In this project, obviously this stream will not be used.

Therefore, the key point here concerns the ability of Kinect to give us three-dimensional data. Using the NUI library (which comes with the SDK and stand for Natural User Interfaces) you will be able to detect the presence of humans in front of the sensor.

8.2.2. Microsoft Visual Studio 2010 Express

Microsoft Visual Studio is a powerful IDE that ensures quality code throughout the entire application lifecycle, from design to deployment. Whether you're developing applications for SharePoint, the web, Windows, Windows Phone, and beyond, Visual Studio is your ultimate all-in-one solution. [http://www.microsoft.com/visualstudio/en-us]

For using Kinect SDK one has to use any Visual Studio 2010 edition. In this project Visual Studio 2010 Express edition is chosen.

8.2.3. Libraries

- NUI library

This library enables the Kinect programmers to get the input in skeleton form.

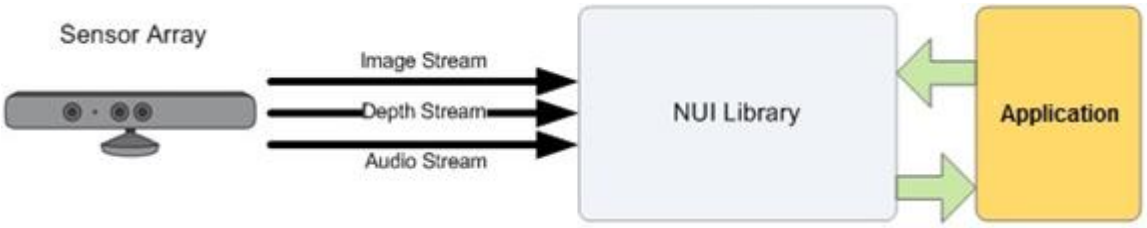


Figure 13: NUI library interaction

- Coding4Fun Kinect Toolkit

The Coding4Fun Kinect Toolkit is a set of extension methods and controls to make developing applications for the Kinect using the Kinect for Windows SDK easier.

9. TIME PLANNING

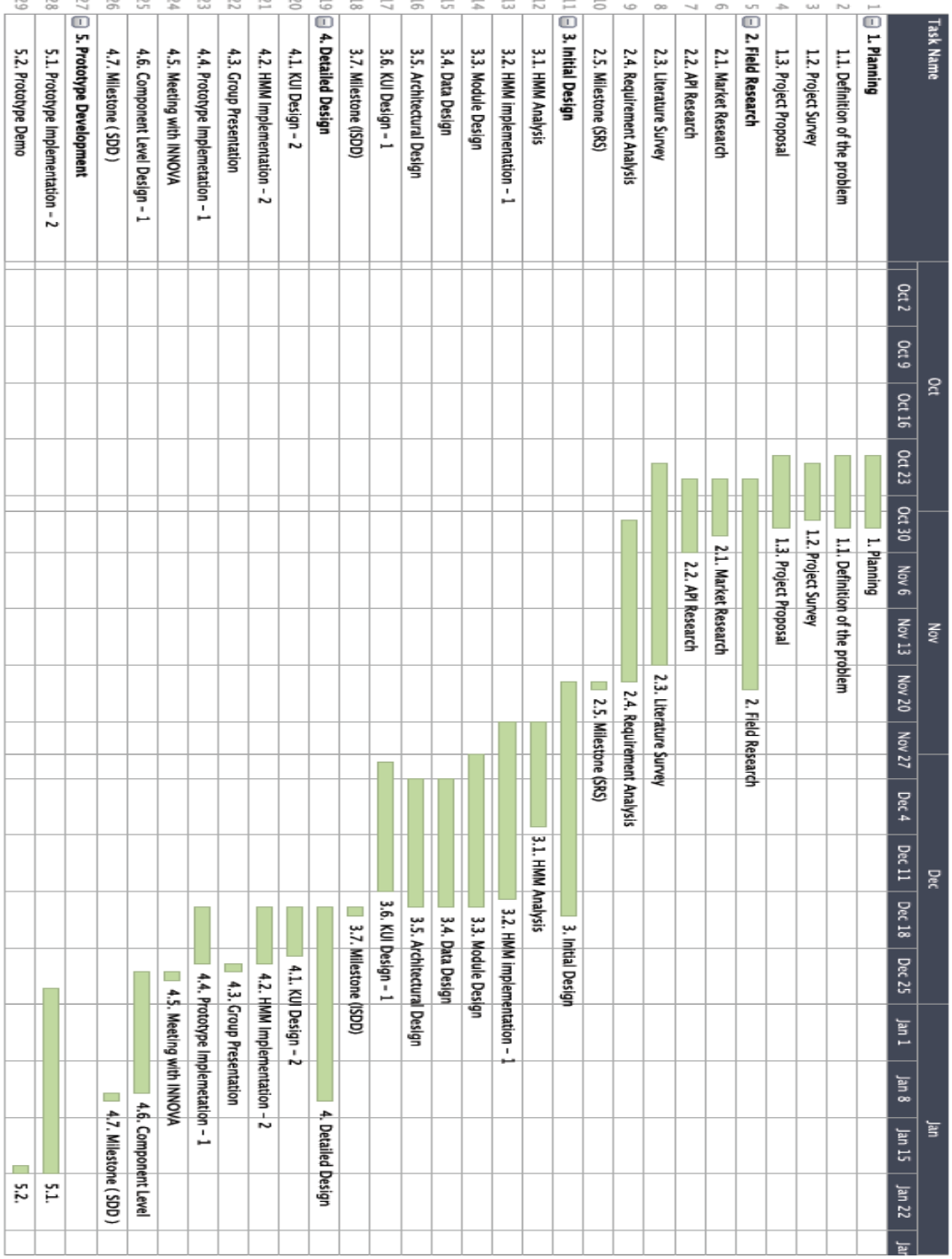


Figure 14: Gantt Chart for Term I

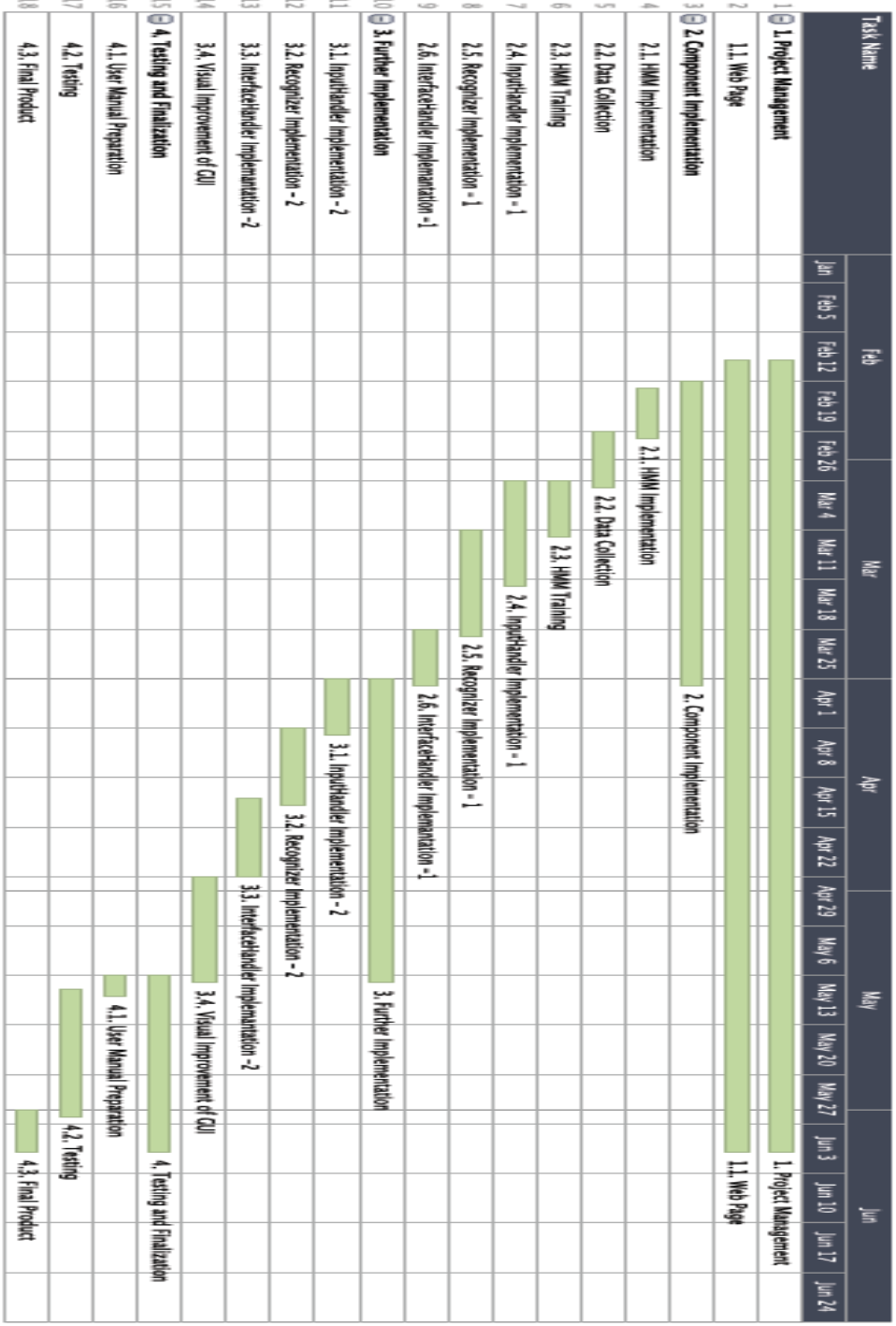


Figure 15: Gantt Chart for Term II

10. CONCLUSION

In conclusion, Detailed Design Report for TSL-Kinect gives the definition, purpose and scope of the project. The possible design and other constraints that can be faced are explained. The tools and the libraries that will be used during developing the project are decided. Component diagrams, class diagrams, sequence diagrams and interface features are given within the document. We have explained the works that we have done so far and within the schedule we give the future work to be done.