TUI TABLE AUTOMATIC EXPERIMENT ANNOTATION TOOL

(ATTAT)

Documentation V1

Author: Mehmetcan Fal

mehmetcanfal@gmail.com

Luxembourg Institute of Science and Technologies

September 2016

1. CHAPTER

INTRODUCTION

ATTAT is an automatic TUI experiment annotation tool that reduces the time and hopefully increase the quality of annotation of multi user tangible table experiments. Basically, ATTAT tries to catch all TUI objects-human interaction and tries to identify human-human interaction while the experiment on progress. ATTAT provides different log files for each user's-object interaction and supports files for detected gesture list for each user. This document consists 5 chapters in which you can find the answer of how ATTAT can be used, how an experimenter can interpret the log files, and how an experimenter can change the target gestures, and finally, technical code description and detailed functional explanation of the application.

Before chapters it could be useful to present technical dependencies and used material that actually form the spine of ATTAT. Since Microsoft Kinect is the central hardware of the application, inevitably all related codes and dependencies taken from Microsoft Kinect SDK pack. Moreover, TUI table and working application within TUI did not changed but a thread class has been added to the original application. In the fallowing sections detailed information about Kinect and TUI table are presented. In is important to note that anyone who may want to use this application needs to understand at least basics of both TUI and Kinect. Indeed, presented information is not detailed as much as in the original source site of devices.

1.1 Microsoft Kinect

Kinect is an advanced user interface that interprets the spatial information of the user and his or her body parts. Simply, it emits an infrared points cloud and then collect this cloud back to measure the brightness and sizes of points; and finally, it interprets the depth information of the object in the environment. The rest of the interpretation depend on an image processing to understand whether any set of points that has similar depth property implies such a shape that looks like a body or not? In order to accomplish mentioned task, Kinect has an infrared emitter, an infrared sensor, RGB camera and a microphone array. An infrared cloud and interpreted depth information can be seen in Figure 1.

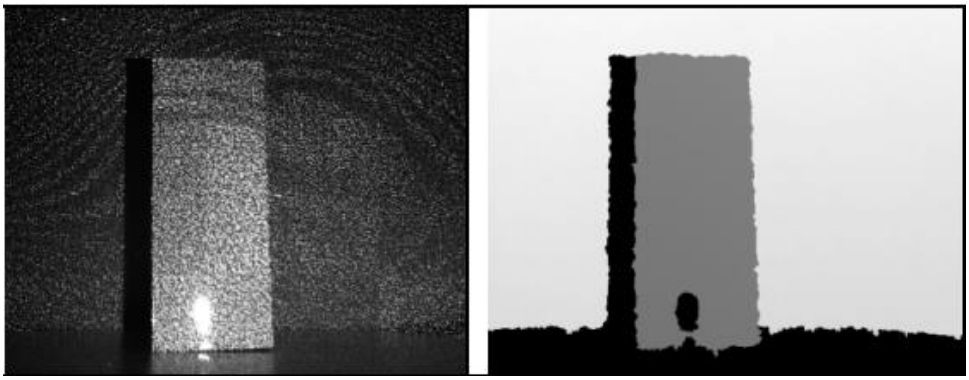

Figure 1. An example of Kinect V2 Points Cloud and the Resulting Depth Image (Khoshelham, 2011).

Kinect is currently a popular user interface for human computer interaction. In HCI, main aim of the novel interface to design more natural and intuitively understandable user interface. Therefore, Kinect is a strong device that user does not need to know any skill or expertise to use Kinect supported applications. There are number of examples that Kinect used as an interface such as games, medical applications or trade service applications.

Kinect V2 provides 15 body's joints that represented in a 3D coordinate system. In Figure 2 body joints provided by Kinect can be seen.

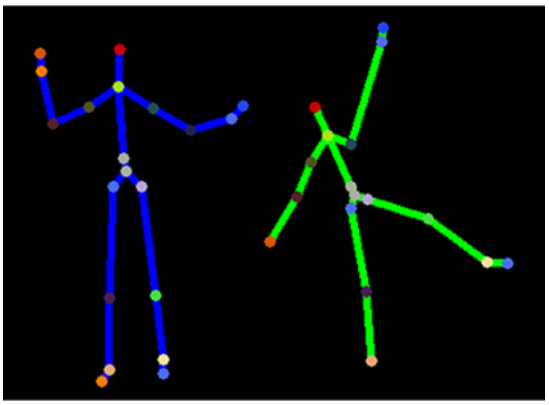

Figure 2.Kinect Body Joints (retrieved from www.microsoft.com, 23.08.2016, 01:00 pm)

In Figure 2, joints are the way that programmers use these joints to understand body posture of the users. Kinect V2 can support 6 different bodies that are separately retrieved from the supported SDKs. In detail, six different tracked bodies' joints refreshed by the sensor 50 times in a second. Further information please visit:

www.microsoft.com/en-us/research/project/kinect-for-windows-sdk-beta/.

1.2 Tangible User Interface

Tangible User Interface (TUI) is a sophisticated user interface that provide a way that user can interact with virtual information through the physical entities. Simply, TUI sense the object on its surface and processes relevant information of the object and provide generally visual information. In Figure 3, a simple TUI table is presented.

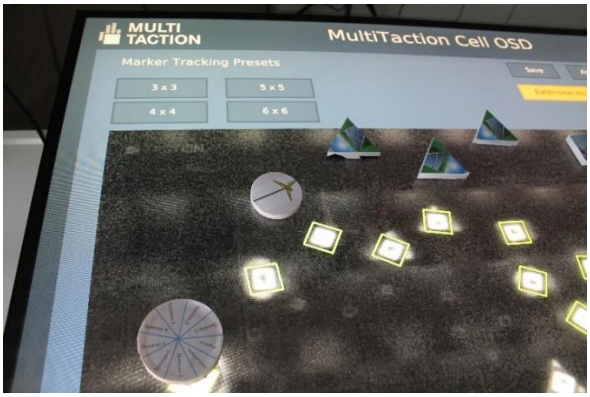

Figure 3Tangible User Interface (Dimitra, Fal, Ras, 2016)

Note that in Figure 3. Several object located on the table and sensed and represented object within the system can be seen that are bounded in a bright square. Mainly, TUI system has two different components: a table and operator pc. Scenario, related object definitions, relations of these objects and images that are being present in the scenario are defined in the operator pc. Table only senses the objects located on the surface and sends information about these objects to the operator pc such as rotation of the objects, positions, id and state of the objects. Processed information send back to the table to provide visual feedback for the users.

2. CHAPTER

HOW TO SETUP ATTAT

In this chapter, detailed instruction of the ATTAT setup is presented. In the following sections, relative position of the TUI and Kinect, calibration procedure and experimental restrictions and important key points are presented.

2.1 Relative Position of Kinect and TUI

This section provides a simple instruction about positions of devices. ATTAT has an ability to locate TUI's surface plane, thus Kinect relative position does not really matter. However, since the collaborative experiments generally consist more than one participants, locating Kinect as the across of the long edge of the TUI provide better sight for Kinect to sense more than one participants gestures and movements. The most appropriate positions of the devices can be seen in Figure 4.

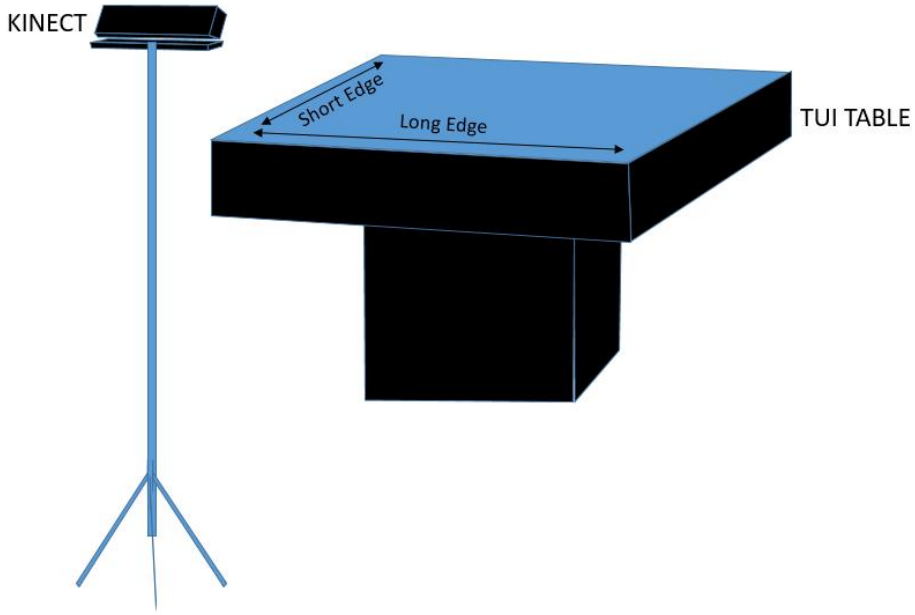

Figure 4.Kinect and TUI Table Relative Position

2.2 Hardware Configuration

In this section detailed hardware setup is presented. ATTAT has a simple hardware configuration that anyone who wanted to use ATTAT can easily configure the setup using this document. The only thing additional to the main components (TUI table, TUI table pc, Kinect, and Kinect pc) is a switch that stands for the communication between all components of the setup. In the Figure 5 you see that all components of the setup need to be connected to the switch, except Kinect. And also the ip configurations are given in Figure 5. In Figure 6, data flow, between components, are also illustrated. Note that arrows depicted in the Figure 6, represent the data flow of the system.

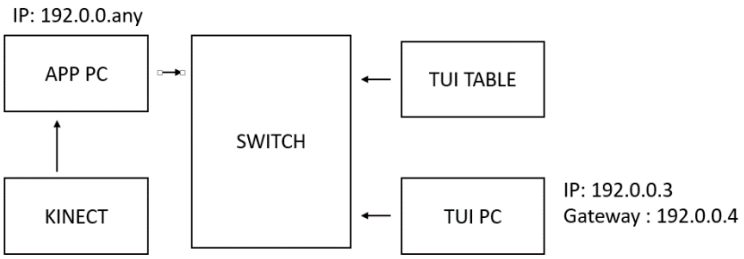

Figure 5. Hardware Configuration of ATTAT

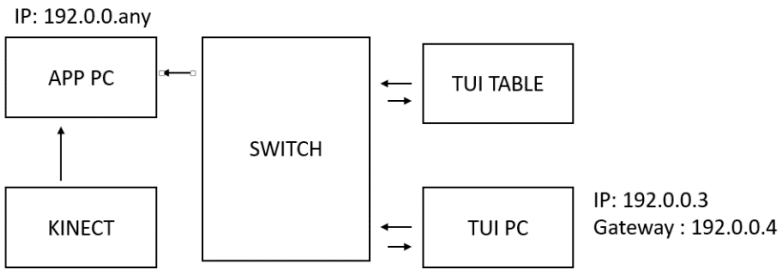

Figure 6. Data Flow Illustration

2.3 Calibration Procedure

Calibration procedure of the ATTAT is vital for understanding object-human interaction. This procedure provides 3D coordinates of the TUI table surface plane. Therefore, an experimenter needs to complete this procedure for each new position of Kinect or TUI table. In order to accomplish this procedure, experimenter should consider that the edge of the TUI table need to be touched and application should run in a same time. Therefore, it is better to have another person to locate each TUI table's corner while experimenter beside the pc that runs the application. To complete this procedure, experimenter should start the application. Application considers the information of TUI table's corners' coordinates, if these coordinates taken previously then application directly starts the experiment. Please note that if the Kinect and TUI table have a new locations and application has a calibration file then whole calculation of objects' coordinates will be wrong. Therefore, experimenter need to be sure about the position of Kinect and TUI table. In order to calibrate the experiment, experimenter should start the application if application has no calibration file then a message box will be appeared that warns the experimenter as "you have no calibration file… get ready to prepare one". When the message box clicked as "ok" then application starts to collect hands' tip location of a person who touches the short edge's corners of the TUI table. Both short and long edge definitions are illustrated on Figure 4. And then, application is going to provide a message box, which is going to warn the experimenter as "please locate your hands other corners of the TUI table". Other corners mean is opposite of the first located corners. Finally, another message box is going to appear about the state of the calibration whether it is successful or not. After calibration procedure you can find the coordinates of the TUI table in the debug directory of the application that consist 4 different double values. This calibration file can be used for any experiment, if the locations of the TUI table or Kinect were never changed.

2.4 Starting Experiment

After the successful calibration message, experimenter should stop the application. Now setup is ready to analyze human-object and human-human interaction. Only thing that experimenter need to do is, application have to be started one more time. Since application has a calibration file, experimental session will start automatically. At the end of the experiment, experimenter should stop the application and she or he has to take the log files into other directory. Note that each experiment has its own data file namely mainlog, outputleft, outputright, outputleft1, outputright1, outputleft2, outputright2, pointingout and tuidifference. Except the "tuidifference", the rest of the log files need to be kept for each experiment, otherwise consecutive experiments' data is going to be written on to existed files. In case, if an experimenter came across with such a problem, he or she needs to analyze the system time to differentiate the taken logs. In order to be clear, log files and their data are introduced in the next chapter. Last but not least, two different gestures installed to the ATTAT that are pointing to the TUI table and pointing to another person. These gestures can be taken out or number of different gesture can be increased. Nevertheless, gesture related information is presented in the fourth chapter.

3. CHAPTER

INTERPRETING ATTAT LOG FILES

In this chapter, each log files is presented. Since current logs have some problems to analyze, there are certain additional works to do for understanding the human-human and human-object interaction. In short, these problems occur because of the event detection. Defining an event, especially for a continuous event, is always problematic. Understanding of the beginning and the end points of any kinematic event is a hot topic in Cognitive Science. Since ATTAT using a model for any gesture, definition of this gesture has a continuum kinematic and temporal sub components, defining the boundaries of this kinematic and temporal continuum is a hard issue. Therefore, in the log files especially related to pointing gesture, number of gestures' logs are taken continuously. Note that, these logs can present a gesture, which is started on time point and end in another point or number of continuously taken same gesture. Additionally, same issue also true for the object-human interaction. There is a hard problem to understand whether these logs represent a same pointing gesture, which was done for only one pointing or there were two or more gestures, which were done for two or more different pointing. If these gestures are consecutive then it is hard to differentiate them. In this point, it can be stated that this is the most obvious problem of the ATTAT that reduces the accuracy. Nevertheless, further improvements of the system hopefully will consist a solution for this problem. In the following sections each log file is explained in detail. Note that each section has a name of the explained log file.

3.1 Mainlog

Before the explanation it is better to present a part of the file. In Figure 7, each line represents the information about the object. In order to make the operation easier, a character "*" star has been used for delimiter that probably provide easier excel operation. So each data that is separated by the star has a certain meaning. The first information is the object's number. And the second is, the horizontal location of the objects with respect to TUI table screen coordinates and the third one is the vertical coordinate. The fourth information provides the state of the object and the last line is the system time that is belongs to the Kinect PC. Actually, these information are only belong to the TUI tablet there is nothing about participants. However, logging the objects information on Kinect PC provides a way to synchronize the TUI table events and Kinect events. Therefore, we can call this log file as fusion log. After a detailed investigation of this file you may recognize that the object states are always indicate an event such as "ROTATING" this is not related to the presented system that TUI table need a noise to get state of the objects' log continuously. The TUI table programmer can provide further information.

```
5*417*319*ROTATING*63599356326172
5*417*319*ACCELERATING*63599356326189
5*417*319*ACCELERATING*63599356326221
5*417*319*DECELERATING*63599356326239
5*417*320*ROTATING*63599356326271
5*417*320*ROTATING*63599356326288
5*417*320*DECELERATING*63599356326305
5*417*320*ROTATING*63599356326321
5*417*320*ROTATING*63599356326371
```

Figure 7. Mainlog File Records

3.2 Outputleft

In this log file, object and human interaction can be seen. Specifically, left hand of the first participants and the object collusion recorded in each line. In the Figure 8, you can see that the number of object, number of participants, hand indicator and the system time are presented. Note that in Figure 8 there are number of records that participants 1 interact with the object 5; however, these logs may belong to only one or number of interaction. In this point, experimenter needs to decide about the identification of each discrete interaction of participants 1 and object 5. Simply, system time can solve the problem that each record line has its own system time and experimenter can decide about the interval between two records. This decision can be made through statistical reasoning or naïve reasoning. Since consecutive records have a homogeneous system time difference, and statistical method can provide a meaningful mean that belongs to mostly same interaction, outliers of this mean can be used as a discriminative metric. Basically, if difference between two consecutive system time is higher than the mean or higher than mean + (2 * standard deviation) then the record can considered as another interaction. Since similar algorithms are used in eye tracking and similar sciences, there is no need to be worry about the result of the system.

```
Object No: 5*****Participants No : 1*****left hand*63600200530167
Object No: 5*****Participants No : 1*****left hand*63600200536523
Object No: 5*****Participants No : 1*****left hand*63600200541587
Object No: 5*****Participants No : 1*****left hand*63600200542499
Object No: 5*****Participants No : 1*****left hand*63600200542866
Object No: 5*****Participants No : 1*****left hand*63600200542967
Object No: 5*****Participants No : 1*****left hand*63600200546168
Object No: 5*****Participants No : 1*****left hand*63600200546266
Object No: 5*****Participants No : 1*****left hand*63600200546300
Object No: 5*****Participants No : 1*****left hand*63600200546372
Object No: 5*****Participants No : 1*****left hand*63600200546934
```

Figure 8. Lefthand and Object Interaction File Records

Note that all other files namely Outputleft 1, Outputleft2, Outputright, Outputright1, Outputright2 are recorded by the ATTAT based on same principles.

3.3 Pointingoout

In Figure 9, participant 3's pointing gesture records can be seen. Since the information only contain the number of participants and the system time, you may recognize that the application only has a one gesture that is namely pointing another person. As mentioned before if the experimenter wanted to used another gesture that records would probably have discriminative data such as the following lines:

3GestureDetected*waivingHand*635993456313995

3GestureDetected*holdingHead*635993456323995

Since the discrete event detection is also problematic for the gesture detection, similar procedure that mentioned on the section 3.2 can be used to discriminate the number of gesture.

```
3GestureDetected*63599356312005
3GestureDetected*63599356312035
3GestureDetected*63599356312071
3GestureDetected*63599356312103
3GestureDetected*63599356312137
3GestureDetected*63599356312170
3GestureDetected*63599356312205
```

Figure 9. Human-Human Interaction File Records

4. CHAPTER

CREATING NEW TARGET GESTURE

Within the application only one gesture has been created and coded to detect pointing gesture of the users. As mentioned before, an experimenter can increase the number of the gestures that need to be detected by the system.

The main purpose of the Chapter is, to give certain recommendation about gesture building and providing information about adaptation of built gesture to the system. In order to get detailed information about gesture building, following link below can be a proper source.

https://msdn.microsoft.com/en-us/library/dn785304.aspx

Following section introduces two vital recommendations about gesture building and the next section give detailed information about how to implement this gesture into code blocks.

4.1 Gesture Building

The first step of the building a gesture with VGB is, an experimenter needs to record a Kinect video where all the sensors of the Kinect have to capture the target gesture. It is important to note that, since building a gesture model is a machine learning approach or lets say, a model that need to be trained, not only the target gesture but also irrelevant gestures must be recorded as well. Another important point is, the number of person that is recorded by Kinect should be at least more than one that these persons have to be recorded not simultaneously but one by one. There are simple reasons to have irrelevant gestures and more than one person. Tagging the target gesture as TRUE is the vital issue but also representing the irrelevant gestures as FALSE also essential information for Microsoft Visual Gesture Builder (VGB). Additionally, tagging the target gesture on the video that consists more than one person, provide better representation of the target gesture model. In order to get detailed gesture building procedure, please see the Microsoft official site.

4.2 Adopting a Gesture

Since the gestures and their database created by the VGB, experimenter should have a number of names that represent the target gestures and a database file. Gesture database file extension must be “gbd” that can provide number of different modeled gestures. Generally, there are more than one gesture in database, it is recommended that name of the gestures should be kept by the experimenter. Simply gesture database file, which has “gbd” extension, must be placed in the Debug directory. If there is a necessity about the location of the database file, path of the location should be given in following format “@C:../…/…/mygestures.gbd/”. Simply, there are several steps to complete the adaptation. These steps are basic modification of the code lines. In order to adopt any other gesture, experimenter should follow the steps given below:

- Place the “gbd” file into Debug directory.
- Open application in Visual Studio 2013 (recommended)
- Click the GestureDetector.cs file in the solution explorer.
- See the GestureDetector class that has two members named as gestureDatabase and seatedGestureName, which are equal to our “gbd” file and the name of the target gesture.

- If there are more than one gesture, for each gesture, a read only private string variable need to be created and should be defined as the name of the other gestures.
- Find the line in the class constructer, named as “public GestureDetector”, which start with a definition of “using”.
- In the using definition, there is a “foreach” statement where an “if” statement was defined to check existence of the defined gesture. Note that, for each gesture, same if statements need to be created. And obviously, same line, under the “if” statement, have to be replicated for each novel if statement. Please recognize that in the if statement there is a definition of our target gesture that we defined it as a read only string member of the class. So, for each new gesture experimenter should change the parameter of the “if” statement, as a new gesture name such as “this.newgesture”. In the code block given below, an example is presented for “newgesture”.

```
using (VisualGestureBuilderDatabase database = new VisualGestureBuilderDatabase(this.gestureDatabase))
        {
                    // we could load all available gestures in the database with a call to vgbFrameSource.AddGestures(database.AvailableGestures),
                    // but for this program, we only want to track one discrete gesture from the database, so we'll load it by name
          foreach (Gesture gesture in database.AvailableGestures)
          {
            if (gesture.Name.Equals(this.seatedGestureName))
            {
              this.vgbFrameSource.AddGesture(gesture);
            }
            if (gesture.Name.Equals(this.newgesture))
            {
              this.vgbFrameSource.AddGesture(gesture);
            }

          }
        }
```

In this point, novel gesture database is ready to use and all gestures defined in the application. Nevertheless, there is one more additional procedure that have to be completed that result of the detected gesture have to be recorded. In order to complete the procedure, please follow the given steps below:

- Find the Reader_GestureFrameArrived method, which checks the incoming frame and gives response whether there is a modeled gesture or not.
- In this code block there is a foreach statement that controls each gesture in each incoming frame.
- Under foreach statement replicate and modify the “if” statement for any additional gesture. An example given below for “newgesture”:

```
foreach (Gesture gesture in this.vgbFrameSource.Gestures)
{
    if (gesture.Name.Equals(this.seatedGestureName) && gesture.GestureType == GestureType.Discrete)
      {
       DiscreteGestureResult result = null;
```

```
        discreteResults.TryGetValue(gesture, out result);

         if (result != null)
           {
            // update the GestureResultView object with new gesture result values
                        this.GestureResultView.UpdateGestureResult(this.vgbFrameSource.TrackingId,true,                result.Detected,
result.Confidence,1);
             }
       }
if (gesture.Name.Equals(this.newgesture) && gesture.GestureType == GestureType.Discrete)
       {
        DiscreteGestureResult result = null;
        discreteResults.TryGetValue(gesture, out result);

         if (result != null)
           {
            // update the GestureResultView object with new gesture result values
                        this.GestureResultView.UpdateGestureResult(this.vgbFrameSource.TrackingId,true,                result.Detected,
result.Confidence,2);
             }
       }

}
```

- Please note that, replication of the if statement has some exact modifications. The first modification is the name of the gesture changed as “this.newgesture” and the second is the last lines of the foreach statement that are different at the last parameters of the called functions. Recognize that the last parameters are different as 1 and 2. This numeric difference provides the differentiation of the detected gesture.

Now application is ready to tale log for different detected gestures. In the pointingout log file you can recognize that each recorded gesture represented as another number. As a last recommendation please take note the number that you defined, as the last parameter of the function given above, and the corresponded gesture name. For instance, the gesture named as “seatedGestureName” was paired with the number “1” and the gesture named as “newgesture” was paired with the number “2”.

5. CHAPTER

IMPLICATION

ATTAT has been programmed onto an application, which was published by the Microsoft Kinect SDK. In the SDK, an application named as Discrete Gesture Basics has been used. Mainly, modifications and additional lines are done in the MainWindow.cs file. All modified codes and additional code lines are well commented within the application. However, in order to reduce the complexity of the application main functionalities are explained and presented in diagrams. In the Diagram 1 you can see the main loop of the application. This loop controls whether a calibration file previously created or not. Please note that for any new position of Kinect or TUI table, application needs a new calibration file. If a calibration file exists then the application starts the experiment, otherwise application starts the calibration procedure.

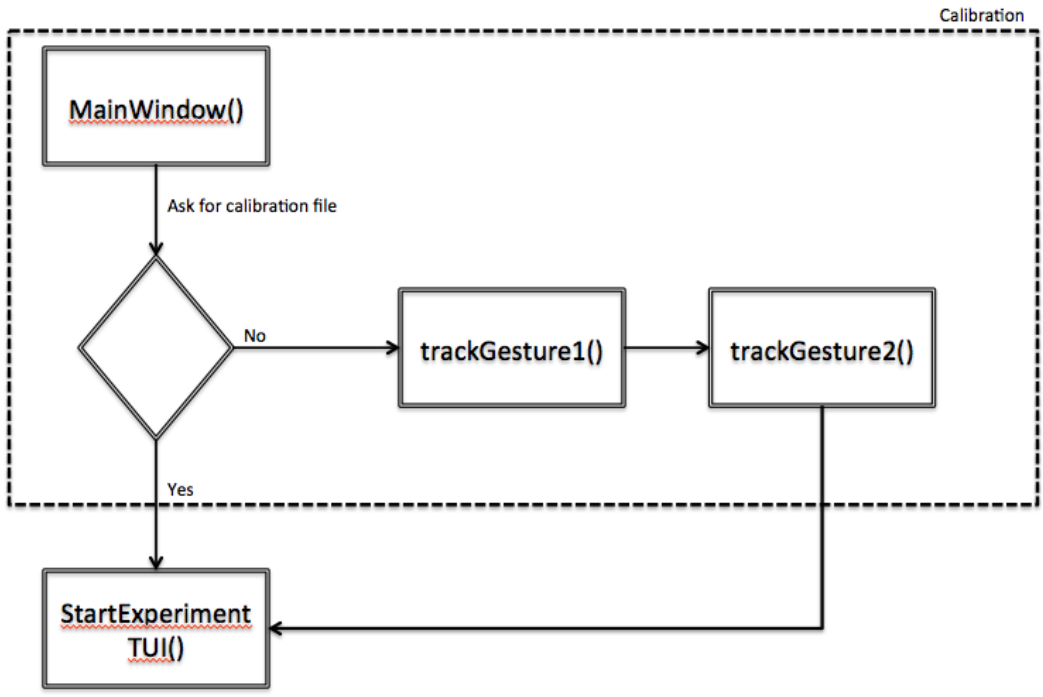

Diagram 1. Main Loop Data Flow Diagram

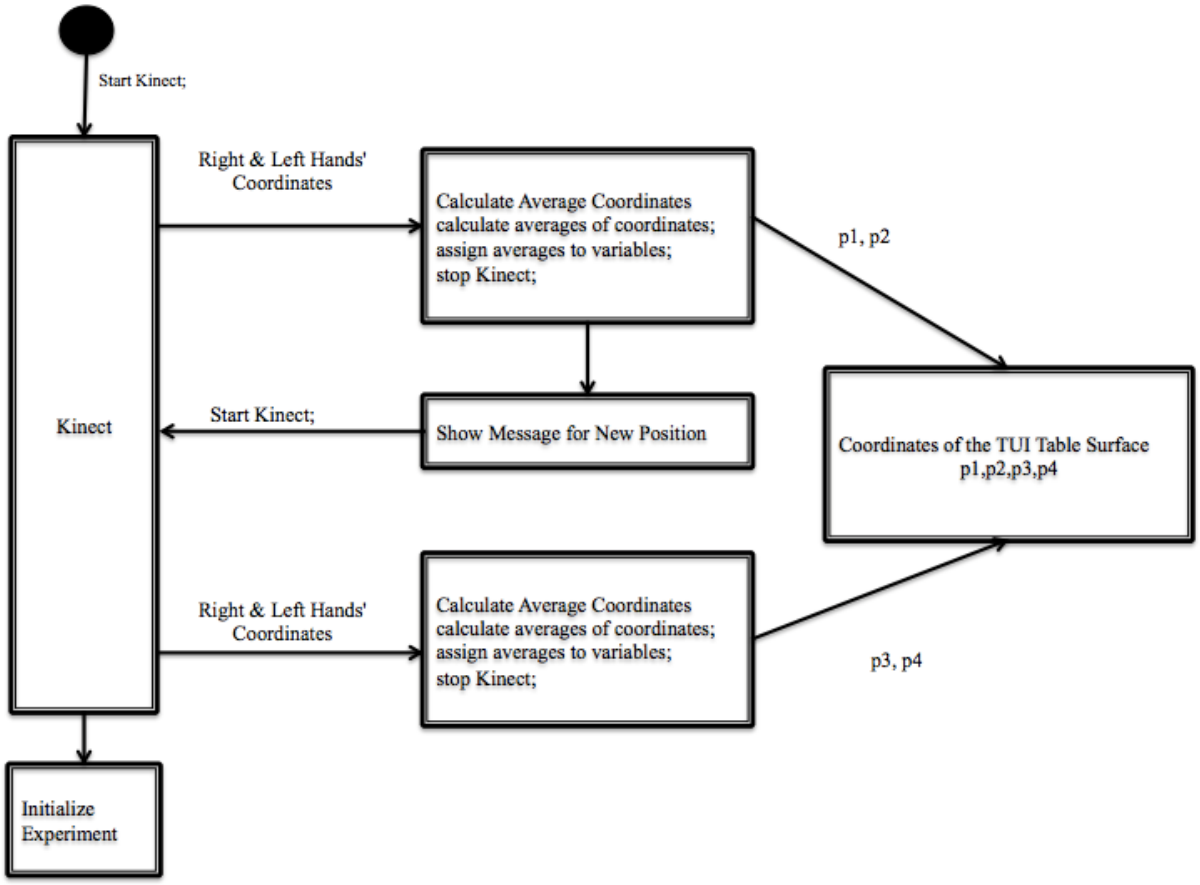

Diagram 2. Data Flow Diagram of Calibration Procedure

In the Diagram 2, calibration procedure can be seen that p1, p2, p3, p4 are the points that represent the corners of the TUI table. These points are essential to calculate tranformation constant that each incoming screen coordinates of the TUI objects are converted into 3D coordinates by simple matrix multiplication. Transformation data flow can be seen in the Diagram 3. In short, application initializes the Kinect sensor takes the first two corners location then warns user whenever the message box clicked as ok takes second two corners of the TUI table and then initialize the experiemental procedure.

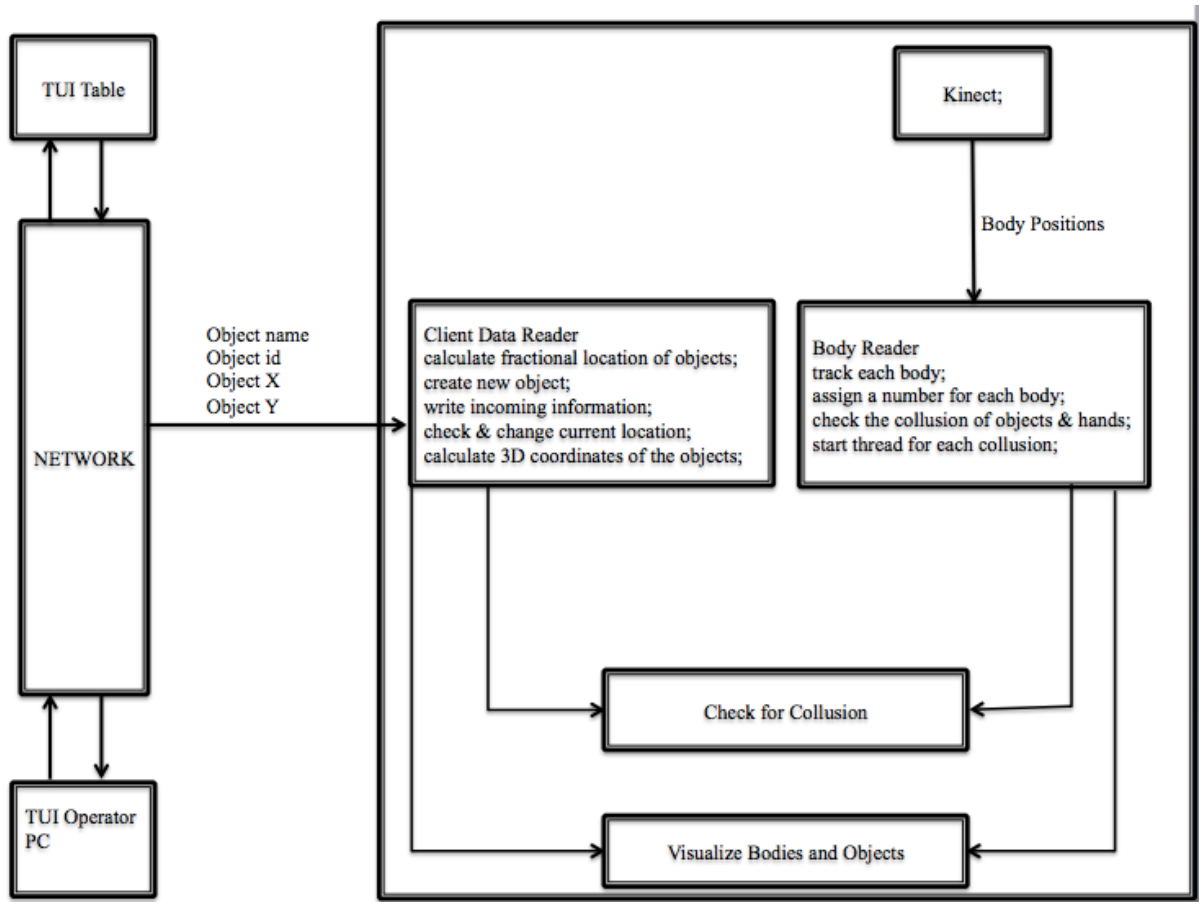

Diagram 3. Human-object interaction flow diagram.

The Diagram 3 presents the human object interaction functionality of the application. Briefly, objects' id, state and location are recieved from the UDP server-client architecture, which is sent by the TUI table. Incoming objects screen coordinates transformed into 3D physical coordinates system then each hand coordinates of participants analized whether any hand positions collide with any active objects location of TUI. Moreover, each calculated objects' 3D coordinates visualized by the application.

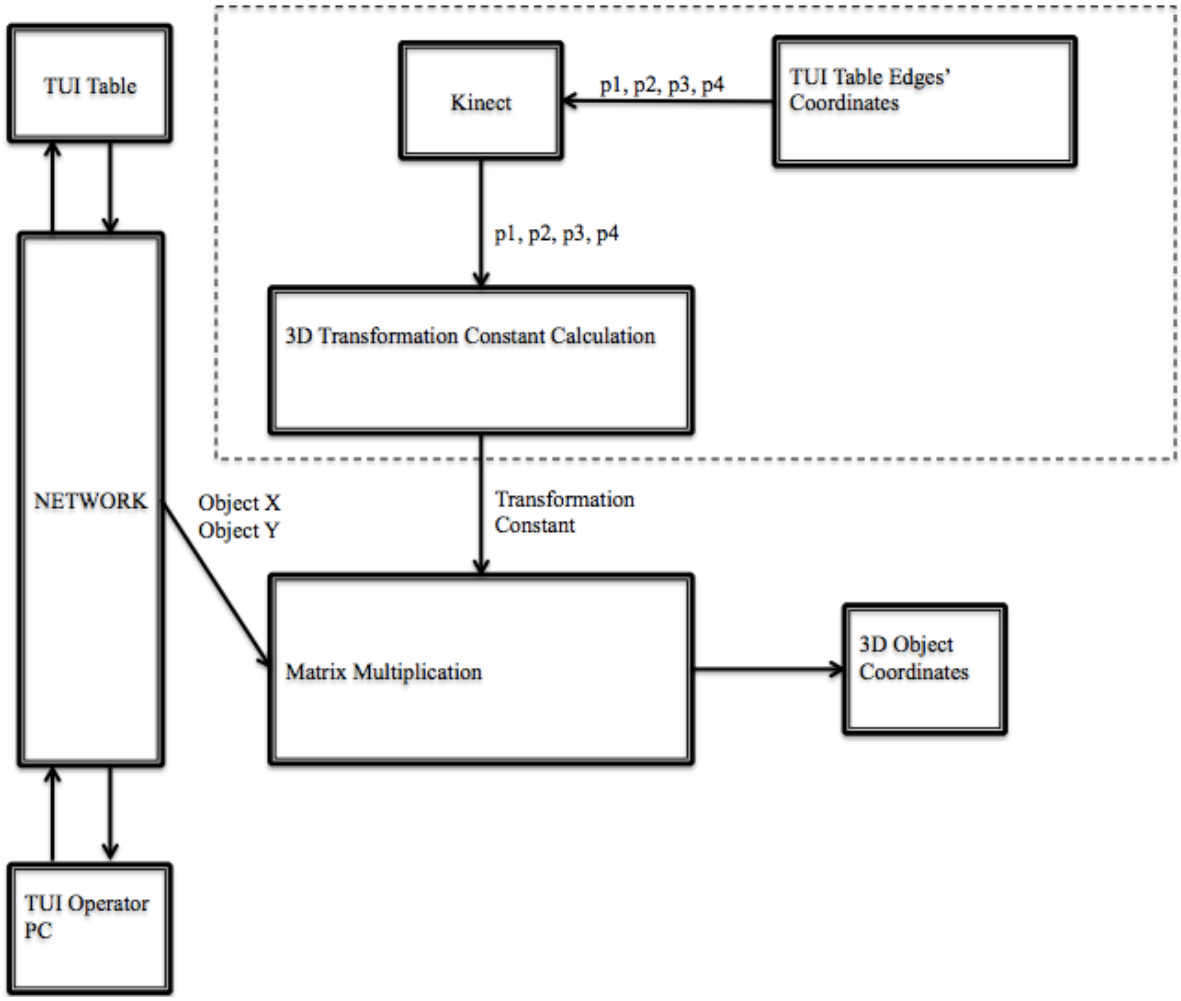

Diagram 4. Transformation Data Flow

In the Diagram 4, TUI objects' coordinate transformation can be seen. Please note that the dashed rectangle in the Diagram 4 represents the calibration procedure where the constant fort he transformation is ready to use. Since application has the constant, simple matrix multiplication is done to find each object's 3D coordinates. After locating each TUI object in 3D physical coordinate system, these locations are kept to analyze collusion of objects of TUI and tracked bodies' hands. You can see the interaction detection data flow in the Diagram 5.

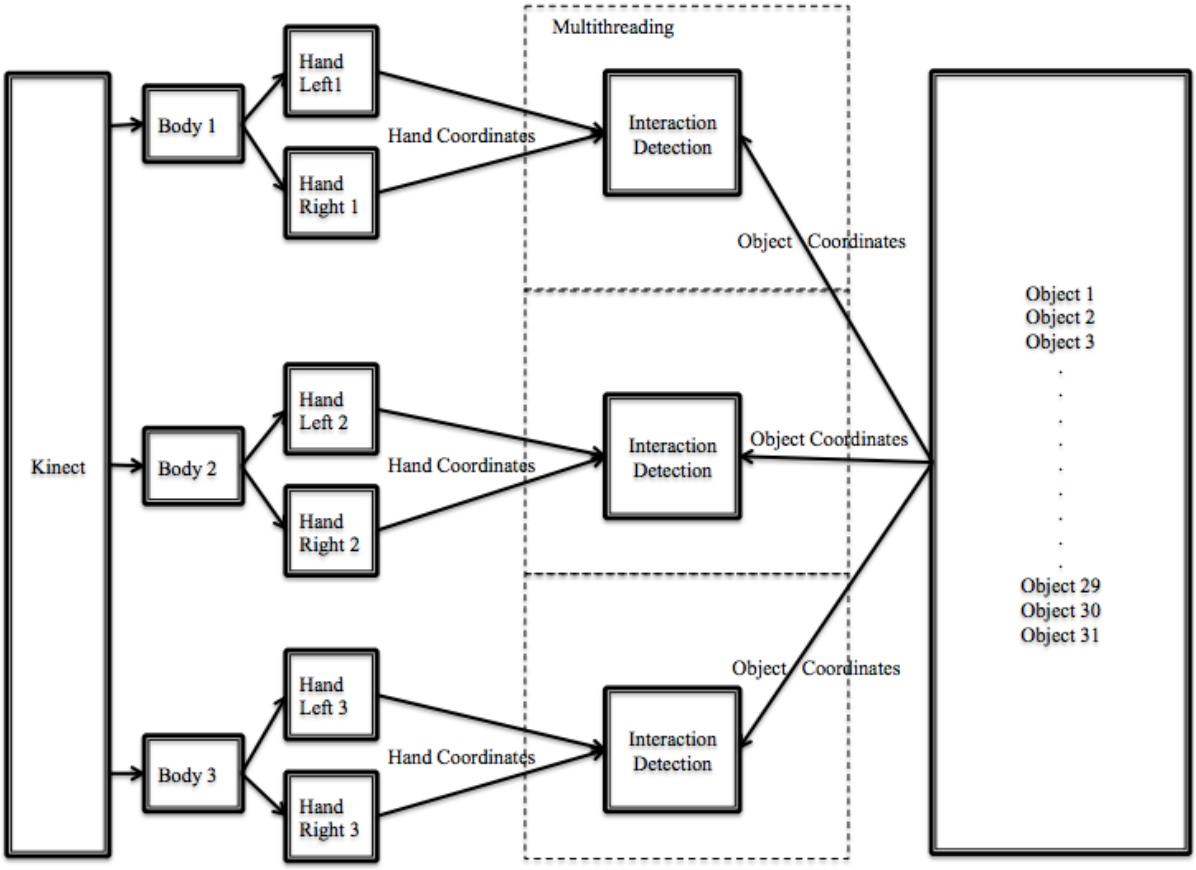

Diagram 5. Human-Object Interaction Detection Data Flow

In the Diagram 4, you can see that each hand coordinates of tracked bodies are taken into collusion detection procedure. For each hand coordinates are being checked by an simple locational collusion detection function within a sperate thread. Therefore, for each frame 3 different threads checking these hand coordinates whether they collide with any objects or not.

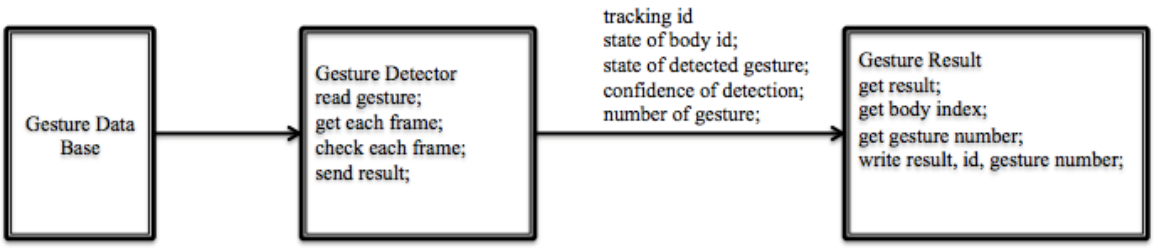

Diagram 6. Gesture Detection Data Flow Diagram

And the Diagram 6 represents the gesture detection data flow that each gesture represented with a single detector and for each body analyzed with this detector for each frame of Kinect.

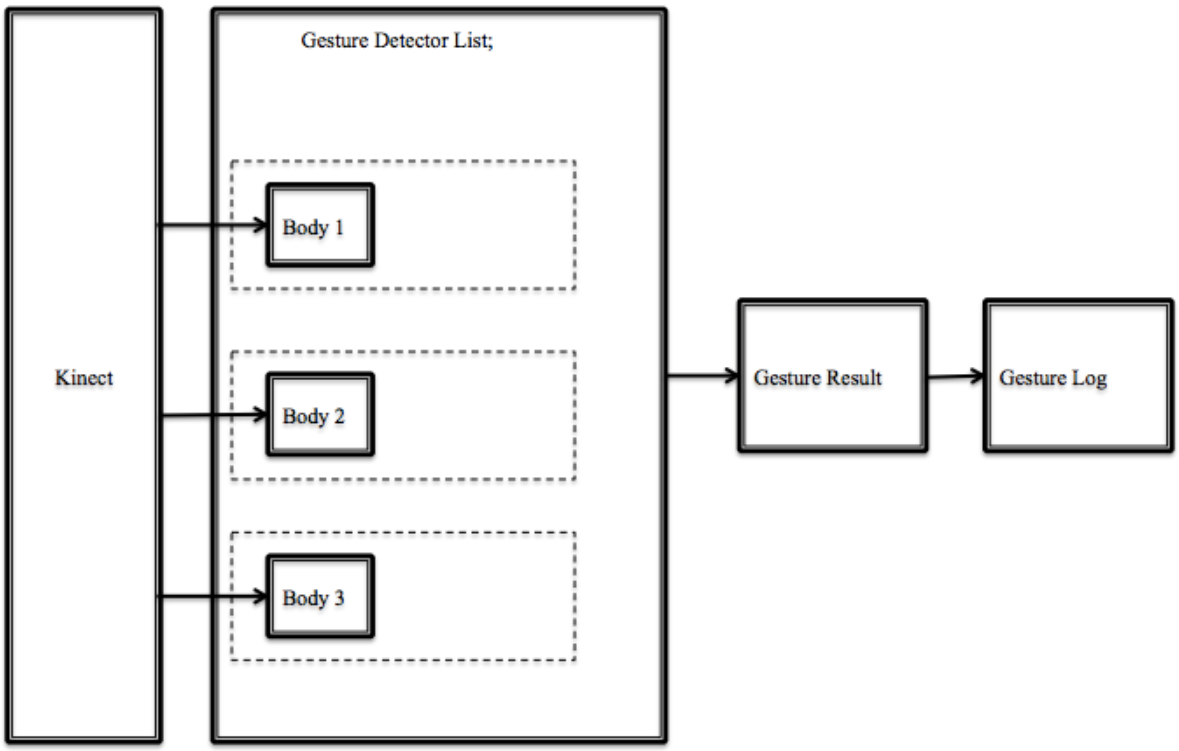

Diagram 7. Gesture Detection Data Flow

In the Diagram 7, you can see that each tracked body has its own gesture detector that is continuously detect for each frame of Kinect. And each frame evaluated and sent to the Gesture result. If any confidence of gesture higher than the predetermined confidence threshold then the application takes it record as a body id and gesture number. Please note that this confidence default value was determined as 0.7, which was the best confidence threshold fort the discrete gestures detection.