Bournemouth University

MSc Computer Animation and Visual Effects

3D motion capture of an Actor for Fall Detection and Health Monitoring

Milto Miltiadou

i7787126

That thesis was submitted to Bournemouth University.

August 2012

Abstract:

Many elderly people are hospitalised because they cannot autonomously live in their own homes due to the high risk of falls. Many researches attempted to solve the Fall Detection problem. The potential use of Kinect sensors may lead to a good markless Fall Detection System.

A scene was set and an actor was asked to performed actions, like sit on a chair and fall down. Using the iPi Studio and two Kinect sensors, .bvh motion capture files were generated. Those files are in the input files of the system.

In that paper two classification methods are proposed: a simple Classification using Euclidean Distance and a Bayesian Inference with Gaussian Functions. Both methods use statistical analysis of 3D training data sets to recognise abnormal activities, when inactivity is observed.

Convolution, low level filtering and buffers that return the average actions are used to filter and improve the quality of the classification results.

Acknowledgements:

I am sincerely grateful to the people who supported me while I was working on my major project and throughout the year. First of all I would like to thanks Dr. Hammadi Nait for the project idea and his guidance. I would also like to thanks Mr. Jon Macey for the demos, he provided to us and his enthusiasm to help us over the year. By the end, I am also grateful to Mathieu Sanchez for his lab support and Tim van Mourik who volunteering became the actor in that project. I am sure the project will not have been successful without their help.

Contents

List of Figures:

List of Tables:

1. Introduction:

1.1. Motivation

Many elderly people are hospitalised because they cannot autonomously live in their own homes due to the high risk of falling. Statistics proves that the number of elderly people is increasing, while approximately 30% of people, who are over 75 years old, fall at least once a year. It was also observed that 70% of accidental deaths of old people are caused by falls. The fall risk makes old people anxious and increases depression. An efficient fall detection system would not only improve the quality of elderly people's life, but it would also decrease the cost of Public Health Services. In other words, it would help the rehabilitation of those people in their homes. (Perolle, 2006)

1.2. Related Work

Many researches attempted to solve the Fall Detection problem. In 2005, McKenna S.J and Nait-Charif H. presented a method of detecting unusual activities of occupants in a home environment. RGB cameras were sets in a real home environment and an actor was asked to performed several actions, like sitting on a sofa. By observing the 2D trajectory of the moving pixels, a threshold of his speed was set to indicate inactivity. Gaussian Mixture Models and Maximum likelihood classification was used to build the system. Then the program recognises whether the person is an inactivity area or on an unusual place every time his speed drops significantly. The system was implemented in Matlab and it was proved to be vulnerable to shadows and lighting.

Similarly Rougier C. et al suggested an approach that observes the 3D trajectory of the head to distinguish falls from other activities. By using the algorithm of Dementhon and applying a particle filter, the system is able to track the 3D position of the head from a single camera. Then by observing the vertical and horizontal velocity, a threshold was set to distinguish falls.

Another related paper was published by Dai J in 2010. In that paper two mobile algorithms were proposed for fall detection. The first one observes the acceleration of its movements. The second algorithm uses a magnetic accessory to capture further information about the user's behaviour. Even though a good performance was achieved, it cannot be guaranteed that old people will not always carry their mobile phones with them.

Finally in 2011, Wang F analysis the gait cycles of humans and found out that the speed is the most robust factor in detecting abnormalities in elderly people's activities.

1.3. Aims and Objectives

The aim of this project is to develop a system that uses markless motion capture for health monitoring and fall detection. The system uses two Kinect sensors to capture the motion of an actor performing an amount of actions in a scene. The scene was set up in a studio and the motion capture data is taken using a

third-party software, due to the limited period of time. By observing the velocity that the actor is moved with, the system is able to recognise inactivity. Then by using statistical analysis of the joints' cloud, the system is able to recognise whether the occupant is in inactivity area or not. In our system possible falls are defined as inactivity of the actor at a non-inactivity area.

2. User Guide

The system is divided in two programs. From the main program the user is able to either generate training data for inactivity areas or watch the classification results of the actor's movements. The Sub-Program is a Cluster Manager used to read the training data sets and output the mean vectors and covariance matrices of them.

2.1 Main Program:

Once the main program is loaded the user have to load the .bvh file from the GUI. This is compulsory for the program to run:

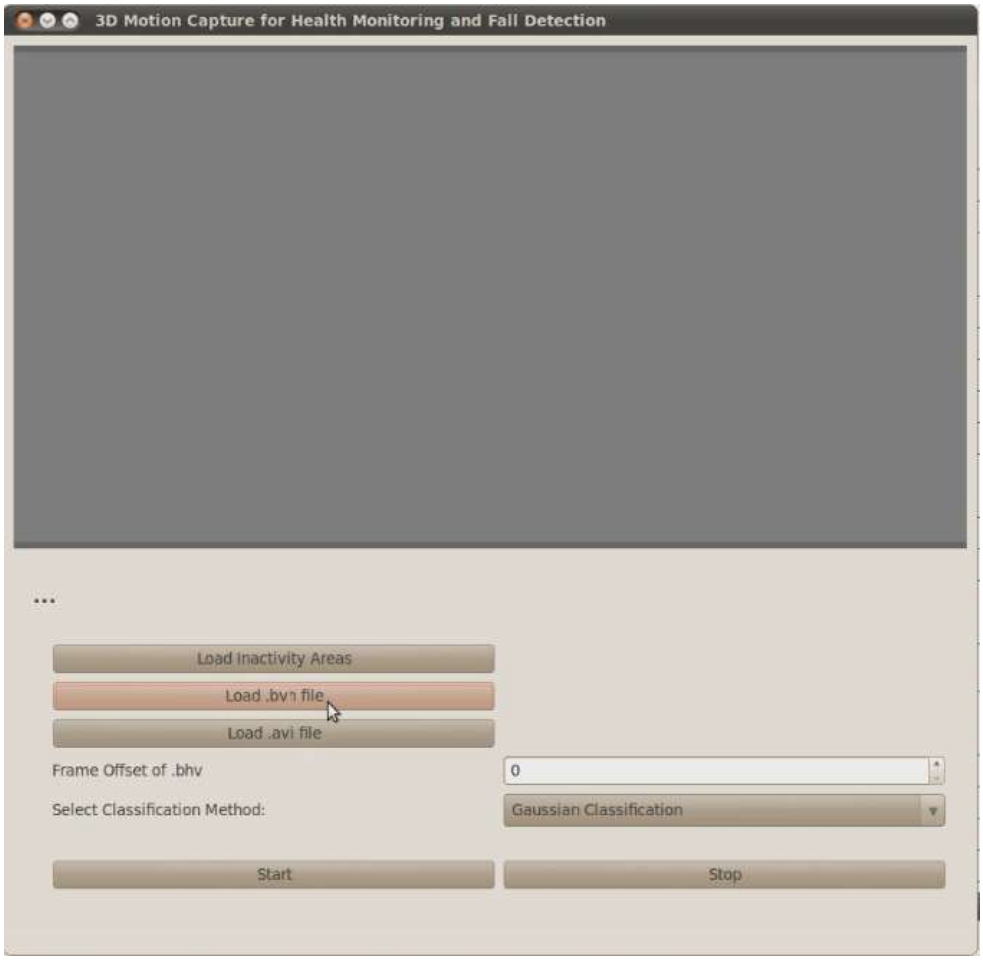

Figure 1: Loading a .bvh file

The Inactivity Areas file should be loaded to the main application, before the classification starts. If no file is loaded, then the system stills runs but in a mode that only training data can be generated.

The training data is generated using the following Keyboard buttons:

L : Loads training data from files. In case, the user already has saved a training data set beforehand but he wants to add more.

A: Adds the positions of the joints during the current frame to the training data.

W: Export the training data.

The following image shows how the program is switched to the data generator:

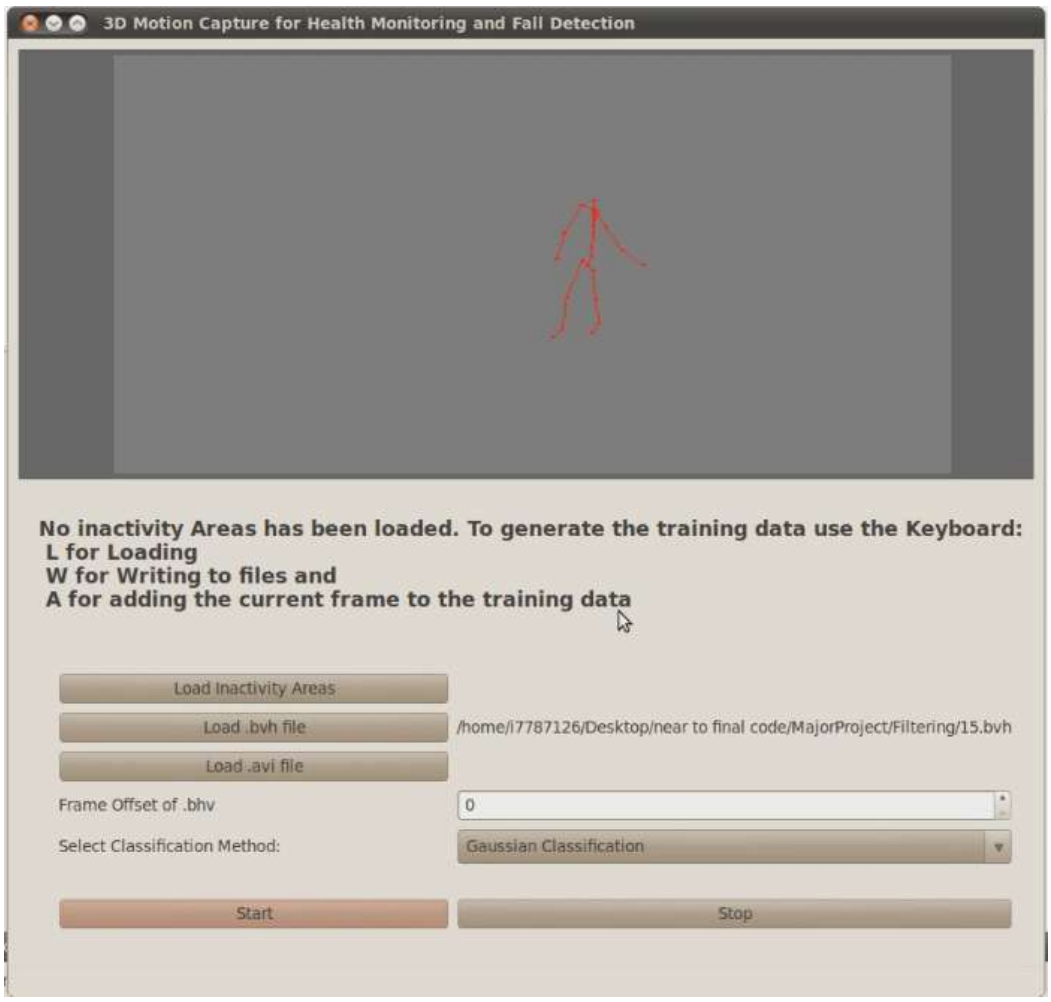

Figure 2: Training Data Generator Mode

The Inactivity areas file should be loaded before the classification is started. That file defines the number of the inactivity areas and it is generated using the sub-program as explained in Section 2.2.

It is not compulsory to load a video and the video does not affect the classification results. But the video makes the application friendlier to the user. The quality of the videos is not good because the depth information captured by the Kinect sensors is in front of the rgb information. The user can swap the video plane with the mocap plane by either clicking on the GL Window or by pressing S from the Keyboard. In addition, the mocap files, which were exported from the iPi Studio, do not start the same time with the videos. For sychronisation purposes, the application allows the user to add an offset value from the GUI. Those values are given in Table 3.

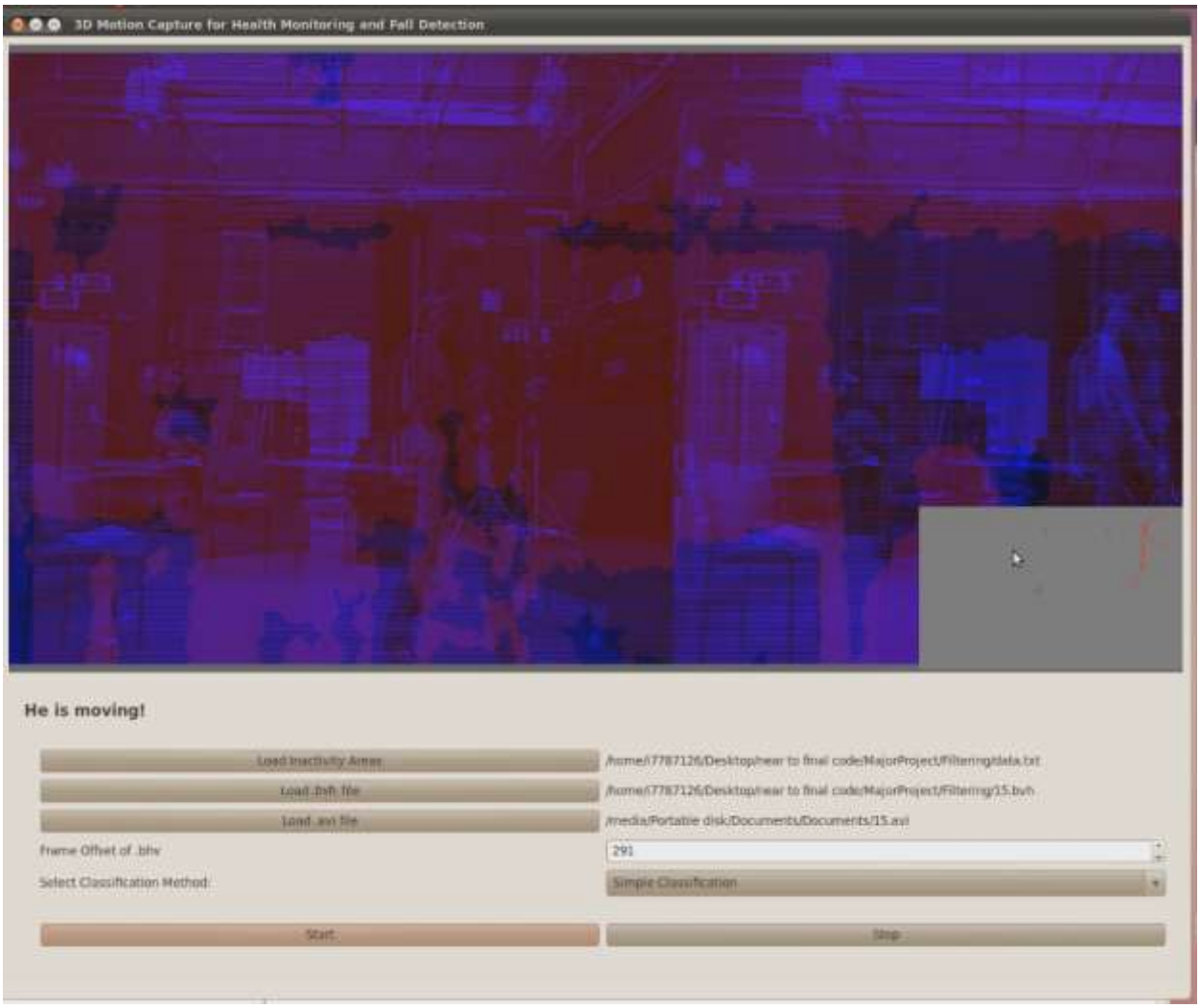

Figure 3: GUI with video Loaded

The classification results are shown on a label in the GUI. For example on Figure 3, the action is moving and the corresponding message appears on the screen.

When the mocap file reach to the end, the program automatically restarts the animation and the classification of data. The system also allows the user to stop the video, mocap visualization and classification of the actor's movements by pressing the button Stop. In addition, if the user loads another .bvh file, while the classification is enable the program continues to run with the first file that has been

loaded until the start button is pressed. Then that file is replaced with the .bvh file that has been loaded last.

2.2 Sub-Program:

The sub-Program takes as input the training data sets that were exported from the main program and generates a single file with the number of clusters and their Mean Vectors and Covariance Matrices.

The program allows the user to select the number of clusters. The current Cluster is the number of the cluster that any modification is applied to. The default value of the Mean is the centre of the coordinate system and the default matrix of the Covariance Matrix is the identity matrix. Each training data sets consists of three files, one for each dimension. Those files must have the same number of values.

Figure 4: Sub –Program, Cluster Manager

When the calculate Results button is pushed, the mean vector and Covariance Matrix are shown on the screen. If the Export button is pushed, then a Save Dialog Box pop up and it asks the user to save those values in a single file, which is required for the classification in the main application.

The 1D tab of the program is an extra feature and it used to create histograms. The histograms indicate the frequency of each point in the training data set.

Figure 5: Sub –Program, Cluster Manager

It is worth mentioning, that the min and max values defines the interval the input data belongs to.

The step value is used to round the numbers. For example, if
a point (P) = 1.3345
min = 0
max = 4
step = 0.1
then P belongs to the 13th interval, (1.3,1.4].

3. Getting the mocap data

The project is divided in two parts: getting the Data from the Kinect Sensors and classifying the movements of the actor. The main focus of the project is the classification of the data. This section explains the first part of the project.

Getting the data was achieved in two stages:

1. Recording the videos in a studio using the iPi Recorder and two Kinect Sensors
2. Getting the rigging information/mocap using the iPi Studio

3.1. Recording the videos:

3.1.1. Setting the scene:

The following image shows the scene, set up for recording.

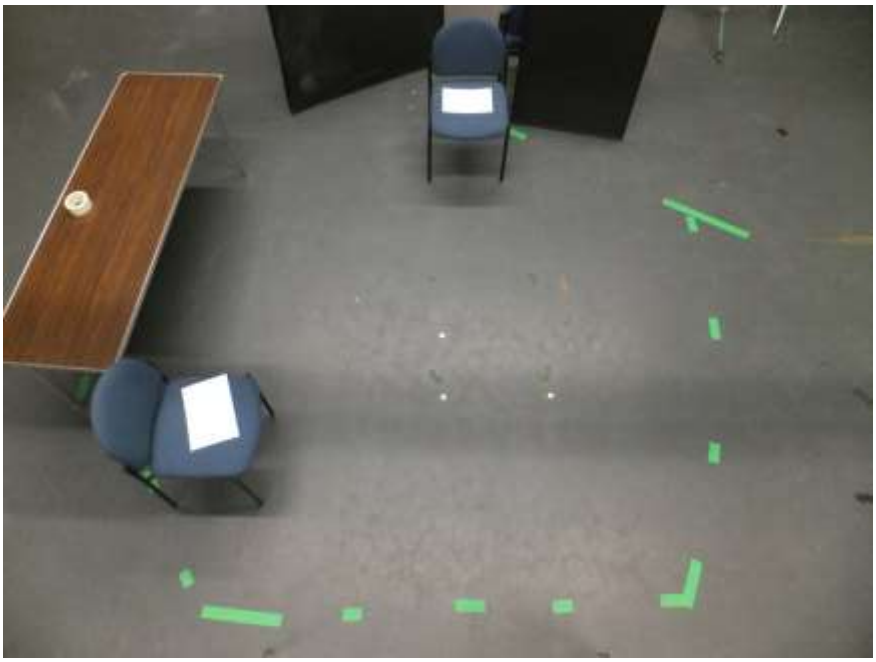

Figure 6: The scene.

The green tapes show the area that was captured by both Kinect sensors. There are two entry points and three inactivity areas (chair A, chair B and phone position, next to the table). The iPi Studio requires 30 frames at the beginning of each video captured to process the background scene. For that reason, the black cloths are positioned next to the entrances, such that the actor can hide himself before he enters the scene.

3.1.2. Actors' Movements

In McKenna J.'s and Nait-Charif H.'s project, the actions of the actor were labeled. Similarly the actions that the actor was asked to perform are numbered as follow:

1 2 3

4 5 6

7 8 9

10 11 12

Figure 7: The actions the actor performed.

Because iPi Desktop Motion Capture is not stable, the 12th action was added in case the program was not able to start the tracking the motion of the actor. The program classifies that action as an abnormal activity, because the velocity of the actor is low and he is not in an inactivity area. Table 1 verbally explains the 12 actions.

Number of Action	Action Performed
1	Enter from entrance A
2	Exit from entrance A
3	Enter from entrance B
4	Exit from entrance B
5	Sit on chair A
6	Sit on chair B
7	Answer the telephone
8	Walk in the room
9	Fall down
10	Start limping with the left leg
11	Start limping with the right leg
12	Skeleton pose

Table 1: The actions that the actor was asked to perform and there numbers.

Sixteen videos of duration 50-120 seconds each have been recorder. The actor was asked to perform actions from Table 1. The actions, captured during each recording, are shown on the following table:

Number of Video	Actions Performed
1	1, 8, 7, 5, 6, 4
2	3, 5, 8, 9, 2
3	1, 12, 7, 6, 5, 8, 9, 2
4	1, 8, 5, 7, 5, 4
5	1, 12, 8, 10, 5, 8, 4
6	1, 6, 12, 8, 7, 8, 2
7	1, 8, 4
8	1, 11, 5
9	1, 7, 6, 12, 8, 7, 4
10	1, 6, 7, 8, 5, 9, 12
11	1, 12, 5, 8, 11, 9
12	1, 12, 8
13	1, 12, 8, 9
14	1, 12, 7, 8, 10, 2
15	3, 12, 8, 5, 6, 7, 8, 9, 11, 2
16	1, 12, 7, 5, 8, 6, 4

Table 2: The numbers of the actions performed on each video recorded.

For example in video 1 the actor was asked to perform 1,8,7,5,6 and 4 actions. This implies that he entered the scene from entrance A, he walked in the room for a while, he answered the telephone, he sat on chair A, then he sat on chair B and he finally exited the scene from entrance B.

3.2. Getting the mocap Data using the iPi Studio

The iPi Studio is a markless motion capture technology. The iPi Studio is able to read the videos captured by multiple cameras, like MS Kinect. At first the scene has to be setted and the position of the rig should manually be initialised. Then the program is able to process the input video and track the movements of the actor. In case the tracking goes wrong the program allows the user to pause the tracking and fix the position of the rig. When processing is done the mocap data can be exported as .bvh files. Those files are the input data of the system proposed on that paper.

3.2.1. Limitations of iPi Studio:

iPi Studio is not accurate and stable. Even though efforts are made to improve the output, sometimes the problem cannot be fixed. The problems mainly occur on the hands' movements. Examples of those occasions are shown below:

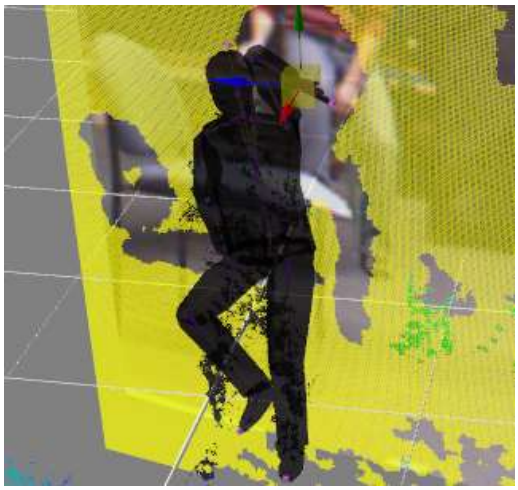

Figure 8: Failures occur while tracking the motion of the actor using iPi Studio

Since the data is noisy, the difficulty of classifying the movements of the actor is increased while the accuracy of the system is decreased.

Another limitation of the iPi Studio is the fact that it needs 30 frames at the beginning of each video fir calibration purposes. For that reason the actor was hiding before each time the recording was started. In addition, motion capture fails at the beginning of each video; if the actor is not in the scene the rig does abnormal movements, then he starts flying and he disappears from the scene.

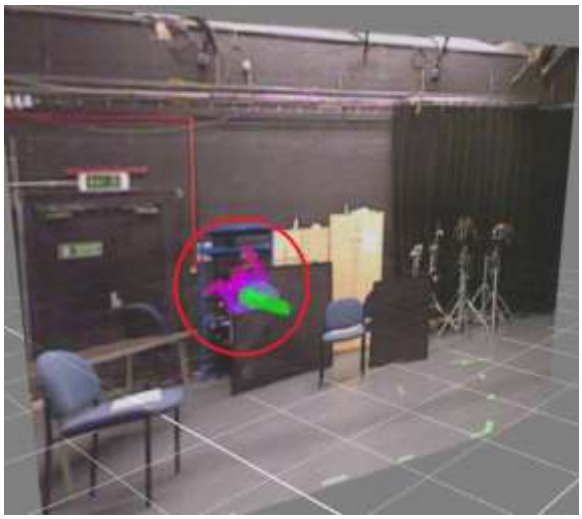

Figure 9: Abnormal movement of the actor

For that reason, the .bvh files exported starts with an offset. For example, if offset = 379 then the 1st frame of the mocap is the actually the frame 379 of the video captured from the two Kinect sensors. The offset of each mocap is shown on the following table:

Number of Video	Mocap Offset
1	379
2	
3	254
4	223
5	229
6	250
7	123
8	321
9	223
10	314
11	233
12	269
13	230
14	322
15	291
16	275
17	177

Table 3: The mocap offset values

The program synchronises the video with the mocap when the user adds the corresponding offset from the GUI.

4. Implementation Details and Technical Aspects of the System:

To increase the robustness of the application, the system is divided in two parts, the main application and a sub application. The main application can either be used to create training data or to classify the movements of the actor. The training data for each inactivity area is a cluster of points and it can be generated using the keyboard. The sub application reads a number of Clusters and exports a single file with the mean vectors and covariance matrices of all the clusters. Once that file is generates, the main application can be used to classify the movements of the actor. The classifier is adjusted according to that single file. As a result, there is no limit to the number of inactivity areas that may exist and the program does not have to load the entire training data sets each time it starts.

The following figure shows how the data flows inside our system.

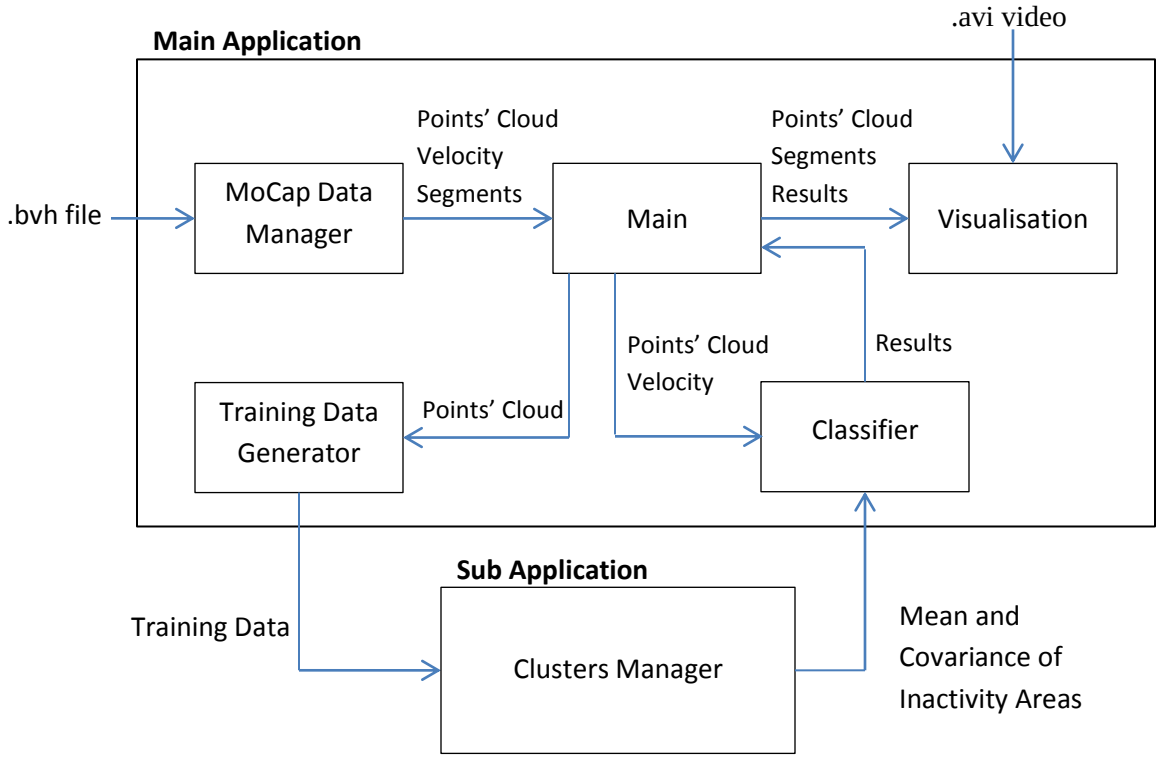

Figure 10: Outline of Design and Data flow

In this section, the technical aspects and implementation details of the systems are explained. That includes how the mocap data is managed, classification details, visualisation and clusters manager. An extensive User Guide of the system is available in Section 2 and a complete class diagram can be found in Appendix B.

4.1. Managing the MoCap data.

4.1.2. Parsing .bvh files

Using the iPi Studio as explained in Chamberlain C.'s report, a .bhv file can be exported for each video. Those .bhv files are imported into our system and used to train and test the program.

BVH stands for Bivision hierarchical data and it was developed by a mocap services company named Bivision, in order to be able to share mocap data. The file is divided in two parts, a section that describes hierarchy of the joints and another section that defines the local rotation and local offset (usually only the rot node has a local offset) of each joint at every frame. The following figure shows the joints and hierarchy information a .bvh file.

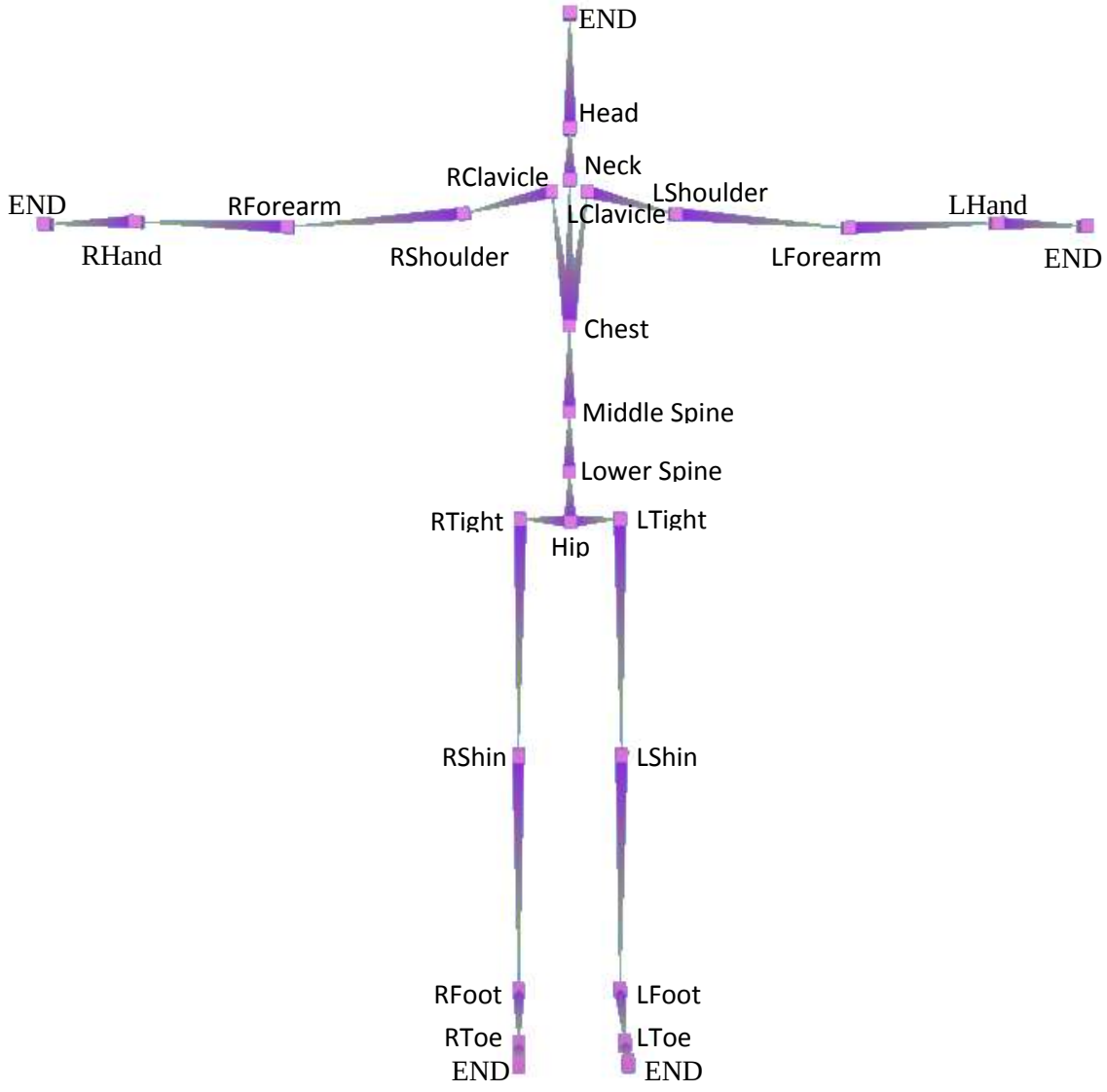

Figure 11: The Hierarchy of the Joints

A tree structure is used to represent the hierarchical linkages. The root node is the Hip and the arcs represent connections between the joints (Parent 2008). The .bvh files are parsed inside the bvh class and the mocap data is saved into the Rig class. The bvh class is a modification version of the bioviewer software published by Licas Walter in 2002.

The following diagram shows how the hierarchy information of Figure 11 is structured inside our system:

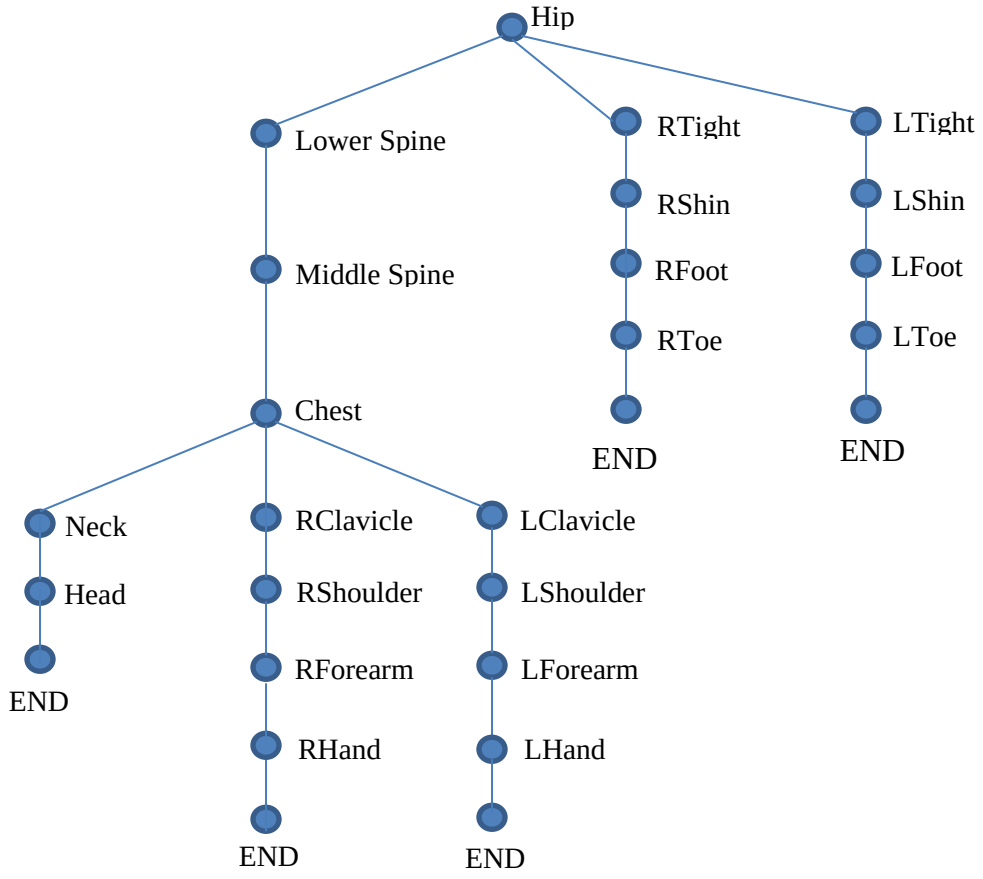

Figure 12: The tree structure of Figure 6

The system assumes that the root node is always the hip to simplify the calculations of the velocity as explained in section 4.1.3.

In addition the .bvh files define only the local transformation of the joints. The global transformation need to be calculated to find the actual positions of every joint per frame. Forward Kinematics is the solution to that problem. More details on that area are given in the following section.

4.1.2. Interpreting the data:

At first the local transformation is computed. The translation is equal to the offset, as it is defined in the hierarchy section. If the joint also have an offset parameter then this is also added to the local transformation. Usually, this only applies to the root node. The rotation matrix is created by concatenate three separate rotation matrices, one of each coordinate:

$$R = YXZ$$

By adding the translation to the final rotation matrix, the local transformation M is created. Then the global transformation of the joint is created by concatenate the local transformation its ancestors.

$$M = Mchild * Mparent * Mgrandparent \ldots$$

(Biovision BHV)

4.1.3. Velocity

The velocity of the actor is calculated by observing the positions of the root note. Here it worths mentioning that we assume that the root is always the first joint of the array. The velocity is equal to the distance between the original position of the actor and his final position over the period of time he spent to travel from one position to another.

$$\mathrm{V} = \frac{\Delta x}{\Delta t}$$

Δx is the Euclidean distance between two positions, e.g A(x1,y1,z1) and B(x2,y2,z2), and is calculated as follow:

$$\Delta x = \sqrt{(\mathrm{x2} - \mathrm{x1})^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

(codasign, 2011)

Δt is the time he spend to travel from A to B. In our case is equal to 5 frames. Due to the amount of noise, the robustness of the classifier was significantly decreased when Δt was set to 1 frame.

4.1.4. Convolution:

Convolution is used in signal processing to blend two functions, one shifter over another.

The equation of the 1D discrete version of convolution is defined as follow:

$$g(x) = \sum_{m=-s}^{s} f(x - m)h(m), \quad e.g\ s = 1,2, \ldots, n$$

where f is a signal and h is the convolution filter/kernel. The result $g(x)$ is actually a modification of the original signal $f(x)$.

(Laurenson et al, 2001)

In our case, convolution is applied to reduce the noise of the mocap. The original signal is the velocity of the hip joint. The filter h used is the following:

$\frac{1}{9}$ | 1 | 2 | 3 | 2 | 1 |

$\frac{1}{9}$ is the normalisation factor, such that the interval values of $f(x)$ are approximately reserved in $g(x)$, which is actually a smoothed version of $f(x)$.

A simple graph generator class was implemented and used to visualise the effect of convolution. In the x-axis every pixel value corresponds to a frame and the y-axis shows the position of the hip during a frame. The following graphs show the velocity of the hip joint during a few frames:

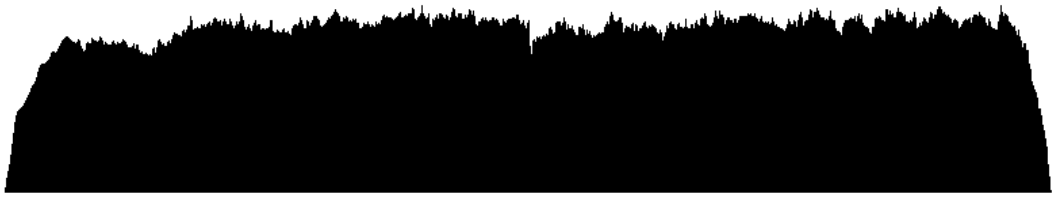

Figure 13: a part of $f(x)$, the original signal

Figure14: a part of g(x), the smoothed signal

The first graph depicts the original velocity of the hip. The second graph shows how the signal is modified when the convolution is applied. It is noticeable that in the noise is reduced and the curve smoother.

4.2. Classification of Data

We aimed to real-time classification, because in the future the system may be connected with a real-time motion capture tool. The classifier takes as input a Points Cloud, the positions of the joints, and the velocity of the actor during the current frame. The output of the classifier is the name of an action.

The system is able to recognise three different activities of the actor:

1. Moving: the actor is moving
2. Inactivity: the actor is not moving but he is in an inactivity area.
3. Abnormal: the actor is not moving and he is not in an inactivity area, possible fall.

The following Flow Chart shows the process of classifying the movements of the actor:

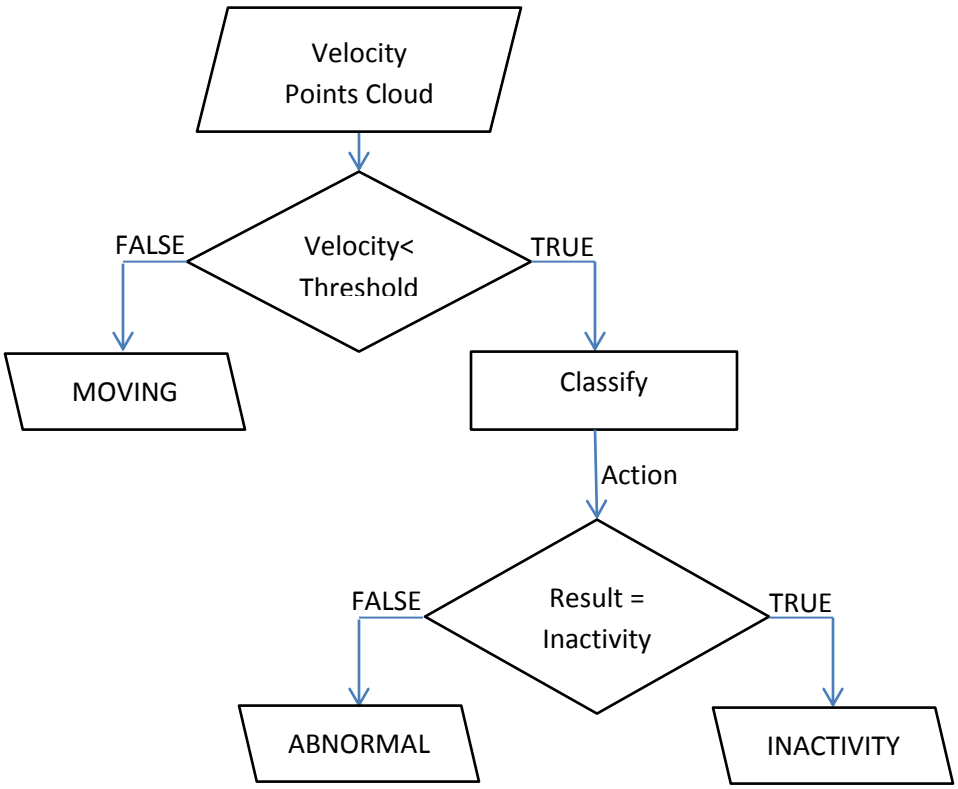

Figure 15: Classification Flow Chart

Two major implementation decisions were made in that stage. As explained in section 4.3.2 the classification part of the system is done in real time, but for filtering purposes the results that are shown on the screen have actually been calculated a few frames earlier. In addition, in order to increase the robustness of the system, before the classification is started, a file with the mean vectors and covariance matrices of all the inactivity areas should be loaded from the GUI. That file is generated using the sub-program of the system (see Clusters' Manager at section 4.5). This way, not only the process is speed up, but also there is no limit to the number of inactivity areas that may exist. Therefore the system should be used easily modified and used for a different scene and maybe a real living environment.

Furthermore, to increase the robustness of the system the only assumption made about the Points Cloud is that the first element is always the root node. Discarding the points of the hands, where the most noise occurs in the mocap was considered to improve the quality of the classifier. But it would have decreased the robustness of the system, so it was avoided. As a result, by just changing the threshold values of the classifier, the system would be able to parse and classify various kinds of hierarchy of the joints.

A threshold was set to distinguish inactivity and movement. Every person's gait is different, so that threshold may change if the system is used for another person. If the velocity is greater than that threshold then the person is moving, else he is not moving. If he is not moving then there are two possible cases: he is either at an inactivity area (e.g. sitting on a chair) or he is doing something abnormal (e.g standing in the middle of ther room, fall). Two classification methods have been implemented to distinguish those two activities:

1. A simple Classification with Euclidean Distance
2. A Bayesian inference with Gaussian Functions

Both Classifiers seems work, but the 1st method proved to be more robust during the Evaluation. In addition, all the threshold values were set by observing only video 15. Therefore, the results may have possibly been better if more training data had been used.

4.2.1 Simple Classifier

That classification method is relatively simple. The system calculates the Euclidean distances between the positions of the joints during the current frame and the mean vector of each inactivity area. Then, the average distance of the points cloud (the joints of the rig) from each inactivity area is calculated. For example if three inactivity areas exist then the system will calculate three average distances. If one of those distances is smaller than the threshold value then the person is in an inactivity area, else he is in another place inside the scene. Therefore, the average distance from one the inactivity area with mean vector M is calculated as follow:

$$A = \frac{1}{\mathrm{N}} \sum_{i=1}^{N} \left(\sqrt{(\mathrm{Pxi} - \mathrm{Mx})^2 + (Pyi - My)^2 + (Pzi - Mz)^2} \right),$$

where A is the average distance, Pi is the position of the i^{th} joint and N is the number of joints.

4.2.2. Bayesian inference with Gaussian Functions

A statistical classifier has also been implemented. For each training data set, the covariance Matrix and Mean vector are calculated using the sub-program of our system (Section 4.5) and exported in a single file. The program read those file and generates a multi-dimensional Gaussian Function for each inactivity area. The formula of a Gaussian function is the following:

$$p(x) = \frac{1}{2\pi\sqrt{||C||}} \exp(-\frac{1}{2}(X-\mu)^T C^{-1}(X-\mu) \ ,$$

where μ is the mean vector and C the covariance matrix.

(Moore, 2006)

Once the Gaussian functions are defined, the location of the actor can be guessed using the Bayesian inference:

$$P(A|x) = \frac{P(x|A)P(A)}{P(X)}$$

P(A|x) is the probability of the actor to be in inactivity area A during frame x and it is given by a likelihood function. In that project the likelihood function is a Gaussian density function. P(X) is the probability of that frame and it can be ignored because every frame is unique. P(A) is the probability of the actor to be in that area and it is also ignored because we assume that it is the same with all the other inactivity areas. Therefore the probability of the action to be in inactivity area A is equal to the Gaussian function of inactivity area A.

(Vose Software, 2007)

Since we are interested in deciding whether the actor is one of those areas or not, a threshold was set. During the classification, the probabilities of a given point to belong to each one of the inactivity areas are calculated using the Gaussians functions. For example, if three inactivity areas exist, then three probabilities will be calculated. If all those probabilities are all less than the threshold then the systems returns that the actor is not in an inactivity area. That threshold is only relevant to our scene and it is subject to change if a new scene is set.

Several tests have been done until the final decision on which points should be included into the calculations. An interesting approach was tested: the use of two Gaussian Functions per inactivity area. We defined robust joints to be the root joint and its children because they seem to be less noisy. Two data sets were generated for each inactivity area, one with all the positions of the joints and one with only the positions of the robust joints. Using those data sets, two Gaussian functions were defined per inactivity area. By the end only the one Gaussian function was used for each inactivity area and only the root joint is classified in that method, because it proved to be more stable this way.

4.3 Filtering

As mentioned before the motion capture is very noisy. Even though convolution (section 4.1.1) significantly decreased noise, more filtering was required to get good results. Two methods have been used for that purpose:

1. Using bitwise operations, the last 32 actions are saved in a buffer and the average action is used in the calculations.
2. Removing all the actions that have a very short duration.

More information about those methods is given below and the results of filtering are shown in section 4.3.3.

4.3.1 Buffer:

It is used to keep track of the movements of the actor. During each frame an action is added to the buffer and the average action of last 32 frames is used in the calculations. The buffer is an unsigned int, therefore 32 actions are saved as bit values.

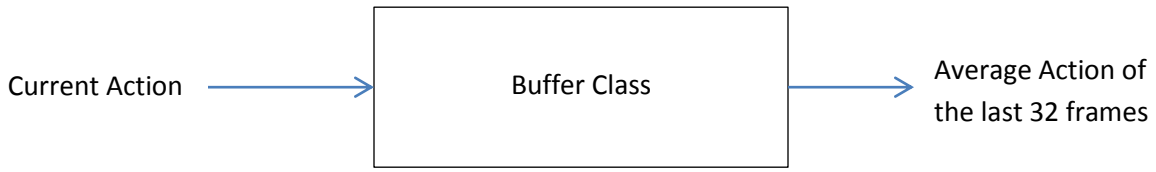

Figure 16: Buffer

As shown in the previous figure, the buffer class has only two function: addAction, which takes as input a Boolean value and getAverageAction, which returns a Boolean value. Bitwise operations are used to simplify calculations and to reduce the memory required to save 32 actions. The pseudocodes of those functions are the following:

Function addAction(Action):

```
Buffer = Buffer SHL 1 + Action
```

Function getAverageAction(ComparisonParameter):

```
Sum ← 0
for i ← 0  to  i < 32 step 1 do
    Sum ← Sum + Buffer AND (1 SHR 1)) SHL i
if Sum > ComparisonParameter
    return 1
else
    return 0
```

There are two buffers in our system. The first buffer is the velocity buffer, (0 for moving and 1 for inactivity), and the second one is the actions buffer (0 for abnormal activity and 1 inactivity in an inactivity area). The initial value of the actions buffer is set to inactivity. As a result the abnormal activity detection is usually delayed. That was observed in the evaluation of the classification results (see Appendix A).

The following flow chart illustrates how the classifier was modified when the buffers were added.

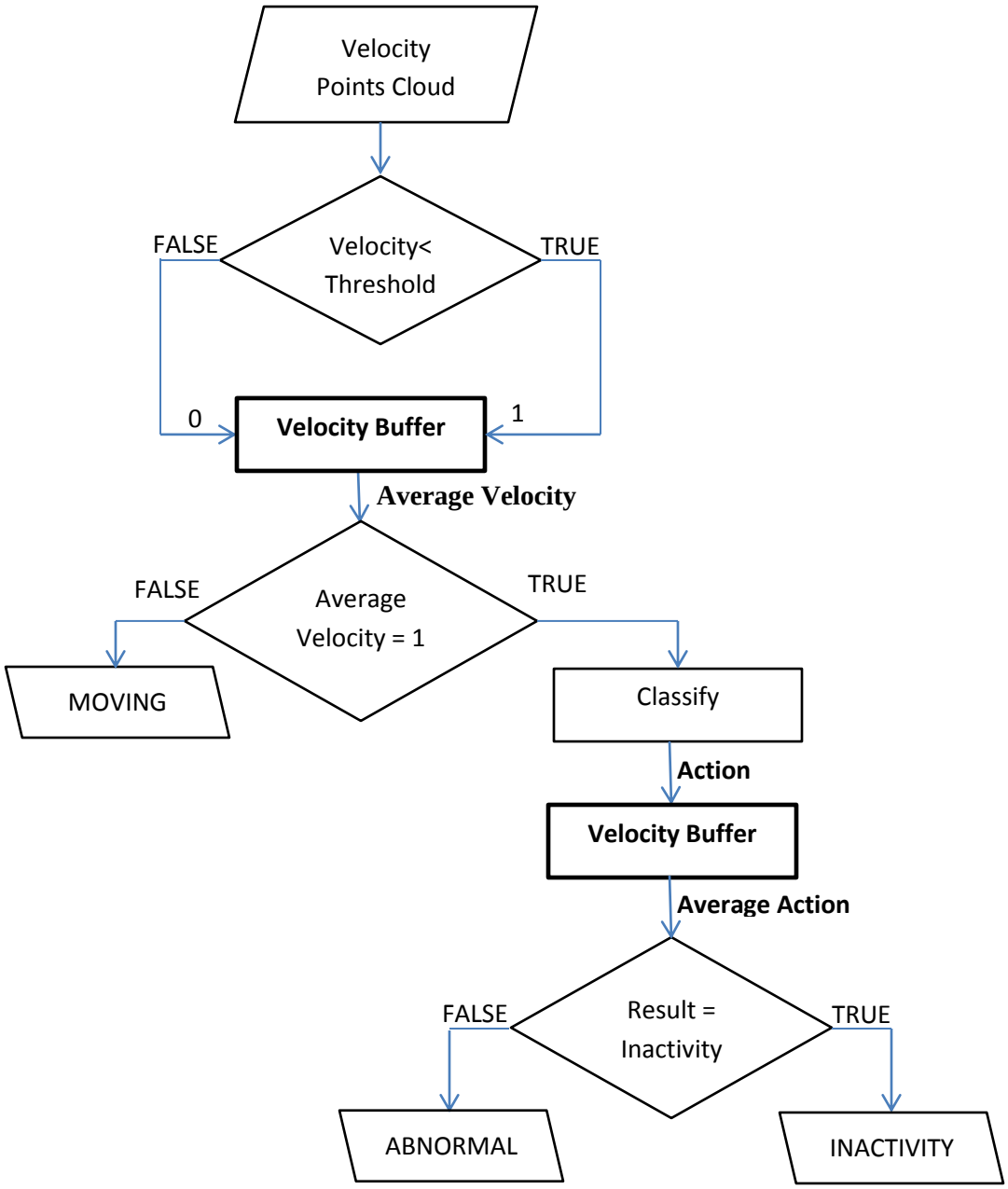

Figure 17: Classification Flow Chart with Buffers

4.3.2 Low level Filtering

All the actions that last for less than 60 frames are removed. The concept of that filtering is very simple, but the implementation was tricky because we aimed to do the classification part of the project in real time. So filtering should be done in real time as well. The method used is similar to the buffers, which are explained in the previous section, but instead of keeping track of the previous movement, the system keeps tracks of the following movements. So, the actual classification is happening in real time, but a few frames later of what the users see. In addition, a list is used instead of an unsigned int because there are three different possible actions (moving, inactivity, abnormal) that need to be stored.

Function getAndAddAction(Action):

```
outputAction ← get first element of ActionsSequence
remove first element from ActionsSequence

if (CurrentAction = Action) then
        Counter++
        push Action to the back side of ActionsSequence
else
        if(Counter>=60) then
                PreviousAction ←  CurrentAction
                CurrentAction ← Action
                push Action to the back side of ActionsSequence
                Counter ← 0
        else
                Counter ← SEQUENCE_SIZE
                replace CurrentAction with PreviousAction inActionsSequence
                CurrentAction ← PreviousAction
                Push CurrentAction to the back side of ActionsSequence
```

In the above peudocode CurrentAction is the action that has just been classified. This mean that is actually the action the user will see after SEQUENCE_SIZE frames.

4.3.3 Filtering Results

To observe the effect of filtering the following images has been generated. Those images show how the classification results of video 5 have been modified during filtering. The horizontal axis represents time and the vertical axis represents actions: 1 for moving, 2 for inactivity and 3 for abnormal. The purple line shows the correct results. The first image shows the results before applying filtering and the final image shows the results when all the filters are appied. It is obvious that filtering has gradually improved the quality of the classification.

(a) No Filtering At All

(b) Filtering with only the Actions Buffer

(c) Filtering with both Buffers

(d) Filtering with both Buffers and Filtering of all the short actions

Figure 18: Filtering Results

4.4. Visualisation – FBO and OpenCV

The visualisation of the mocap information and the video is done in a single GL Window. By using Frame Buffer Objects the motion capture is rendered to a texture. In addition, using the OpenCV library an IplImage is generated each frame of the input video. In the GL Window, there are two planes. On each plane a texture, either the rendered textured or the IplImage, is attached. Those two planes can be swapped by either pressing S from the Keyboard or by clicking on the viewport of the GL Window.

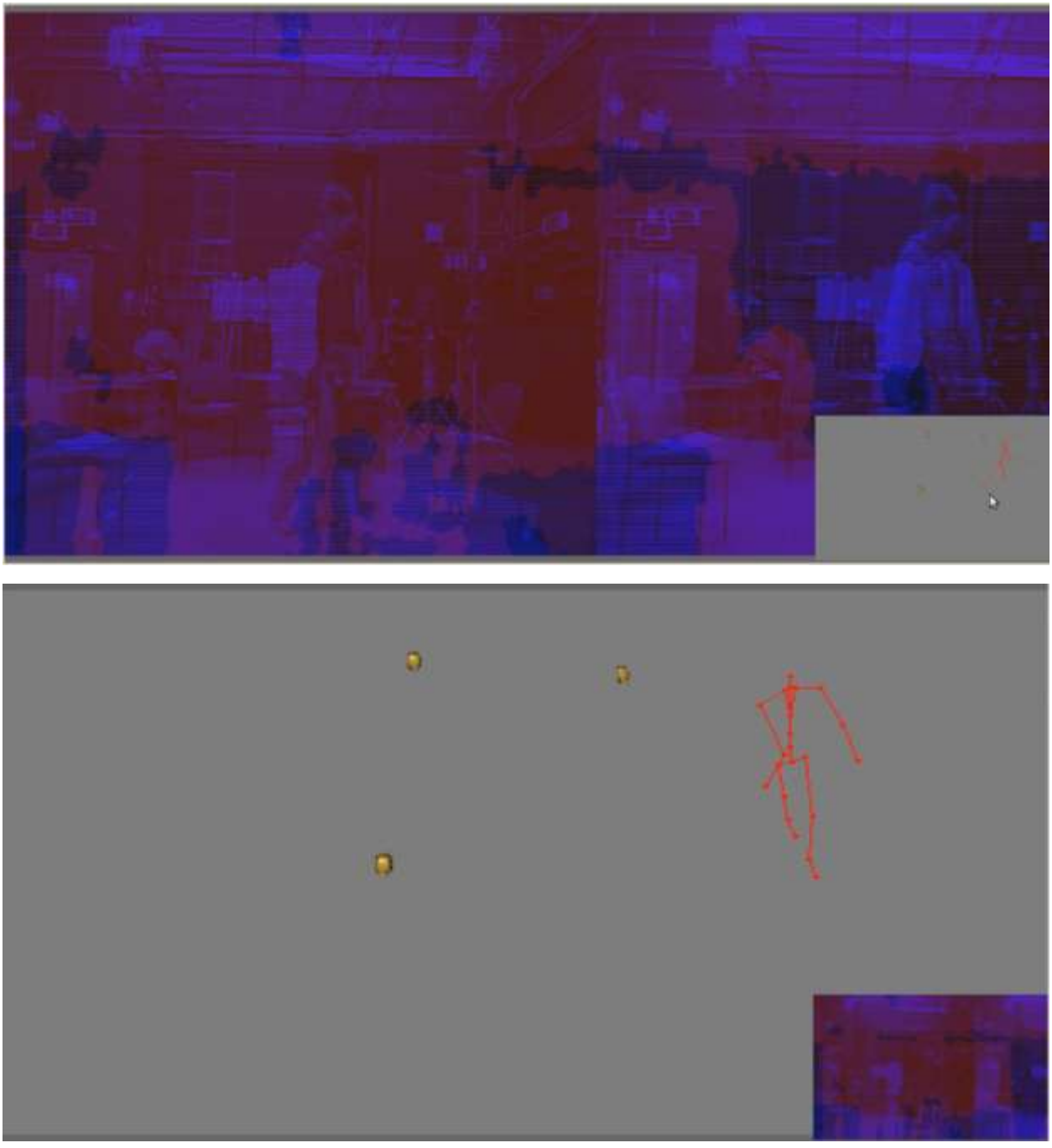

Figure 19: Visualisation of Mocap and Video

4.5. Clusters' Manager

Clusters' Manager is a sub-program of our system. In that application's GUI, there are two tabs: 1D and 3D. The 3D tab is used to calculate the mean vectors and covariance matrices of a given number of clusters. It takes as input the x,y and z coordinates of each cluster and exports a file with the mean vectors and covariance matrices. The 1D tab is an extra feature explained in Section 4.5.3. The User Guide on Section 2.2 gives more details on how to use that sub-application.

The Inactivity Areas file, which was generated using the 3D tab and used for testing, is the following:

```
3                                   → the number of clusters

-12.4321 17.1921 9.49851            → the mean value of the 1st cluster

3.25843 2.02061 -2.63753
2.02061 9.2005 -6.36907             } the covariance Matrix of the 1st cluster
-2.63753 -6.36907 11.0801

3.975 2.82986 9.98906               → the mean value of the 2nd cluster

4.57397 -1.13012 1.76465
-1.13012 6.41501 1.39357            }
1.76465 1.39357 12.2342

-11.6459 3.43403 11.843             → the mean value of the 3rd cluster

2.97414 -2.46606 -0.71863
-2.46606 5.01679 0.240022           } the covariance Matrix of the 3rd cluster
-0.71863 0.240022 21.1933
```

That sub-application could have been embedded in the main application, but if it was a part of it then the whole training data sets should have been loaded each time the application was started. In addition, this way the system has no limitations on the number of inactivity areas that may exist.

4.5.1 Mean Vector

Mean is the sum of all the given points divided by the number of points. It is sometimes referred as the average point of a cluster.

$$M = \frac{1}{\mathrm{N}} \sum_{i=1}^{N} (\mathrm{Pi})$$

(Engineering Statistics, 2003)

4.5.2. Covariance Matrix

Covariance is used in statistical analysis to measure the spread of a cluster and to define the possible relation between two dimensions of a data set. The formula of calculating the Covariance is the following:

$$COVxy = \frac{1}{\mathrm{N}}\sum_{i=1}^{N}(Xi - Mx)(Yi - My)$$

In the Covariance matrix all the possible covariance values are saved. Therefore the Covariance Matrix defines the spread of a data set between all its dimensions. A 3D Covariance Matrix is Calculated as follow:

$$C = \begin{pmatrix} COVxx & COVxy & COVxz \\ COVyx & COVyy & COVyz \\ COVzx & COVzy & COVzz \end{pmatrix}$$

(Rosenbloom)

4.5.3. Histograms

The 1D tab of the sub-program is an extra feature added to our system. It is used to generate histograms that illustrate the frequency of each value. It also calculates the mean value (μ) and the standard deviation (σ) of the data set. Standard Deviation or Variance defines the spread of the data.

$$\mu = \frac{1}{\mathrm{N}}\sum_{i=1}^{N}(\mathrm{Xi}) \ , \qquad \boldsymbol{\sigma} = \frac{1}{\mathrm{N}}\sum_{i=1}^{N}(Xi - \boldsymbol{\mu})^{\mathbf{2}}$$

That was useful in observing the training data and better understanding the amount of noise that exists. Example of output histograms are shown on Figure 21 and Figure 22.

5. Conclusions

5.1. Summary

Many elderly people are hospitalised because they cannot autonomously live in their own homes due to the high risk of falls. Many researches attempted to solve the Fall Detection problem. I n that project, it is claimed there are many advantages of using Kinect sensors for Fall Detection. A scene was set and an actor was asked to performed actions, like sit on a chair and fall down. Using the iPi Studio, the mocap data derived from the videos captured by the Kinect sensors.

The program parses those mocap (.bvh) files and saves the joints hierarchy in a tree structure. Then using the local transformations of the joints defined in the mocap files, the system computes the global transformations of the joints and consequently the actual position of the joints.

The classification is based on the method proposed by McKenna S.J and Nait-Charif H in 2005. When the speed of the actor significantly decreased it implies that he is inactive. If the person is inactive there are two probabilities: he is either in an inactivity area or he is doing something upnormal. Two classification methods have been implemented:

1. Simple Classification using Euclidean Distance
2. Bayesian Inference with Gaussian Functions

They both use statistical analysis of training data sets to recognise whether the actor is in an inactivity area or not.

Since the input data is very noisy, filtering is applied to improve the quality of the classification results.

1. Convolution is used to smooth the change of the velocity.
2. A method of using bitwise operations and buffer was proposed. It is used to keep track of the last movements during 32 frames and returns the average of two actions.
3. All the short movements are removed.

5.1. Evaluation

The following images have been generated to show the performance of the system. Two images have been generated for each video that has been recorded: one using the Bayesian inference and using the simple classification method. The horizontal axis of the images indicates times and the vertical axis indicates the action of the actor. The program is able to recognise three possible actions and they are number as follow:

1. He is moving
2. He is in an inactivity area
3. He is not moving and he is not in an inactivity area.

The evaluation is done by comparing the output of the systems with a manual classification. The purple lines show the expected results of the systems, which has been defined by eye. Appendix B is a list of the evaluation images of all the videos. The following figure shows the classification results of the 1st video.

Video 1: Simple Classification

Video 1: Bayesian inference with Gaussian Functions

Figure 20: Images with Classification Results

The Bayesian inference with Gaussian Functions was an attempt to build a higher level classification method. But during evaluation it was proved to be unstable.

5.2. Limitations

One of the main limitations of the system is the fact that only the classification part is done in real time. Due to the limited amount of time, a third party software, the iPi Studio, was used for the motion capture. As a result, the application is not ready to be used in a real environment with a real patient.

Furthermore, the Gaussian Classification is quite vulnerable. The main issue is the spreads of clusters, used to train the system, because some of those are big. There are cases where the spread seems to be good like the spread of the x- coordinate of the inactivity area Chair A. The following histogram has been generated using the extra feature of the sub-Program and the horizontal axis represents the length of the scene in that dimension.

Figure 21: Histogram of the x-coordinate of inactivity area Chair A

It is noticeable that the spread of the data values approximately forms a Gaussian Curve. But this does not apply to every case. For example the z coordinate of the inactivity are Z is very big:

Figure 22: Histogram of the z-coordinate of inactivity area Telephone

Consequently the classification results, when the actor is answering the telephone are not very good. In addition, the threshold value, which indicates inactivity or not, was significantly decreased, due to that big spread of that training data set. Therefore, the probability of misclassifying non-inactivity to inactivity was increased.

Another limitation of the system is that it was trained only in one scene and only with one actor. As a result, the threshold values set only apply for that specific scene and actor and they would have to be modified if the scene or the actor changes.

5.3. Future Work

Both mocap and classification should be done real-time if the system would be set in a real environment with a real patient. As mentioned before the classification part is done in real time with only a couple of seconds offset for filtering. Therefore, it is possible and it would have been beneficial to merge our system with a real time motion capture tools.

As mentioned in the previous section when the spreads of training data sets of the inactivity areas are big, the robustness of the system is decreased. To increase the quality of the classification result in the simple classification method, a second type of inactivity area may be added. For the simple classification, the average distance between the mean value and the positions of the joints is calculated. The second type of the inactivity area would be defined by a line segment instead of a point, the mean value, and the system would calculate the average distance between that line segment and the positions of the joints. In our scene there are three inactivity areas: chair A, chair B and near the telephone. If two types of inactivity areas existed then chair A and chair B would have been of the first type with the mean value and the place near the telephone would have been defined by a line.

It would have also been beneficial if the systems allowed the user to modify all the thresholds. If that was allowed then the system could have been used in multiple environments. The thresholds could have been modified from the GUI and they could have been exported in a file as scene information. The system would have also allowed the user to load a pre-saved scene from the GUI.

6. References:

1. Bayesian inference – Wikipedia. Available from: en.wikipedia.org/wiki/Bayesian_inference [Accessed 16th of June 2012]
2. Biovision BHV, Graphics Courses, Available from: http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html [Accessed 16th June 2012]
3. codasign Lerning, 2011, Dinstance in 3D Space, Available from: learning.codasign.com/index.php?title=Distance_in_3D_Space
4. Chamberlain C., 2012, Innovations Projects, Bournemouth University
5. Dai J., Bai X., Yang Z., Shen Z., Xuan D., 2010, Mobile phone-based pervasive fall detection, Springer-Verlag London.
6. Engineering Statistics Handbook, NIST SEMATECH, 2003, last update 2012. Available from: www.itl.nist.gov/div898/handbook/ [Accessed 21th July, 2012]

7. Gaussian function – Wikipedia. Available from: en.wikipedia.org/wiki/Gaussian_function [Accessed 17th June 2012]
8. Laurenson D., Jackson M., 2001, Digital Signal Processing Tools, The University of Edingurgh, Department of Electronics and Electrical Engineering. Available from: www.see.ed.ac.uk/~mjj/dspDemos [Accessed 15th July, 2012]
9. McKenna S.J., Nait-Charif H., 2005, Summarising Contextual Activity and Detecting Unusual Inactivity in a Supportive Home Environment.
10. Moore A., 2006, Statistical Data Mining Tutorials, Gaussians. Available from: www.autonlab.org/tutorials/gaussian.html
11. Parent R., 2008, Computer Animation Algorithms and Techniques. Second Edition. US: Morgan Kaufmann
12. Perolle, G., 2006, Automatic Fall Detection and Activity Monitoring for Elderly. Cooperative Research project – CRAFT
13. Rosenbloom, A., Face Recognition, Chapter 4. Mathematical Background. Available from: individual.utoronto.ca/rav/FR/cov.html [Accessed 22th July, 2012]
14. Rougier C., Meunier J., Demo: Fall Detection Using 3D Head Trajectory Extracted From a Single Video Sequence. Universite de Montreal.
15. Taleb T., Bottazzi D., Guizani M, Nait-Charif H., 2009, ANGELAH: A Framework for Assisting Elders At Home
16. Vose Software, Risk Specialists. 2007 Bayesian Inference. Available from: http://www.vosesoftware.com/ModelRiskHelp/index.htm#Analysing_and_using_data/Bayesian/Bayesian_inference.htm [Accessed 25th July, 2012]
17. Wang F., 2011, Motion analysis for in-home gait and balance assessment using inexpensive video sensors. University of Missouri.
18. Walter L. 2002, bioviewer – Linux/Windows Biovision BVH Viewer, Available from: bioviewer.sourceforge.net [Accessed 28th of June 2012]
19. Youtube, 2011, Tutorial – Mocap using Kinect and iPi Desktop Motion Capture, Available from: http://www.youtube.com/watch?v=eyyUXZzKYr4 [Accessed 8th June 2012]

Apendices:

Appendix A: Classification Results

In this appendix the classification results of all the videos have been listed. In Section 5.1, the results are explained and discussed. The length and width of the graphs depends on the length of the video.

Video 1: Simple Classification

Video 1: Bayesian inference with Gaussian Functions

Video 2: Simple Classification

Video 2: Bayesian inference with Gaussian Functions

Video 3: Simple Classification

Video 3: Bayesian inference with Gaussian Functions

Video 4: Simple Classification

Video 4: Bayesian inference with Gaussian Functions

Video 5: Simple Classification

Video 5: Bayesian inference with Gaussian Functions

Video 6: Simple Classification

Video 6: Bayesian inference with Gaussian Functions

Video 7: Simple Classification

Video 7: Bayesian inference with Gaussian Functions

Video 8: Simple Classification

Video 8: Bayesian inference with Gaussian Functions

Video 9: Simple Classification

Video 9: Bayesian inference with Gaussian Functions

Video 10: Simple Classification

Video 10: Bayesian inference with Gaussian Functions

Video 11: Simple Classification

Video 11: Gaussian Classification

Video 12: Simple Classification

Video 12: Gaussian Classification

Video 13: Simple Classification

Video 13: Gaussian Classification

Video 14: Simple Classification

Video 14: Gaussian Classification

Video 15: Simple Classification

Video 15: Gaussian Classification

Video 15: Simple Classification

Video 15: Gaussian Classification

Appendix B: Class Diagrams

Sub-Program:

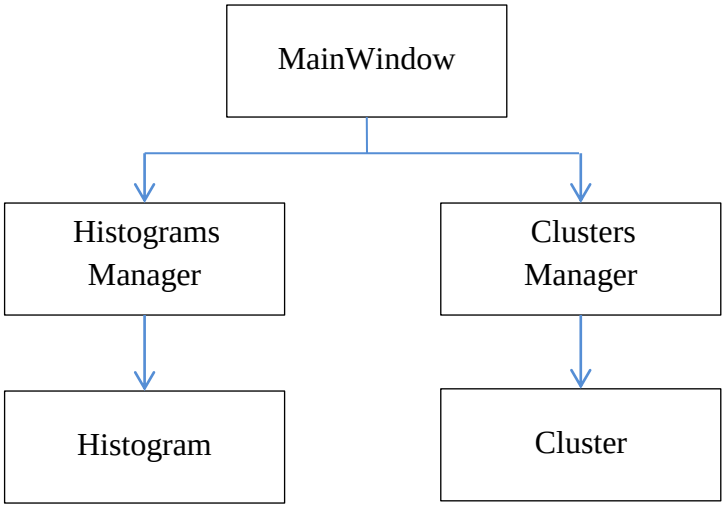

Figure 23: Class Diagram of Sub-Program

Class Cluster:

Cluster
- m_filenames: std::vector<std::string> - m_Xs: std::vector<float> - m_Ys: std::vector<float> - m_Zs: std::vectot<float> - m_mean: ngl::Vec3 - m_covariance: ngl::Mat3x3 - m_isMeanCalculated: bool - m_isCovCalculated: bool
+ CLuster() + getMean(): const ngl::Vec3 & + getCovarianceMatrix(): const ngl::Mat3x3 + readFiles() + calculateMean() + calculateCovarianceMatrix() + setFileName(_name: const std::string &, _index: unsigned int) + writeToFile(_filestream: std::ofstream &) + getFileName(_coordinate: unsigned int): const std::string - getCovariance(_dimension1: const std::vector<float> &, _dimension2: const std::vector<float>, _mean1:float, _mean2:float) : float + ~Cluster()

Cluster Manager:

ClusterManager
m_clusters: std::vector<Cluster>
+ ClusterManager() + getAndSetInputFileName(_index: const unsigned int, _coordinate: const unsigned int) : const std::string & + getInputFileName(_index: const unsigned int, _coordinate: const unsigned int) : const std::string & + calculate(_index: unsigned int) + getMean(_index: unsigned int) : const ngl::Vec3 & + getCovarianceMatrix(_index:unsigned int): const ngl::Mat3x3 & + getCluster(_cluster:unsigned int) : const Cluster & + exportToFile() ~ ClusterManager()

Histogram Class:

Histogram
m_numOfPoints: std::vector<unsigned int> m_values: std::vector<float> m_standardDeviation: float m_mean: float m_histogram: QImage * m_width: unsigned int m_height: usigned int
+ Histogram() + Histogram(_input: const std::string) + LoadImage(_input:const std::string) + SaveImage(_name:const std::string) + getMean(): float + getStandardDeviation(): float + createHistogram(_max: float, _min: float , _step: const float) - getInterval(_point: const float, _min: float, _step: const float) - drawHistogram() + ~ Histogram()

HistogramManager

HistogramManager
m_histogram: Histogram m_input: std::string m_max: float m_min: float

m_step: float
+ HistogramManager() + getStandardDeviation(): float + getMean(): float + loadFile() + saveImage() + setMax(_max:const double) + setMin(_min:const double) + setStep(_step:const double) + ~HistogramManager()

MainWindow

MainWindow
- m_ui: UI:MainWindow - m_histogram: HistogramManager - m_clusters: ClustersManager - m_currentCluster: unsigned int
+ MainWindow(parent: QWidget *) + closeProgram() + calculate1D() + getFileNameX() + getFileNameY() + getFileNameZ() + calculate3D() + setCurrenCluster(_index:int) + exportToFile() + changeNumOfClusters(int _numOfClusters) + ~MainWindow()

Main Program:

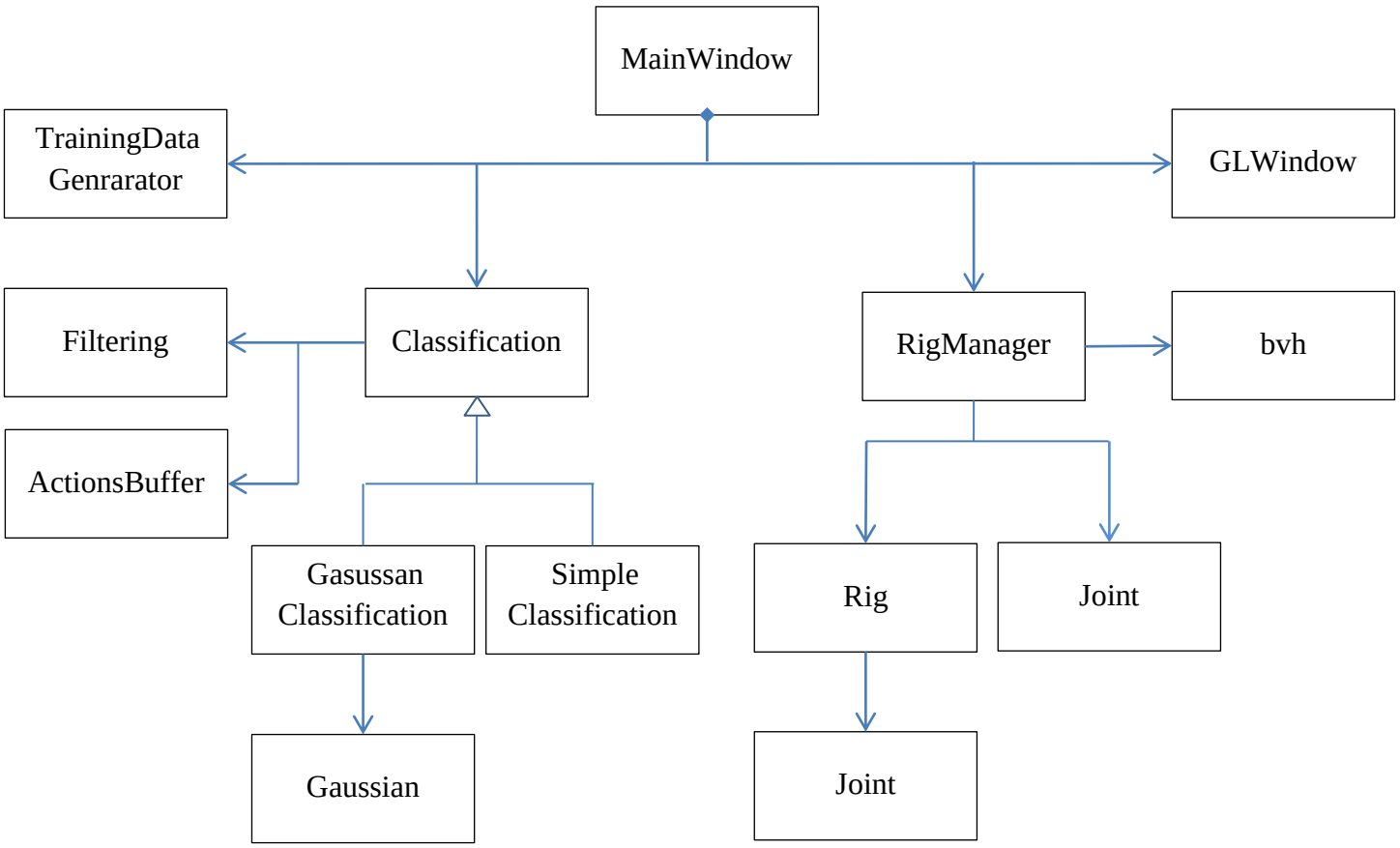

Figure 24: Class Diagram of Main-Program 47

Class Buffer

Buffer
- m_buffer: unsigned int
+ Buffer() + addAction(_f: bool) + getAverageAction(): bool + ~Buffer()

Class bvh:

bvh
- m_root: const unsigned int - m_current: unsigned int - theMode: mode - m_tempRotation: ngl::Vec3 - m_tempPosition: ngl::Vec3 - m_channelIndex: unsigned int - m_partIndex: unsigned int - m_channelsNum: unsigned int - m_vertIndex: unsigned int - m_rig: Rig* - m_framesNum: unsigned int
+ bvh(_rig: Rig *) + bvh() + init(_bvhFile:const std::string) + set(_bvhFile:const std::string) - degreeToRadians(_angle: const double) - process(line:const std::string) - recurs(_some:unsigned int) + ~ bvh()

Classification class:

Classification
m_Actionsbuffer: Buffer # m_velocityBuffer: Buffer # m_filtering: Filtering # m_meanValues: std::vector<ngl::Vec3> # m_message: std::string # m_count: unsigned int
+ Classification() + readFile(_file: const std::string &)=0 + hasHeFallenDown(_positions: std::vector<ngl::Vec3>&) + getMeanValues(): cont std::string<ngl::Vec3>& + classify(_velocity: float, _pointsCloud: std::vector<ngl::Vec3> + classify() - isHeInAnInactivityArea(_positions: std::vector<ngl::Vec3>&)=0

Filtering Class::

Filtering
- SIZE: static const unsigned int - m_actionsSequence: std::list<Action> - m_previousAction: Action

- m_currentAction: Action - m_counter: unsigned int
+ Filtering() + getAndAddAction(_action: Action): Action + getAction(): Action ~ Filtering()

Gaussian Class:

Gaussian
- m_mean: ngl::Vec3 - m_covMatrix: ngl::Mat3x3 - m_inverseCovMatrix: ngl::Mat3x3
+ Gaussian() + setMean(_mean: const ngl::Vec3) + setCovMatrix(_matrix: const ngl::Mat3x3) + GaussianFunction(_point: const ngl::Vec3 &) + ~Gaussian()

GaussianClassification Class:

GaussianClassification
- m_gaussians: Gaussian
+ GaussianClassification() + readFile(_file: const std::string&) - isHeInAnInactivityArea(_positions: const std::string<ngl::Vec3> &) + ~GaussianClassification()

GLWIndow Class:

GLWindow
- m_cam: ngl::Camera * - m_transformStack: ngl::TransformStack - m_light: ngl::Light * - m_textureID: GLuint - m_fboID: GLuint - m_rboID: GLuint - m_width: GLuint - m_height: GLunit - m_vaoSegments: ngl::VertexArrayObject * - m_vaoJoints: ngl::VertexArrayObject * - m_meanValuesOfInactivityAreas: std::vector<ngl::Vec3> - m_material: ngl::Material - m_pointsCloud: std::vector<ngl::Vec3>*

- m_segments: std::vector<ngl::Vec3>* - m_frame: IplImage* - m_capture: CvCapture* - m_data: unsigned char - m_isAviEnabled: bool - m_rgbTextureAvi: GLuint - m_aviTrans: ngl::Transformation - m_nglTrans: ngl::Transformation
+ GLWindow(_parent: QWidget *) + timerEventCalledFromMW () + setPointCloud(_pointsCloud: std::vector<ngl::Vec3>* + setSegments(_segments: std::vector<ngl::Vec3>*) + buildVao() + start(_meanValues: const std::vector<ngl::Vec3>) + setAviFileName(_name: const std::string &) + restartVideo() + swapPlanes() # loadMatrixesToShader(_tx: ngl::TransformStack &) # initializeGL(); # draw() # paintGL() - mousePressEvent(_event: QWheelEvent *) - createTextureObject() - createFramebufferObject() + ~GLWindow()

Joint Class

Joint
m_name: std::string # m_motion: std::vector<ngl::Transformation*> # m_globalTransforamtions: std::vector<ngl::Matrix> # m_positions: std::vector<ngl::Vector> # m_offset: ngl::Vector # m_parent: unsigned int # m_children: std::vector<unsigned int> # m_channels: std::vector<channelTypes>
+ Joint() + Joint(_j:const Joint&) + Joint(_name: const std::string) + Joint(_offset: const ngl::Vector) + addMotion(_motion: const ngl::Transformation) + addOffsetToFirstMotion() + clearAllExpectFromPositionsAndChildren() + getNumOfChildren(): unsigned int + getPositions(): const std::vector<ngl::Vector> & + getChildren(): const std::vector<unsigned int> & + Joint()

MainWindow Class

MainWindow
- m_file: std::string - m_bvh: std::string - m_aviFile: std::string - m_ui: Ui::MainWindow * - m_gl: GLWindow - m_rig: RigManager - m_inactivityAreas: Classification * - m_classificationMethod: unsigned int - m_currentFrame: unsigned int - m_numOfFrames: unsigned int - m_isHeInAnInactivityArea: bool - m_isEnabled: bool - m_offset: unsigned int - m_tOffset: unsigned int - m_isAviLoaded: bool - m_isAviStarted: bool - m_pointsCloud: std::vector<ngl::Vec3> * - m_trainingDataGenerator: TrainingDataGenerator
+ MainWindow(_parent: QWidget*) + setFile() + setBvh() + setAvi() + start() + stop() + setClassificationMethod(_method: int) + setOffset() # keyPressEvent(_event: QKeyEvent *) # resizeEvent(_event: QResizeEvent*) # timerEvent(_event: ATimerEvent*) - openDialogBox() - initialiseClassification() + ~MainWindow()

Rig Class

Rig
m_size: unsigned int # m_joints: std::vector<Joint> # m_numOfFrames: unsigned int # m_segments: std::vector<unsigned int
+ Rig() + getJoints(): const std::vector<Joint> + getParent(_joint: const unsigned int): unsigned int

+ addJoint(_parent: unsigned int): unsigned int + setNameOfJoint(_joint:const unsigned int, _name: const std::string &) + setOffset(_joint: const unsigned int, _type: const Joint::ChannelTypes) + addChannelToJoint(_joint: const unsigned int, _type: const Joint::channelTypes) + getNumOfChannels(_joint: const unsigned int): unsigned int + addFirstMotion() + getChannelType(_joint: const unsigned int, _channelIndex: const unsigned int): Joint::channelTypes + getTheNumberOfChannels(_joint: const unsigned int) + addMotion(_joint: unsigned int, _m: ngl::Transformation) + clear() + calculateJointsPosition() + setNumOfFrames(_n: unsigned int) + getPosition(_joint: const unsigned int, _motionIndex: const unsigned int): const ngl::Vector + getNumOfFrame(): unsigned int + getSegments(): const std::vector<unsigned int> + calculateSegments() + getOffsetOfJoint(_joint: unsigned int): const ngl::Vector & + computeGlobalTransformations() + getPositions(_index:const unsigned int): const std::vector<ngl::Vector> + reset() - computeGlobalTransForFrame(_frame: unsigned int) - clearSomeMemory() + ~Rig()

RigManager Class

RigManager
m_velocity: Velocity # m_rig: Rig # m_pointsCloud: std::vector<ngl::Vec3> # m_size: unsigned int # m_segments: std::vector<ngl::Vec3>
+ RigManager() + set(_name: const std::string) + calculatePositions(_frame: const unsigned int) + getPointsCloud(): const std::vector<ngl::Vec3>* + getSegments(): const std::vector<ngl::Vec3>* + getSize(): unsigned int + getNumOfFrames(): unsigned int + getVelocity(_frame: unsigned int): float + ~RigManager()

SimpleClassification Class:

SimpleClassification
+ SimpleClassification() + readFile(_file: const std::string) - isHeInAnInactivityArea(_positions: const std::vector<ngl::Vec3>&) - getEuclideanDistance(_point1: const ngl::Vec3 & _point2: const ngl::Vec3 &) + ~SimpleClassification()

TrainingDataGenerator Class

TrainingDataGenerator
m_coordinates: std::vector<ngl::Vec3>
+ TrainingDataForInactivityAreas(_rig: const RigManager) + addTrainingData(_positions: std::vector<ngl::Vec3>) + loadDataFromFiles() + writeToFiles() - openOpenFileDialogBox(_dialogFoxTitle: const std::string&): const std::string - openSaveFileDialogBox(_dialogFoxTitle: const std::string&): const std::string + ~TrainingDataForInactivityAreas()

Velocity Class

Velocity
- m_positions: std::vector<ngl::Vec3> - m_velocities: std::vector<float> - m_filter: std::vector<float> - m_sum: float
+ Velocity() + setRootPositions(_positions: const std::vector<ngl::Vector> &) + getVelocity(_frame: unsigned int): float - smoothVelocity() - calculateVelocity() - getNewPosition(_index: unsigned int): ngl::Vec3 - getEuclideanDistance(_point1:ngl::Ve3, ngl::Vec3 _point2) + ~Velocity()