D-BUG

Initial Software Design Document

DUYGU ARALIOĞLU
BEDİA ACAR
ZÜLFÜ ULAŞ ŞAHİN
GÜLNUR NEVAL ERDEM

MIDDLE EAST TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
2011

Table of Contents

1. INTRODUCTION

Initial Software Design Document of TSL Kinect project is the representation of the system by outlining necessary design demands. 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 ISDD 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 ISDD is divided into nine 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 are explained.

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

In section eight, 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 operation 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 is 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 initial 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.

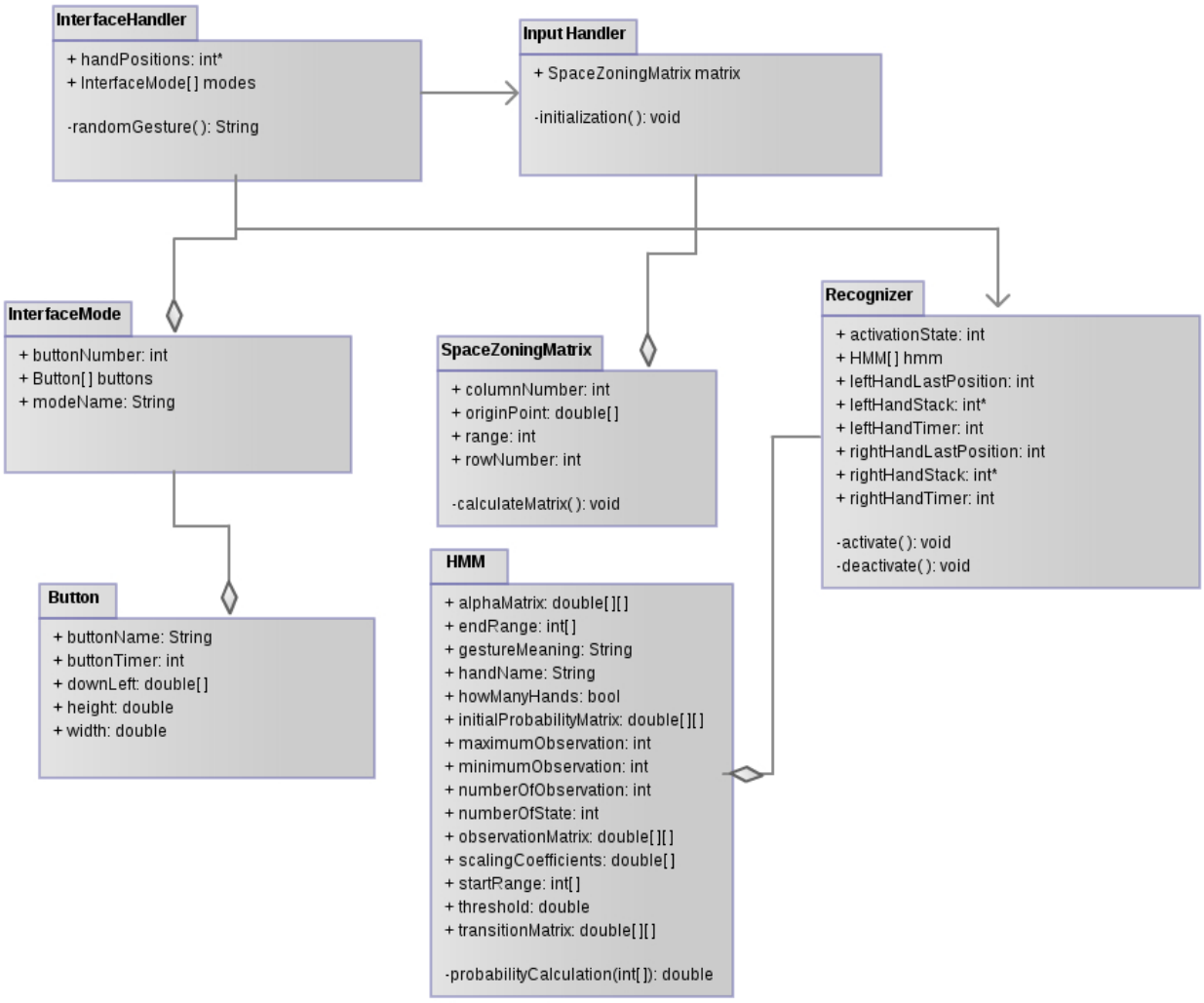

Figure1 : Class Diagram of the System

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.
- observationList:
- 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:
- InterfaceMode[] modes:
- 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.

Methods:

- randomGesture(): String Randomly chooses a gesture name from gestureNames array.

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:
- modeName (string):

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):

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.

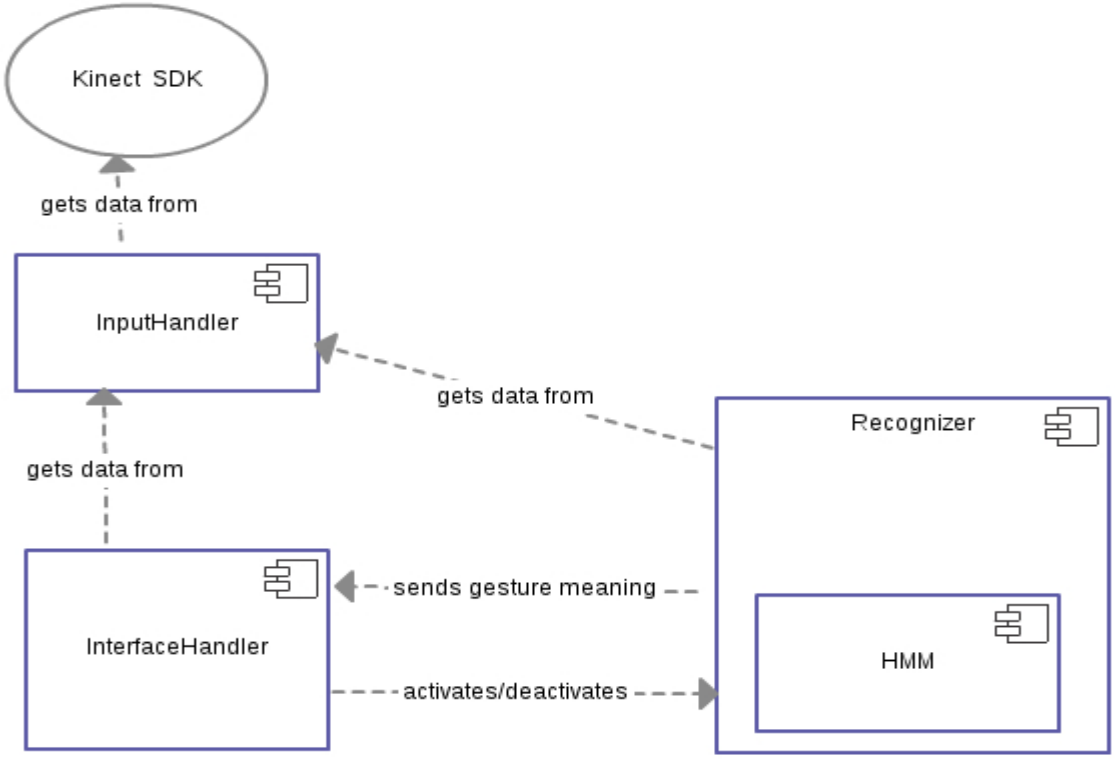

Figure 2 : Component Diagram of the System

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 below.

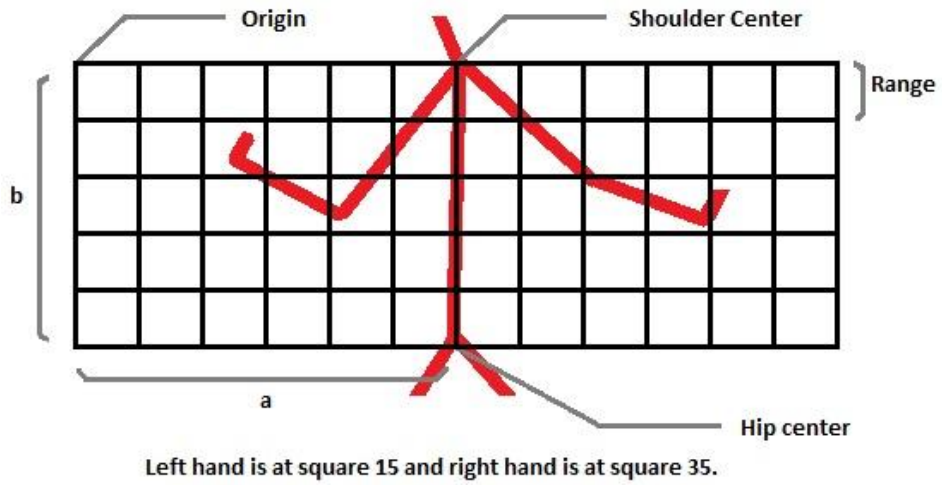

Figure 3: Space zoning matrix

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.

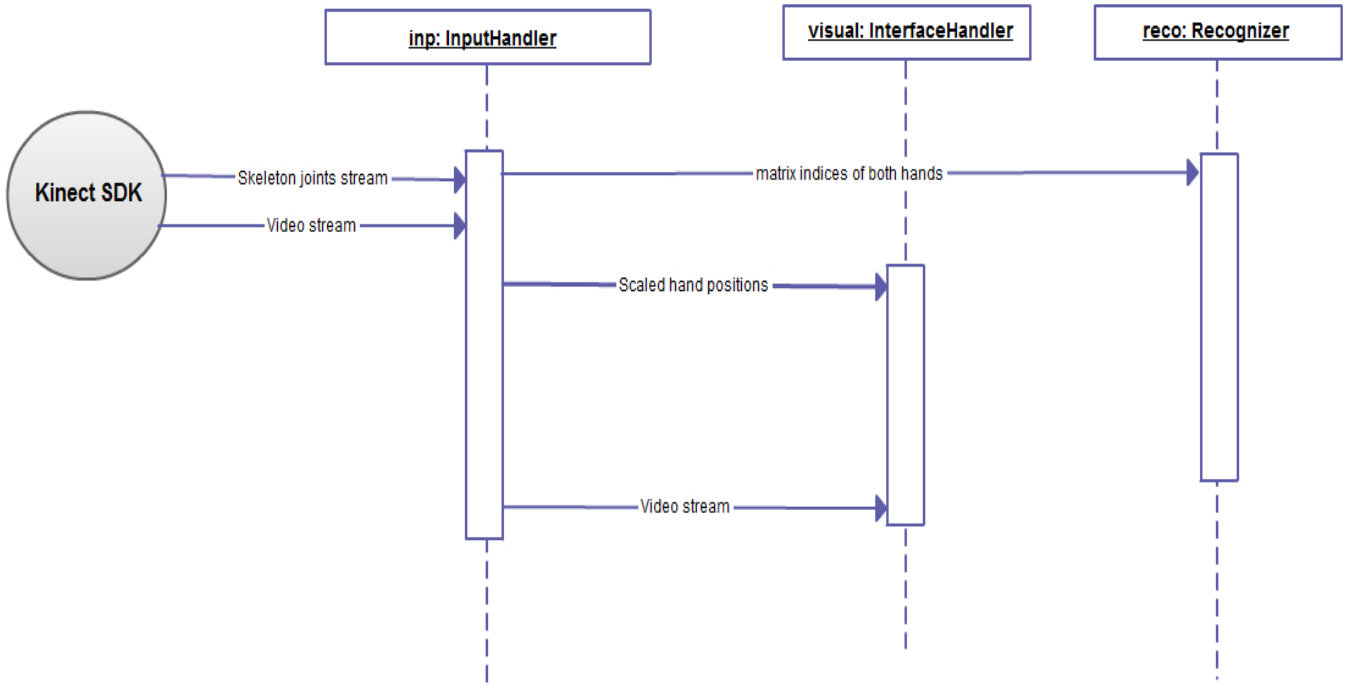

Figure 4: Sequence Diagram of InputHandler

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 HMM's 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.

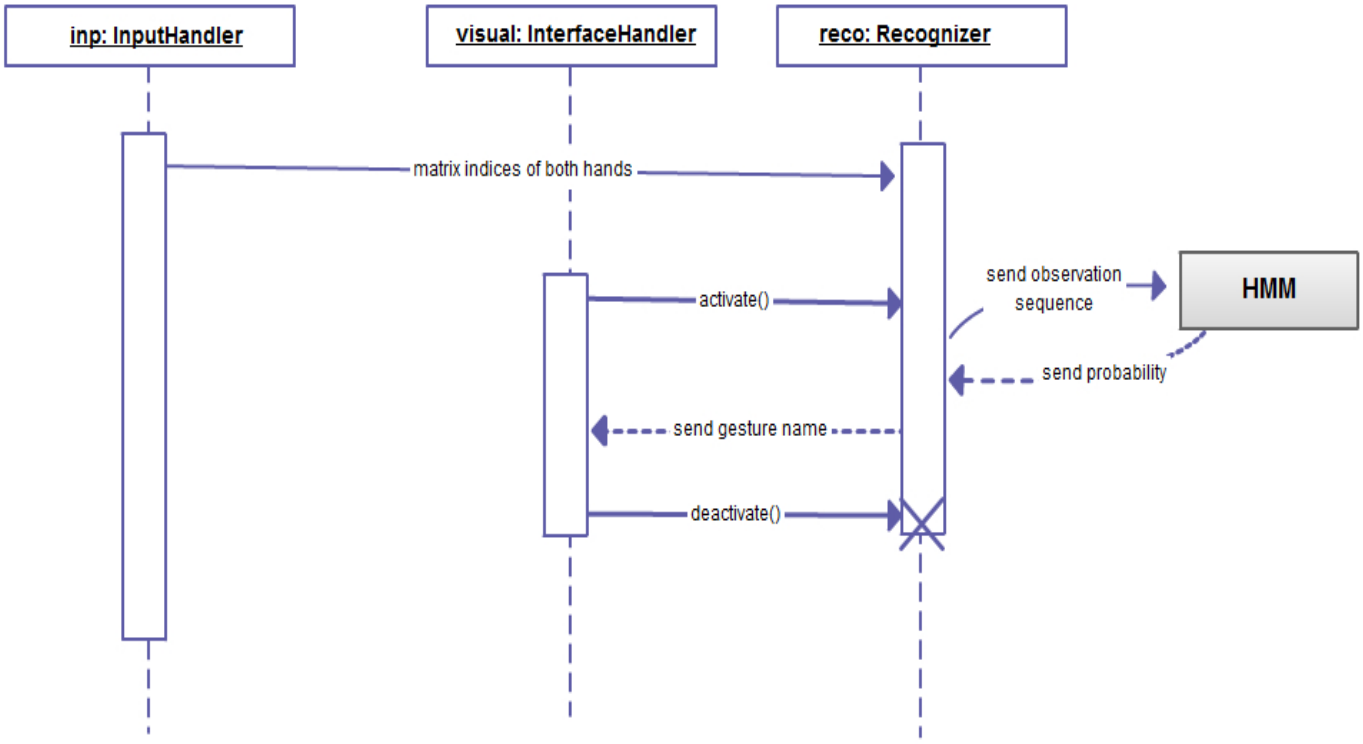

Figure 5: Sequence Diagram of Recognizer

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 is a 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

This 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.

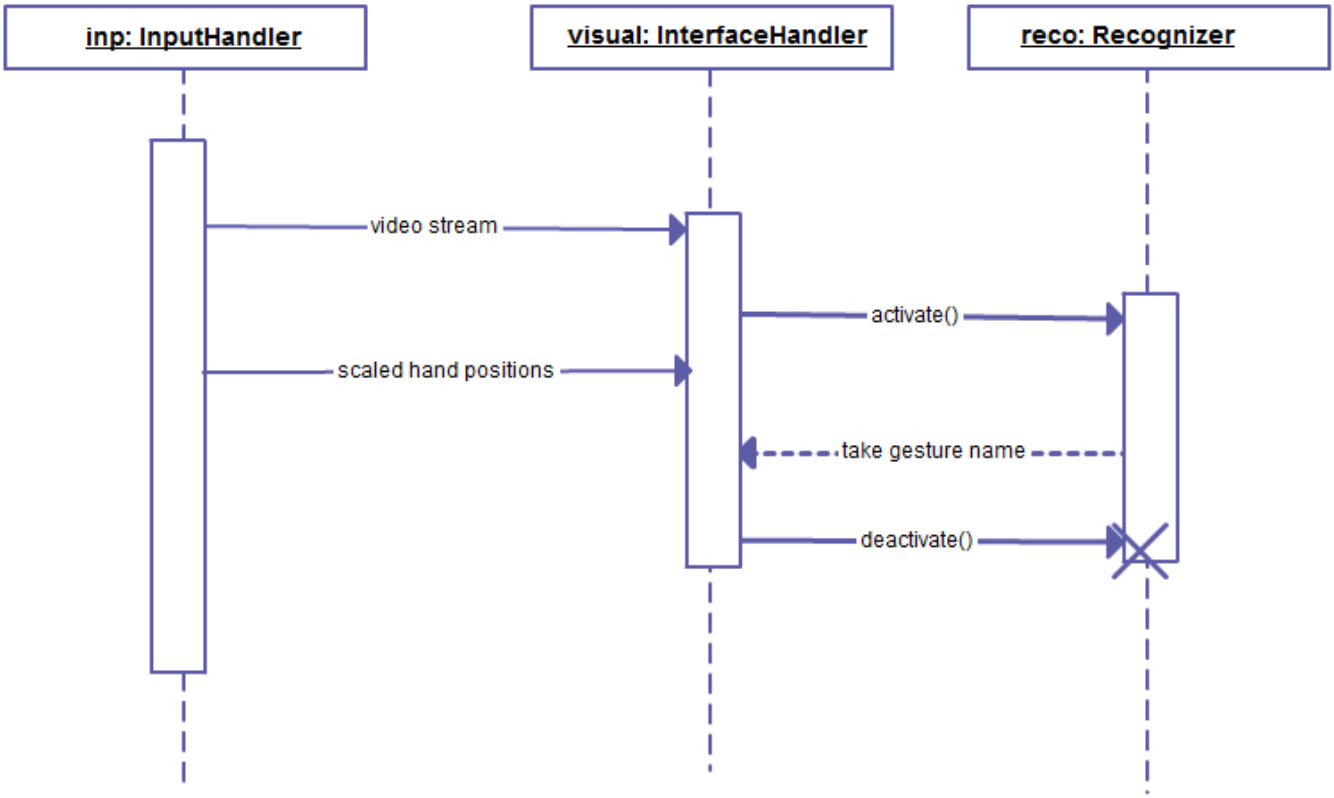

Figure 6: Sequence Diagram of InterfaceHandler

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. TRACEBILITY 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 no 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 is 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

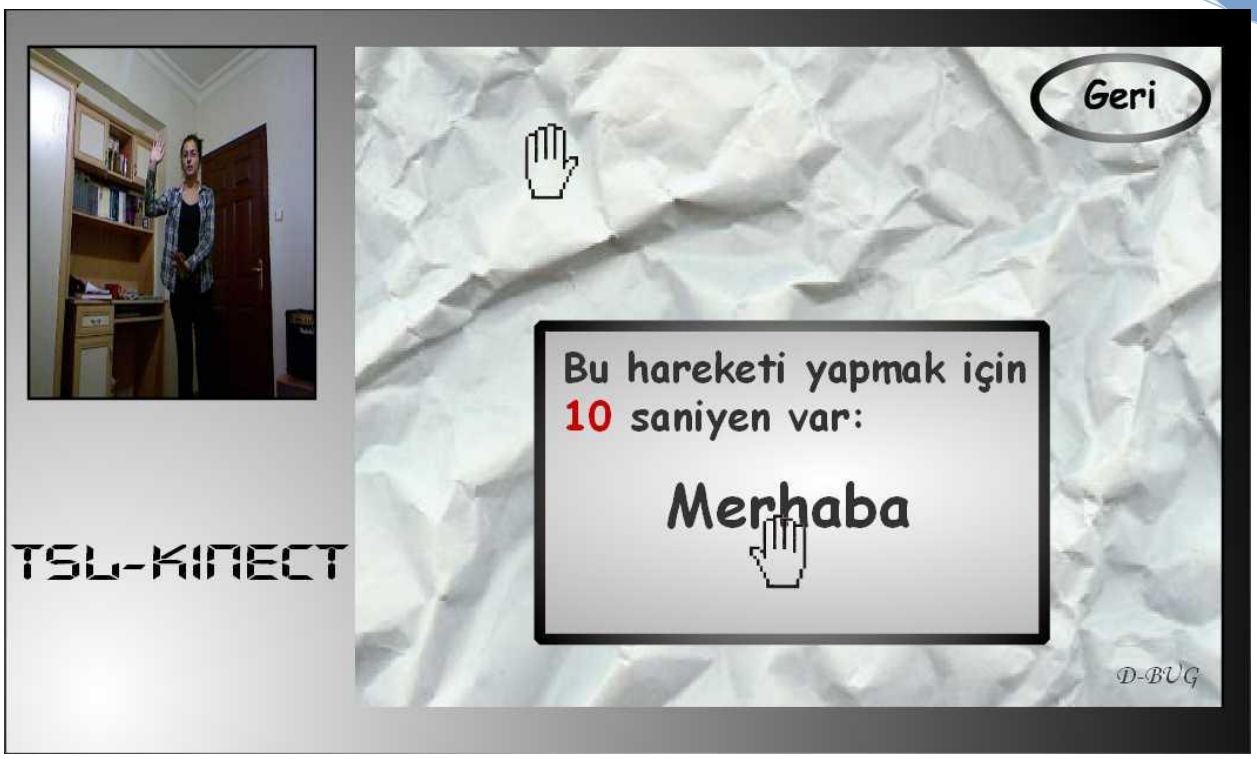

Figure 8: ScreenShot of Education Menu

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. LIBRARIES AND TOOLS

7.1. Hardware

7.1.1. Kinect

7.1.1.1. 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.

7.1.1.2.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:

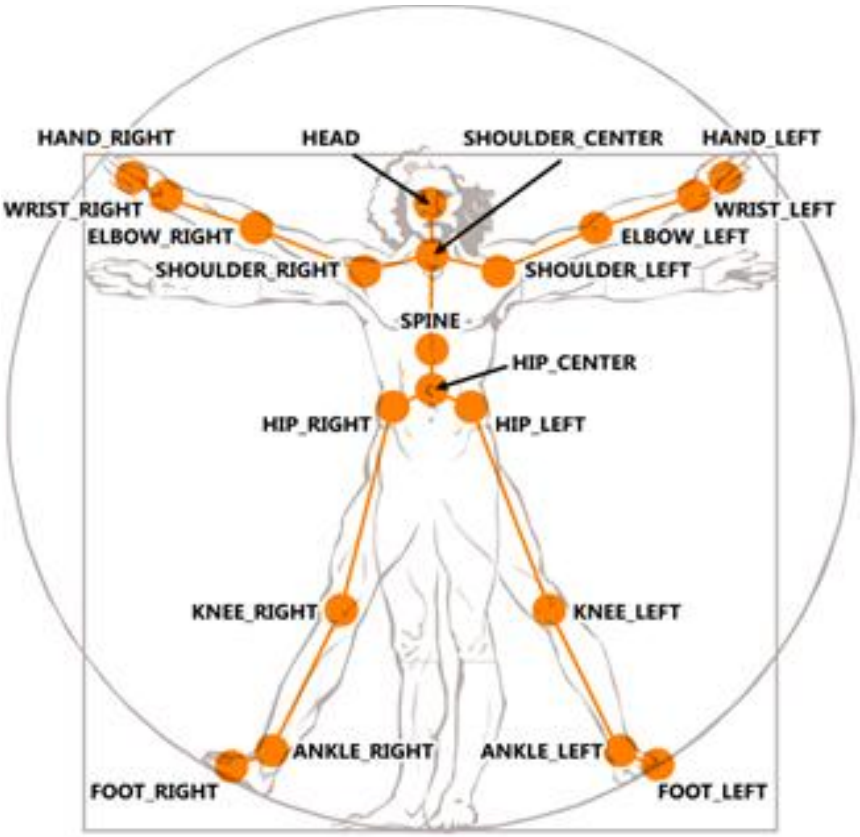

Figure 10: Skeleton joints of Kinect SDK

7.2. Software

7.2.1. 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.

7.2.2. 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.

7.2.3. Libraries

- NUI library

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

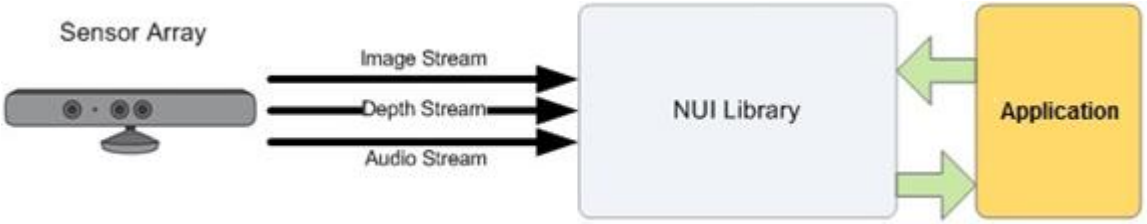

Figure 11 : NUI library interaction

- 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.

8. TIME PLANNING

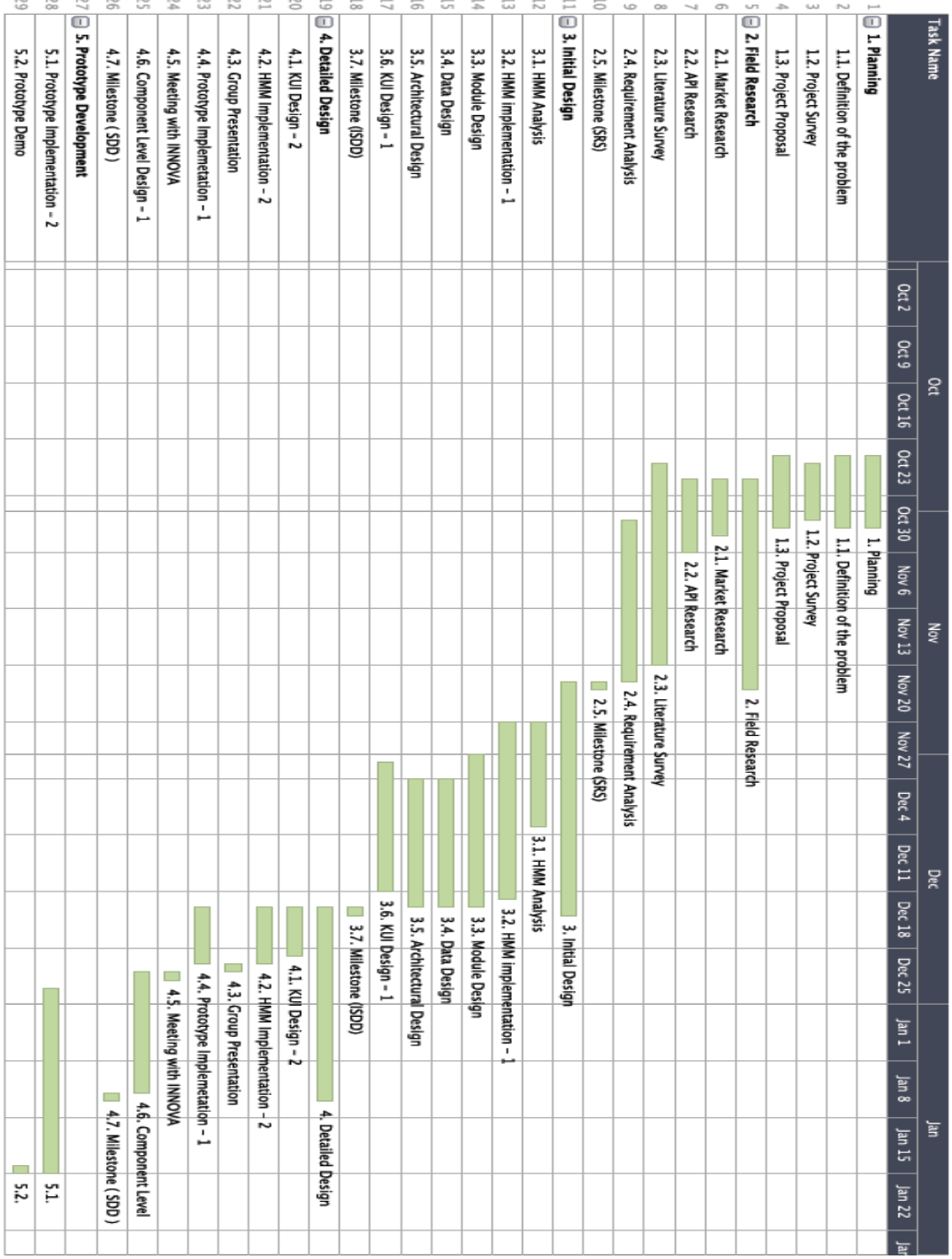

Figure 12: Gantt Chart for Term I

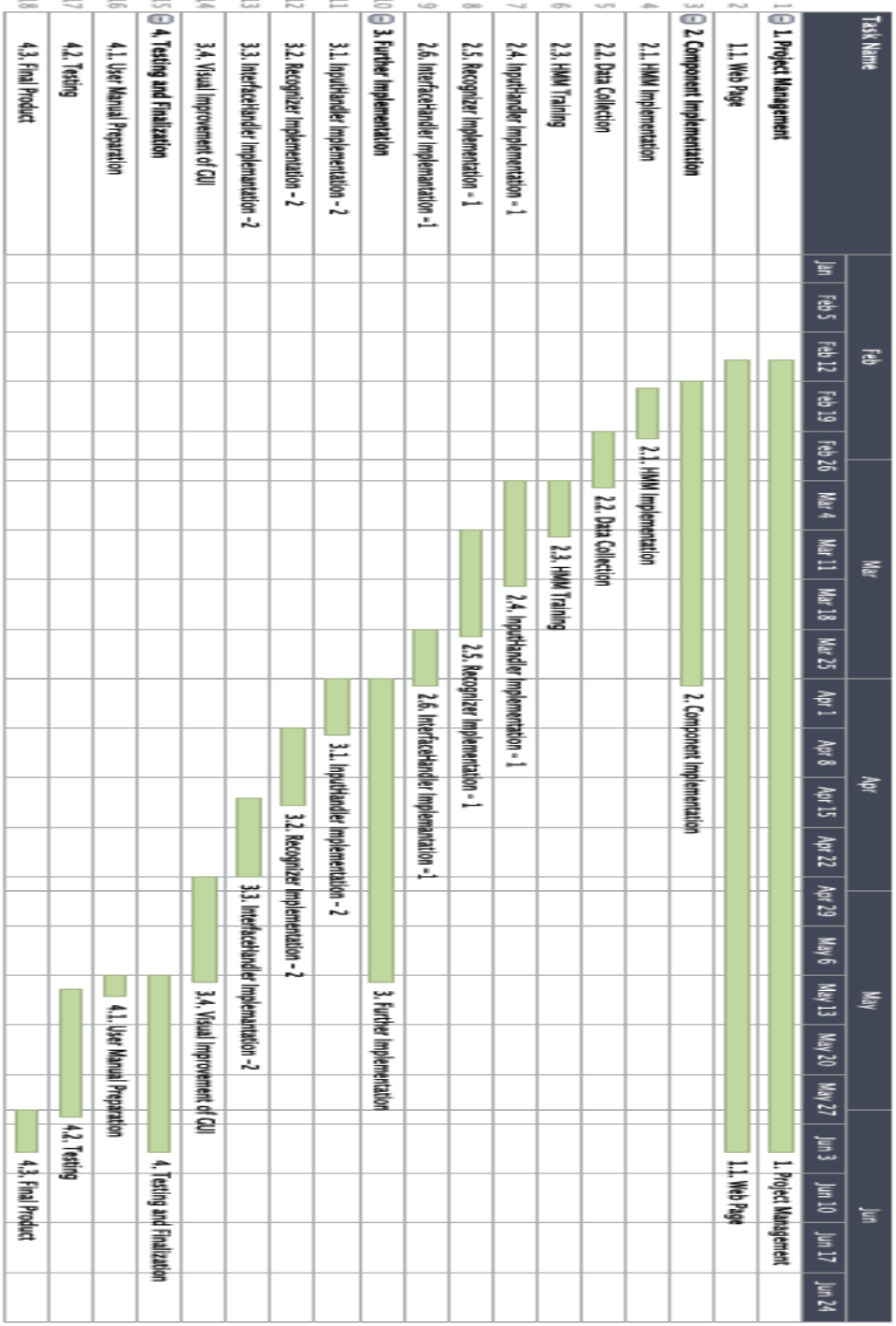

Figure 12: Gantt Chart for Term II

9. CONCLUSION

In conclusion, Initial 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.