Van Hoang Giang Phan

Behaviour Recognition in an Elevator Using Kinect Sensor

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electronic Engineering

Bachelor's Thesis

16 November 2021

Abstract

Author:	Van Hoang Giang Phan
Title:	Behaviour Recognition in an Elevator Using Kinect Sensor
Number of Pages:	51 pages
Date:	16 November 2021
Degree:	Bachelor of Engineering
Degree Programme:	Electronic Engineering
Professional Major:	Electronics
Supervisors:	Claudio Stuppi, Senior Expert (KONE)
	Janne Mäntykoski, Senior Lecturer (Metropolia UAS)

Increasing population and urbanization has led to widespread use of elevators in buildings. There is always a concern associated with the safe usage of elevators. Manual video monitoring is often slow and ineffective when responding to emergency situations inside the elevator cabin, which can lead to undesirable consequences.

The objective of this thesis project was to develop a behaviour recognition prototype based on the Kinect depth sensor that could automatically recognize different behaviours in an elevator. Recognition results were recorded in the database and would be analysed by professionals to prevent emergency situations and to further improve safety and riding experience in an elevator.

The prototype utilized human silhouettes for recognizing behaviours. Project work comprised of designing gesture database, developing behaviour recognition software, testing and research on camera placement. For gesture database design stage, videos of gestures were recorded and tagged. In the software development stage, the behaviour recognition functionality of the prototype was implemented. During the testing stage, underlying algorithm's performance was evaluated. In the final stage, case studies of camera placement for standing and lying gestures were carried out.

The developed prototype was able to detect defined behaviours with high accuracy. With further advancements, it would provide complete intelligent monitoring capability for elevators and further leverage their safety standards.

Keywords: Behaviour recognition, machine learning, Kinect, gesture, elevator, depth sensor, computer vision

Contents

List of Abbreviations

2D:	2-Dimensional
3D:	3-Dimensional
CSI:	Channel State Information
HAR:	Human Activity Recognition
IDE:	Integrated Development Environment
IoT:	Internet of Things
IR:	Infrared Radiation
LED:	Light Emitting Diode
RFID:	Radio Frequency Identification
RGB:	Red Green Blue
RSSI:	Received Signal Strength Indicator
TOF:	Time of Flight

1 Introduction

Behaviour recognition is an action of recognizing human behaviour based on non-verbal communication signs: facial expressions, hand gestures, arm and leg movements. Nowadays, there already exists a number of automated monitoring solutions across industries, that perform behaviour recognition without human intervention. They observe people, classify their behaviours and provide recommendations to businesses or individuals.

With increase in population and rising urbanization rate in cities, it is extremely hard to manually monitor all people's behaviours to maintain social order. Automated monitoring systems would be able to assist authorities and companies to monitor densely populated areas. Rise of IoT ecosystems and smart cities create opportunities for implementing those systems. It is predicted that by year 2030, there will be 25.4 billion IoT connected devices worldwide [1]. Data from those devices would be used in automated monitoring systems to recognize the activities of people.

It is widely known that office buildings are excessively crowded during peak times and the flow of people is congested. This especially concerns the elevators and escalators, that are the main transportation means of customers and employees. It is nearly impossible to monitor all people in those locations during busy times, therefore a method for automatic behaviour recognition had to be proposed.

The goal of the thesis project was to develop a prototype based on the Kinect depth sensor, that could automatically recognize human behaviours in an elevator. The prototype would be installed in an elevator and use human silhouettes to detect activities. The activities' data would be logged in the local database and used by professionals to deal with emergency situations and improve safety and riding experience in elevators.

The implementation of the prototype was carried out in four stages. First stage comprised of recording the gestures and creating a gesture database. Second stage covered the behaviour recognition software design of the prototype. Third stage was testing of the prototype and evaluating its performance. In the fourth stage, studies about the potential prototype placements in an elevator were made.

2 Theoretical Framework

2.1 Human Body Language

Human body language contains a great amount of information about person's intentions and thoughts. Eyes and mouth express person's attentiveness and mood while movements of hands and legs can determine one's relationship with another person. Hand gestures can also hold different meanings, for instance, clenched fist means anger and holding thumb upright means affirmative action. Open or closed posture conveys whether a person is friendly or hostile. Lastly, a distance between people gives a hint on how they view each other. The better people know each other, the smaller is the distance between them. [2.]

Understanding and inferring useful information from the body language is crucial in intelligent systems that complete tasks without human intervention. In this thesis project, posture, legs and hands movements were used to create a prototype that understands the behaviour of a person in an elevator.

2.2 Human Activity Recognition Methods

Human activity recognition has found vast applications in the modern society: from healthcare and commerce to security. Hospitals employ automated systems, which can monitor patients' behaviours and notify staff in case of any emergencies. Businesses infer useful information from their customers' buying activities to predict future trends for their products. Security companies use intelligent monitoring systems to monitor the behaviour of people and prevent

crimes. Utilizing systems that can understand human activities brings many opportunities to the society and individuals; therefore, extensive studies in human activity recognition area have been made in the recent years.

Human activity recognition or HAR can be defined as capability of a system to recognize activities based on the sensor data. Different cutting-edge HAR methodologies have been proposed by researchers based on sensors and computer vision. [3.]

2.2.1 Sensor Approach

Sensor approach utilizes sensors for data gathering and machine learning techniques for analyzing data and recognizing behaviour. The common framework for sensor-based activity recognition can be described with a diagram shown in Figure 1 [3]:

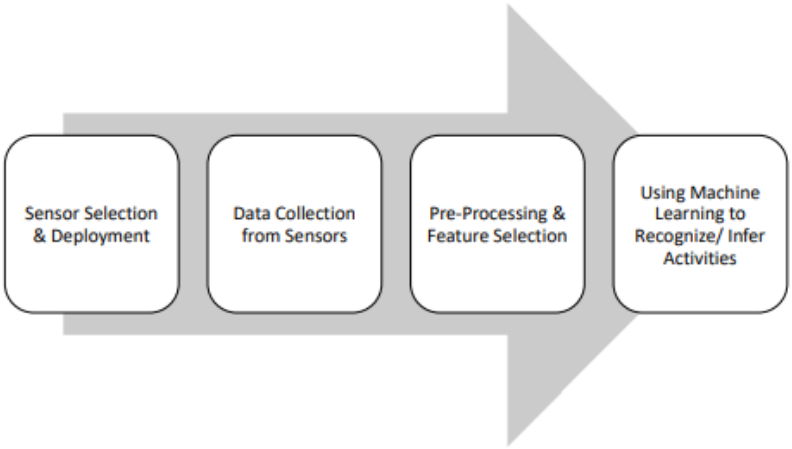

Figure 1. Framework for sensor approach [3]

At the sensor selection and deployment stage, different kinds of sensors are selected and deployed based on the application. There are different criteria for choosing suitable sensors, some of them are accuracy, range of operation, resolution, sensitivity, signal to noise ratio, etc. [4]. They are then deployed on a human body, specific objects or in the environment based on the nature of the HAR project.

At the data collection stage, the raw input data is received from the sensors. Sensor fusion is widely used during this stage for the purpose of increasing the accuracy of the collected data.[3].

During the pre-processing step, the pre-processing techniques, such as data cleaning, transformation and reduction are utilized. The purpose of those techniques is to handle missing, noisy data, and to transform raw data into the useful form for further processing. Then, the feature selection techniques are applied to the data to select relevant features for the machine learning algorithm. [5.]

Lastly, a machine learning algorithm recognizes activities based on the selected features. Different kinds of machine learning algorithms are used for classification of data and inferring the activities. The criteria for choosing the suitable machine learning algorithm are for instance the accuracy, the training time and the number of features used [6].

Various sensors have been used for data gathering in sensor based HAR projects. Due to their low cost, small size and low energy consumption, they are widely used in various industrial, and research projects. Figure 2 presents accelerometer, magnetometer, motion sensor and proximity sensor from left to right.

Figure 2. Sensors used in activity recognition [3]

Accelerometer is a device, which can measure acceleration in different directions. It contains sensors for measuring acceleration in each X, Y and Z directions. It is used in gesture recognition, behaviour recognition, where it measures acceleration of user's body or body parts.

Magnetometer detects changes in magnitude and direction of a magnetic field. It is commonly used in the field of gesture recognition, where magnetic changes caused by movements of hands are used for interaction with machine or computer.

Motion sensor detects motion in the nearby area by measuring infrared radiation. People emit infrared radiation, and therefore their motion can be detected by a motion sensor. It is usually used for detection and tracking purposes.

Proximity sensor detects presence of objects located in its detection range. It sends an electromagnetic radiation towards a target and measures changes in the return signal. It is often used in gesture recognition applications, where the detection of hands movements is required.

Sensor-based solutions can be divided into three categories: wearable, object-tagged and dense sensing. The categorization is based on where the sensors are deployed: human body, specific objects, or environment respectively. [3].

Wearable solutions require a user to wear sensors on the body. Sensors collect data from a user, for example, heart rate, GPS information or acceleration. This data is then used for recognizing human activity using machine learning techniques. Wearable approach is popular in various fields, however not feasible at times. Some users forget to wear sensors or have disabilities, which prevent them from properly wearing the devices. [3].

In object-tagged (or device-bound) solutions, sensors (or tags) are attached to the objects. These objects detect activities based on the user's interactions with them. However, this approach has a disadvantage similar to the wearable

approach, as data gathering is bound to the specific objects, which the user has to use. [3].

Dense sensing (or device-free) approach deploys sensors into the environment, where a user performs daily activities. Data is gathered whenever the user performs the activity, which is then used for further activity recognition. This approach is the most practical among the three sensor approaches due to the fact that the user does not have to wear or interact with any specific devices. The major downside of this approach is that data gathered from the sensors is prone to noise from the surroundings. [3].

It is important to note that hybrid solutions exist, which utilize several of the above-mentioned approaches. The main idea is to capitalize on the advantages of the multiple approaches, which in turn creates a more accurate and less noisy dataset. [3].

Sensors have been used in posture recognition to identify simple postures such as sitting, standing and lying. C.A.Ronao et al. [7] suggested a posture recognition method using a gyroscope and an accelerometer installed in smartphones. The angular velocity and acceleration data collected from those sensors were fed into the deep neural network for posture classification. [3].

2.2.2 Wi-Fi-based Approach

Wi-Fi based recognition techniques have been widely used in HAR activity recognition to recognize human activity both indoors and outdoors. This is due to the widespread usage of Wi-Fi technology and availability of the Wi-Fi access points.

The main idea for the WiFi-based activity recognition is that WiFi signals are diffracted, reflected and scattered during the transmission through a human body. The information held by those signals is extracted to recognize human activity. The commonly used indicators for activity detection are: Received

signal strength indicator (RSSI), Channel state information(CSI) and Doppler shift. RSSI measures the power in the received radio signal. CSI describes properties of the communication link, mainly the amplitude and phase of each subcarrier signal. [8.]

Static transmission model uses the direct and reflected radio signals for activity recognition. The Friis equation can be used to describe a radio transmission:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2} \quad (1)$$

In the formula (1)[8]:

- P_t is the power at the transmitting antenna input terminals
- P_r is the power at the receiving antenna output terminals
- d is the distance between the antennas
- G_t is the transmitting antenna gain
- G_r is the receiving antenna gain
- λ is the wavelength of the transmission

Taking into the account the reflected signal, the Friis equation takes the following form:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 (d+4h)^2} \quad (2)$$

In formula (2)[8] h is the distance between the reflection points and the direct path.

When a human shows up the transmission path changes, resulting in the equation:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 (d+4h+\Delta)^2} \quad (3)$$

In formula (3)[8] Δ is a path difference caused by a human body.

Variable Δ depends on the position and orientation of the human body, therefore P_r is also dependent on the state of the human body. Consequentially, P_r gives the information about the human activity. However, the static transmission model describes activities, assuming that a person stays at the same place. [8.]

Dynamic transmission model also takes a human motion into account and uses the Doppler shift to recognize activities:

$$\Delta f = \frac{2v\cos\theta}{c} f \quad (4)$$

In the formula (4)[8]:

- Δf is Doppler shift value
- v is the velocity of the human motion
- c is the speed of light
- f is the transmission frequency
- θ is the angle between the transmitted signal and the velocity

By calculating Doppler shift of the receiving signal, a pattern of the human motion can be understood. This formula recognizes activities while considering human movements. [8.]

2.2.3 RFID Approach

RFID technology has been very popular in activity recognition field in the recent years due to its high efficiency, low cost, compactness, and low power consumption. RFID system works according to the following principle [3]:

- A reader's antenna sends radio waves to the tags.

- RFID tags modulate the received radio waves using their identification numbers.
- RFID reader antenna picks up the modulated waves, which are scattered back by tags' transponders and extracts the identification information from the tag.

One of the applications of RFID in activity recognition is recognizing the shopping behaviour of clients. Han. et al [9] suggested a system, which utilizes RFID tags, attached on the products to recognize customer's behaviour. Whenever a customer picks up a product, the attached RFID tag sends the signal to the system's reader. The phase shift and Doppler's shift information is extracted and analysed to infer the activity patterns. The proposed method was able to identify the most popular product in a shop and correlations between different products. [3.]

2.2.4 Vision-based Approach

Vision-based HAR approach utilizes computer vision techniques to analyze the video or images and infer human activity information. Vision-based approach is able to recognize more difficult activity patterns, compared to sensor-based approach, but is more complex in nature and requires more processing power. There are many challenges in building a good vision-based activity recognition system, namely [10]:

- cluttered background, meaning the presence of noise and redundant objects in the background
- partial occlusion, where parts of a human body are covered by objects
- lighting, which affects a person's appearance on a video
- viewpoint and scaling, that can distort how a person looks on videos or images
- data labelling, which is time-consuming and is prone to bias
- similarities between gestures, for example standing and walking, can lead to false positives

To mitigate these issues, the following techniques are applied [10]:

- background subtraction, which removes static or non-moving objects from the frame. It allows to focus on the moving objects, which are of interest to the activity recognition system.
- human tracking, in which the human movement is tracked with time. This method allows to observe the specific person and decreases the chance of confusing performers of an activity.

Vrigkas et al. [10] have proposed the categorization of vision based HAR methods based on the nature of applied sensors and the modelling of a human activity. The classification can be seen in the Figure 3 [10].

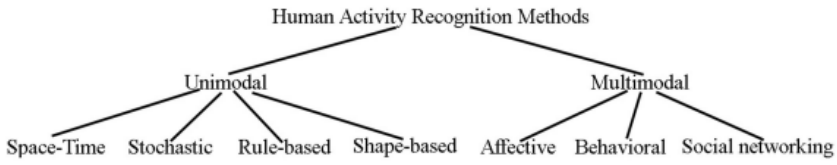

Figure 3. Classification of HAR methods [10]

HAR methods are divided into two categories: unimodal and multimodal. Unimodal methods use one type of sensor data, while multimodal methods use different types. The unimodal and modal methods are further classified into different types based on how they represent a human activity. [10]. This section will give an overview on some of the vision-based methods.

Space Time Methods

Space-time methods represent a human activity as movement trajectories. Wang et.al [11] proposed a HAR method based on dense trajectories. Dense trajectories are applied on an image to capture the local motion features, which are then tracked using optical flow. Based on the features, feature descriptors are calculated and used for detecting the trajectory. [10].

Space time methods have several disadvantages, such as noise and partial occlusion. Due to the fact, that activities are represented as trajectories, which tend to overlap, these methods have problems recognizing activities for several

people. In addition, low recognition rate for complex activities and usage of one camera angle are further challenges for space-time methods. [10].

Stochastic Methods

Stochastic methods view a human activity as a sequence of states, which can be predicted by the means of statistics. Algorithms used in stochastic methods provide high recognition performance but are complex and require many computations. Stochastic methods are not suitable for real-time activity recognition, due to their high time complexity, i.e., the amount of time they require to run an algorithm. [10].

Shape-based Methods

Shape-based methods use human silhouettes to recognize the activities. Silhouettes consist of human limbs connected to each other by joints and are represented by interconnected rectangles in 2D space or cylinders in 3D space. [10]. Kinect sensor's gesture recognition is a famous application of a shape-based method, as Kinect utilizes the user's silhouette to infer the activity information.

Shape-based methods provide highly accurate results for human activity recognition; however, they have few drawbacks. Localization and tracking of different joints are very challenging tasks, due to the consideration of multiple degrees of freedom of each body part. In addition, shape-based methods are sensitive to illuminations, different viewpoint of the camera and clothing's color, as they affect how the human silhouettes appear in the videos. [10.] This thesis project utilized a shape-based method with a Kinect sensor to recognize behaviours in an elevator.

2.3 Discrete and Continuous Gestures

Discrete gestures have only two possible values: true or false, which correspond to them happening or not. There is also a confidence value,

associated with those gestures, which shows the probability of the gestures happening. [12.] Examples of the discrete gestures are sitting, standing and lying.

Continuous gestures describe the progress of gestures and consist of several discrete ones. Their confidence value is between 0 and 1 and shows the current progress of the gesture. [12.] An example of the continuous gesture is the progress of raising up an arm. In that specific case, the state of the arm, when held down can correspond to 0% progress. The states of the arm held horizontally and up correspond to 50% and 100% respectively. By combining these 3 discrete gestures: arm held down, horizontally and up, the continuous gesture of raising up the arm can be formed.

Discrete gestures are used for the gesture detection, while continuous gestures are utilized for the gesture's progress evaluation. These concepts are important when defining and predicting gestures in Kinect applications. [12.]

2.4 Heuristic and Machine Learning Gesture Recognition Approaches

There are two main approaches to gesture recognition in Kinect: heuristic and machine learning.

Heuristic approach is a programmatic approach to describing and comparing gestures. It uses the coordinates of the joint: colour, depth, space and rotation to compare with the coordinates of the other joints. This approach is simple to implement in code and is used to recognize simple gestures. One example of the heuristic approach is to determine if a hand is above a head. To implement this, the space coordinates of the hand joints should be larger than the space coordinates of the head. [12.]

Machine learning approach uses machine learning methods to recognize gestures based on the learnt data. It extracts the relevant characteristics of the gesture and based on those, builds the model to recognize the gesture. This

approach is great for recognizing complex gestures, which are hard to describe programmatically. It also creates an insight to understanding the gesture and can find good features for heuristic approach. [12.]

2.5 Machine learning algorithms used in the prototype

Machine learning algorithms applied in the prototype are of supervised learning type. Supervised learning determines the relationship function between an input and an output, based on a labelled data. The labelled data is annotated with correct answers, desired for the machine learning algorithm by humans. [13.]

Supervised learning algorithms solve regression and classification problems. Regression problem tries to predict the numerical value based on previous observations, while a classification problem estimates the category or class based on the given data. [13.] This section gives an overview of machine learning algorithms implemented in behaviour recognition prototype.

2.5.1 Decision Tree

A decision tree is a supervised machine learning model, which is applied in regression and classification tasks. The general structure of the decision tree can be seen in the Figure 4.

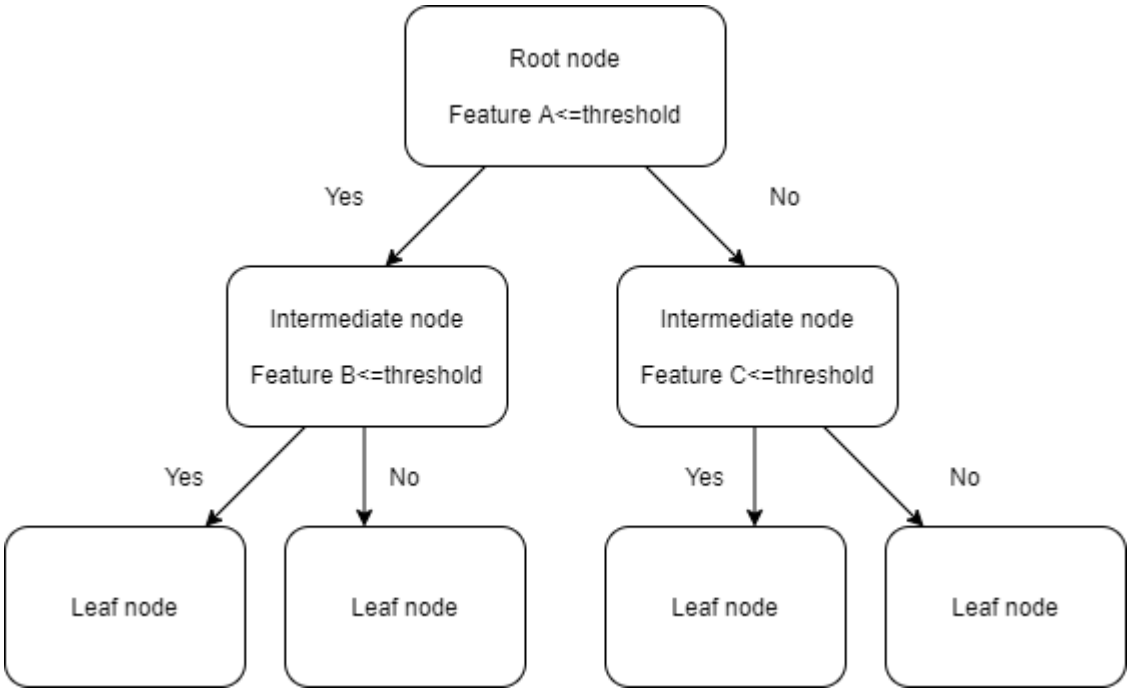

Figure 4. Decision tree's structure

The decision tree consists of root, intermediate and leaf nodes. Root node is the starting point of the decision tree, intermediate nodes evaluate data features against a threshold to split the tree into branches and the leaf nodes contain tree's prediction values. The terms of parent node and child are used in the terminology of the decision tree. Parent node is a node, which is divided into sub nodes, which are called child nodes. For example, root node is a parent node for intermediate node, and intermediate node is its child node. [14.]

To understand the usage of the decision tree in classification problems, an example is given for the Iris dataset. In the Iris dataset, there are 150 iris samples with features: sepal length, sepal width, petal length, petal width. The aim is to classify irises into classes: Iris Setosa, Iris Versicolour, Iris Virginica. The Decision tree for this problem can be seen in Figure 5 [15].

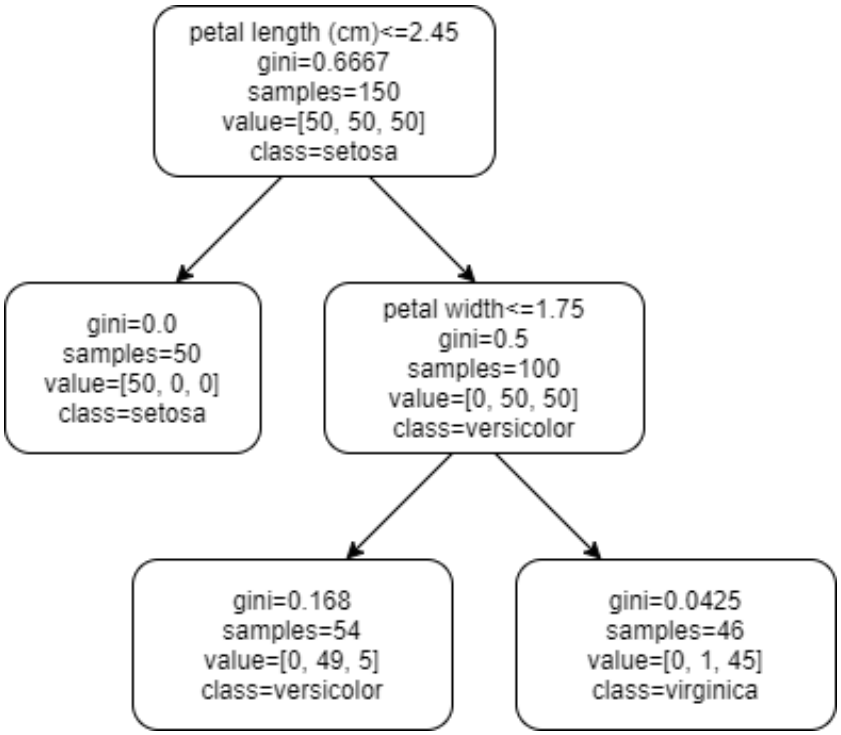

Figure 5. Decision tree for Iris dataset [15]

Building process of the decision tree starts by calculating Gini index for each feature to determine the feature, which best splits the dataset. Gini index describes the probability of wrongly classifying a data point taken randomly from a dataset and can be calculated using formula (5)[16].

$$Gini = 1 - \sum_{i=1}^{k} {p_i}^2 \quad (5)$$

The feature with lowest Gini gain is picked for the data split. [16.] In this case petal length is chosen for splitting at root node. Root node is divided by petal length into two nodes. Left node contains 50 samples of setosa class, and right node has 50 samples each of versicolor and virginica. Node's class is the one that contains the most samples in it, i.e. the mode of the node's dataset. Therefore, for left node, the class is setosa. The class for the left node is determined and requires no further splitting, while the right node requires further splitting.

The same procedure is repeated for the right node: it is divided by petal width with lowest Gini gain into two new nodes. Resulting left node contains most versicolor samples, therefore its class is versicolor, while resulting right node has most virginica samples and its class is virginica. Now the decision tree is able to classify the iris samples into three classes based on their features.

Prediction on a new test sample is made by feeding it into the tree and evaluating its features to determine its path down the tree. For example, a class can be predicted for an iris with features: petal length 3.0, petal width 1.5. This iris's prediction path is shown in Figure 6 [15], denoted by red arrows:

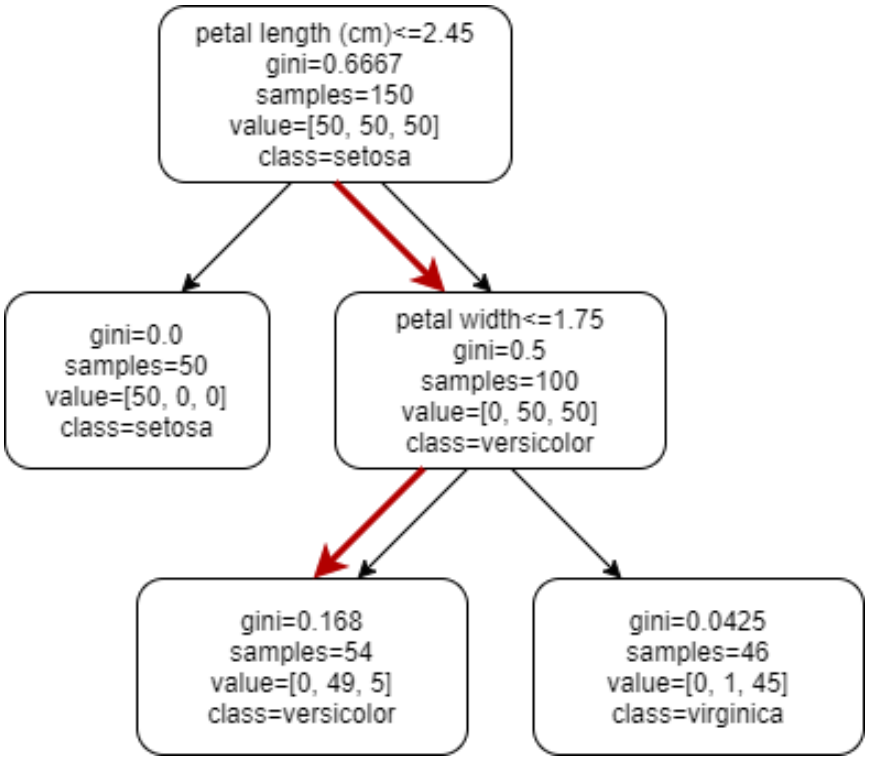

Figure 6. Classification of a new iris sample [15]

From the Figure 6, it can be determined that this iris belongs to class versicolor.

Aside from Gini gain, entropy and information gain are used for finding the best splitting feature. Entropy describes the uncertainty in the dataset and is calculated using the formula (6)[17]:

$$E = -\sum_{i=1}^{N} p_i log_2 p_i \quad (6)$$

where p_i is the probability of randomly choosing the observation of class i and N is the number of classes in the dataset.

Information gain measures the quantity of the useful information provided by the feature. Information gain is calculated by formula (7)[17].

$$Gain = E_{parent} - E_{children} \quad (7)$$

In the formula (7), E_{parent} is the entropy at the parent node and $E_{children}$ is the average entropy at the children's nodes. When these metrics are used to split samples in the node, the feature with highest information gain is chosen for the split. In Kinect's tracking algorithm entropy and information gain are used instead of Gini gain.

Decision tree is also applied in regression problems. A regression example is given for the Boston house price dataset, where the objective is to predict the price of the house (in thousands dollars) based on its features. The decision tree for Boston house price dataset is shown in Figure 7 [14].

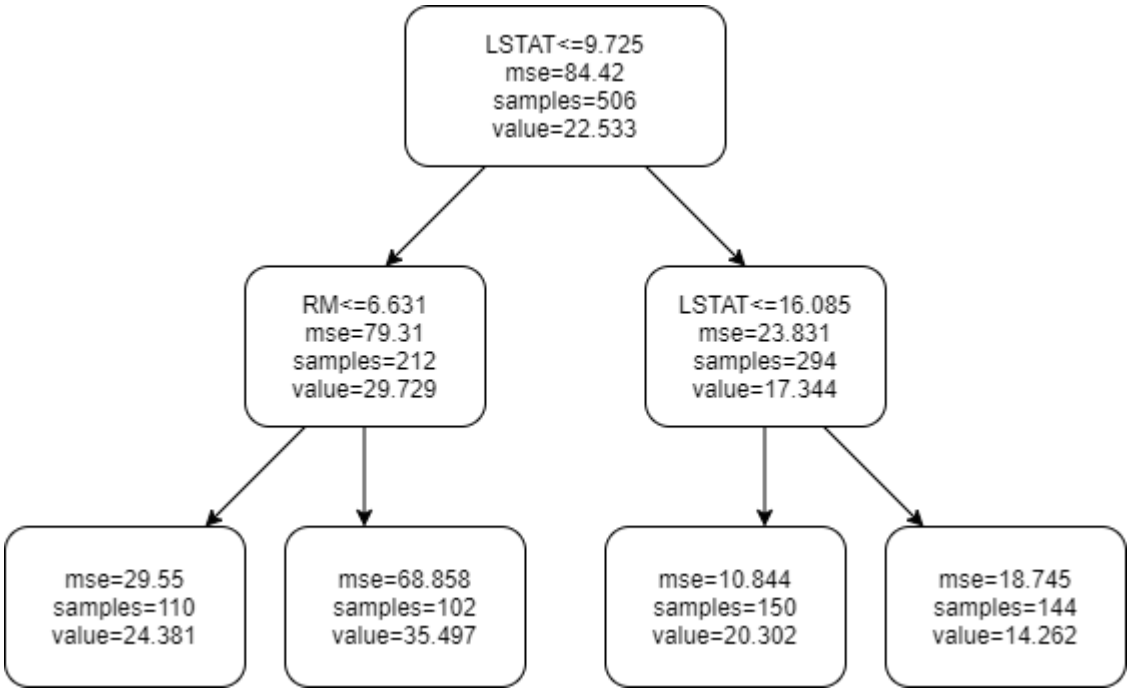

Figure 7. Decision tree for the Boston house price dataset [14]

The tree is built in the same way, as described above. However, the split for each node is determined by the feature with least mean squared error. Mean squared error describes how much the actual value is different from the planned value. It can be computed by a formula (8)[18].

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_{i-}\hat{y}_i)^2 \quad (8)$$

In the formula (8) y_i is the actual value, $\hat{y}_i$ is the predicted value and N is the number of data points.

The value for each node is not a class, but a continuous value - the price of the house. This is understandable as regression problems try to predict a continuous value, based on previous observations.

Prediction on a new test sample is made the same way as in classification case. For instance, the aim is to predict the price of a house with features: %lower status of population (LSTAT)=7.2, average number of rooms per dwelling (RM)

=10. Path for this house sample is shown in Figure 8 [14], marked by red arrows.

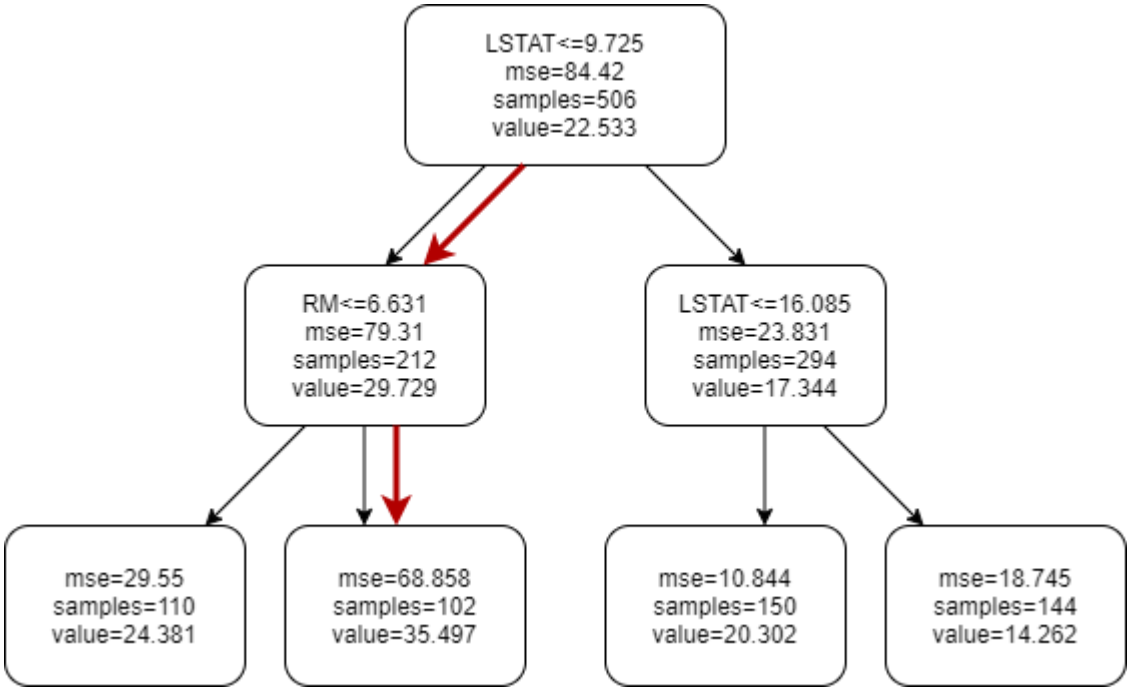

Figure 8. Price prediction for a new house sample [14]

From Figure 8, it is evident that the price prediction for that house would be 35.497 thousand dollars.

It is possible to further divide leaf nodes and grow deeper decision trees to further improve the classification and regression predictions' accuracy. However, growing too deep trees might lead to the problem of overfitting: when the model becomes too dependent on its training data, that it performs poorly on testing samples. Therefore, decision trees usually have specified maximum depth to prevent overfitting, but in turn this diminishes trees' prediction accuracy. Thus, decision tree is a weak learner, that makes low accuracy predictions and has potential of overfitting. [14.]

To overcome the problems of overfitting and low prediction accuracy trees are combined in ensemble models. Two ensemble models: Random forest and Adaptive boosting will be introduced in the next sections.

2.5.2 Random Forest

Bias and variance are the error metrics of machine learning model and are used to assess its prediction accuracy. Bias is the error associated with assumptions of the model. High bias models have simple assumptions about the data, resulting in high training error and not accurate predictions. Variance is the error connected with the flexibility of the model towards unseen data. High variance models fit too well on specific data and make imprecise predictions on unseen data. This also means that high variance models are prone to overfitting. Decision trees have medium bias and high variance; therefore their predictions have low accuracy and not flexible to all kinds of data. [19.]

Random forests combine several decision trees to lower their variance. As a result, their predictions generalise better on different types of data. [19.] An example of the regression random forest is given below.

Assuming that a decision tree used for predicting the house prices for Boston house price dataset was not accurate enough. A random forest will be used for estimating the prices.

A building process for the random forest starts by creating bootstrapped datasets for each tree in a forest. The bootstrapped dataset is a random dataset, created by sampling the initial dataset with replacement. The Figure 9 shows the bootstrapped datasets created for each tree:

Figure 9. Bootstrapped datasets

As can be seen from Figure 9, a variety of bootstrapped datasets are created for each tree by sampling with replacement the initial dataset.

Then each decision tree is built based on its bootstrapped dataset. The building process is illustrated in Figure 10.

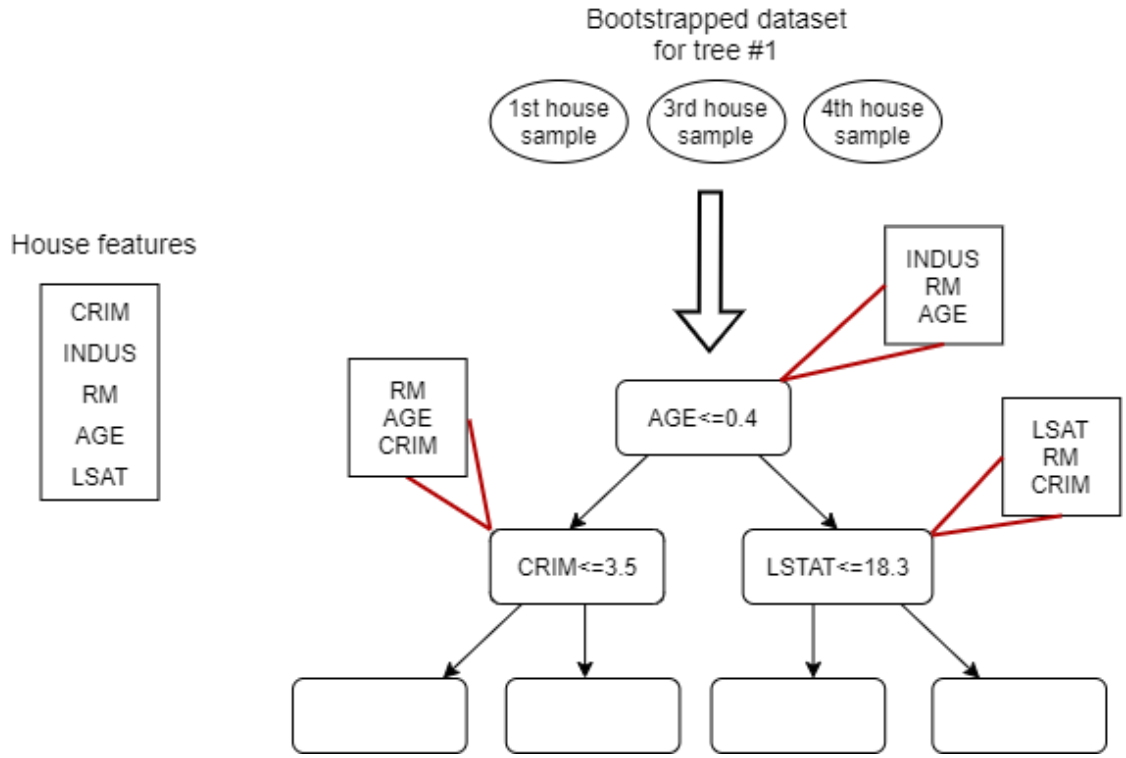

Figure 10. Building a decision tree in a Random Forest

In Figure 10, following house features are present: Per capita crime rate by town (CRIM), Proportion of non-retail business acres per town (INDUS), Average number of rooms per dwelling (RM), Proportion of owner occupied units built prior to 1940 (AGE), %lower status of the population (LSAT).

A bootstrapped dataset is inputted into decision tree and a random subset of features is assigned to each node. The assignment of a feature subset to a node is indicated by red lines in Figure 10. Feature with the lowest mean squared error from a subset will be chosen for splitting at a node. In a root node, the splitting feature is AGE, while in intermediate nodes, those features are CRIM and LSTAT. Same process is repeated hundreds of times to create a large forest with different decision trees.

After the forest is built, it is possible to make predictions on new test samples. The prediction process is shown in Figure 11.

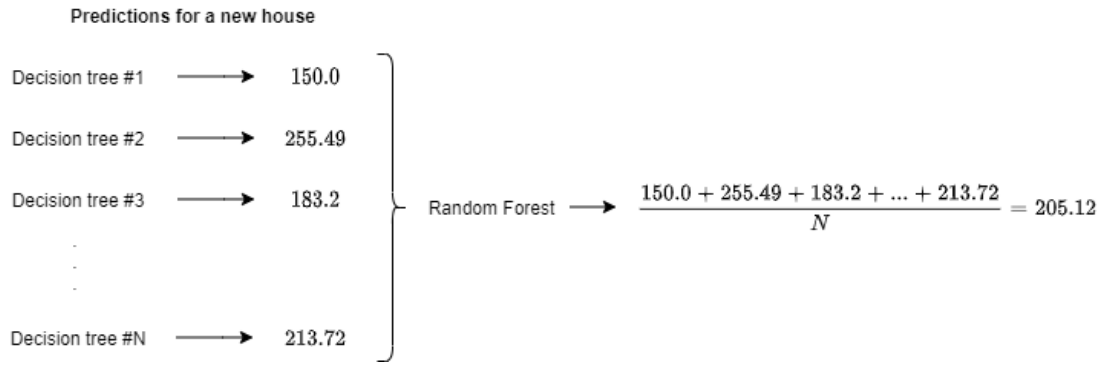

Figure 11. Predicting with a Random forest

Each tree makes a house price prediction based on a new house sample. Prediction made by the forest is a mean of all tree's predictions. Mathematically it can be written as formula (9)[19]:

$$S_L(.) = \frac{1}{L}\sum_{l=1}^{L} w_l\,(.) \tag{9}$$

where S_L is the random forest's output and w_l is the decision tree's output. Finally, the house price predicted by the random forest is 205.12 thousand dollars.

By randomizing datasets and features for each tree, random forest is not closely associated with any training samples. This allows a forest to avoid overfitting and decrease the overall variance. Therefore, predictions made by the random forest will be more accurate towards test data compared to a decision tree. In a case with Boston house price dataset, random forest makes more accurate predictions for a new house's price than a decision tree. [20.]

2.5.3 Adaptive Boosting

Adaptive boosting combines decision trees to lower their bias. This way it makes more accurate predictions compared to an individual decision tree. An example usage of adaptive boosted model in a classification task is given below. [19.]

Presuming that a decision tree used for classifying irises in Iris dataset did not have enough prediction accuracy. Adaptive boosting model can be used to estimate classes of the irises.

The building process of the adaptive boosting model starts by creating tree stumps - decision trees containing only one feature. Tree stump for the Iris dataset is shown in Figure 12 [15].

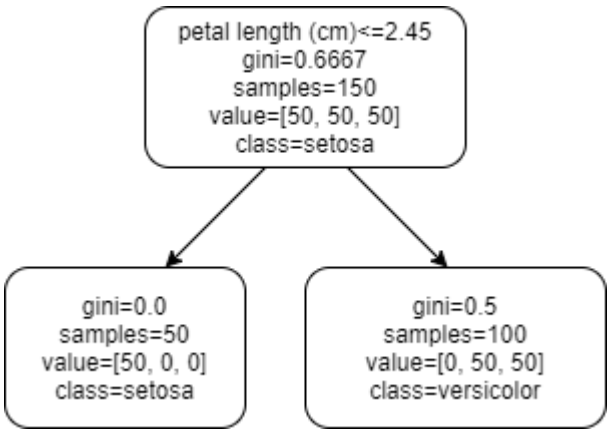

Figure 12. Tree stump for Iris dataset [15]

Then, these stumps are used in building an adaptive boosting model. The building process is illustrated in Figure 13.

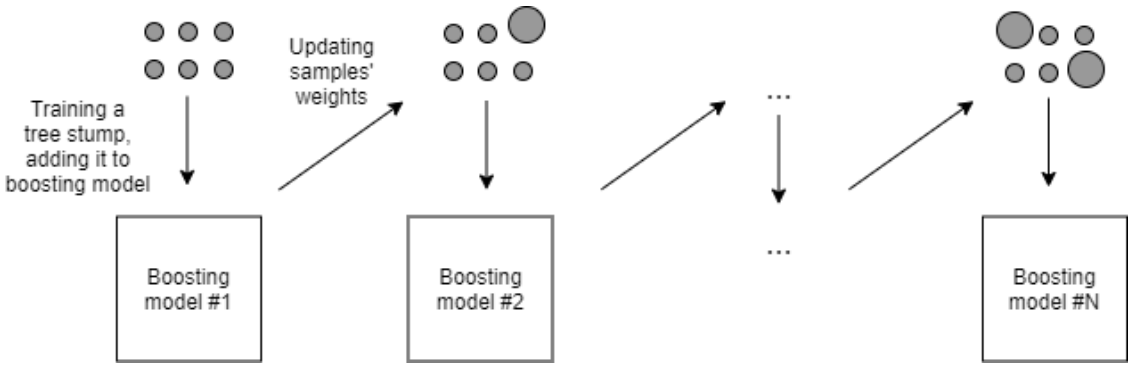

Figure 13. Building process of an Adaptive boosting model

First tree stump is trained on iris samples. It gets a weight according to its training performance and it is added to the boosting model according to the following formula:

$$S_l(.) = S_{l-1}(.) + c_l \times w_l(.) \quad (10)$$

where $S_l(.)$ is a current boosting model, $S_{l-1}(.)$ is a previous boosting model,c_l is stump's weight and $w_l(.)$ is an added tree stump.

Weights of samples misclassified by the current boosting model are increased, so that next tree stump would focus more on them during training. Samples with increased weights are indicated by large grey circles in Figure 13. The same process repeats for the subsequent tree stumps until the specified number of them is reached. The result is a boosting model with large amount of tree stumps.

After the adaptive boosting model is built, it is possible to predict classes for new iris samples. The prediction process is shown in Figure 14.

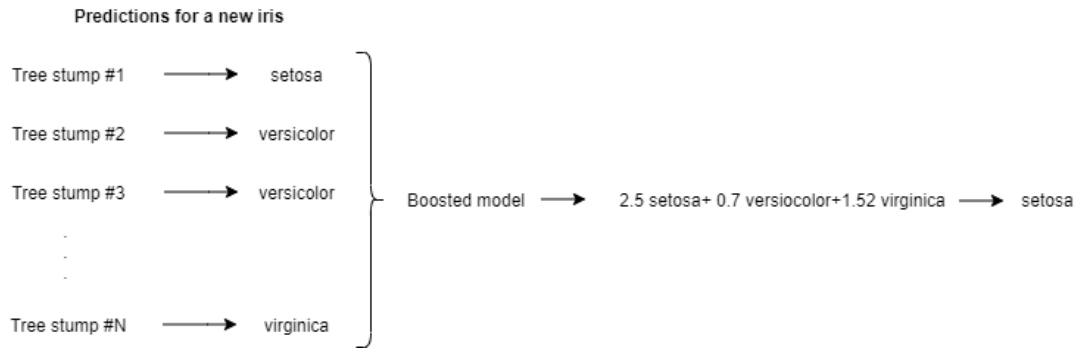

Figure 14. Predicting with Adaptive Boosting model

Each tree stump makes a class prediction based on a new iris sample. Prediction made by the boosting model is a weighted sum of all stumps' predictions. Mathematically it can be written as:

$$S_L(.) = \sum_{l=1}^{L} c_l \times w_l(.) \quad (11)$$

In formula (11)[20], c_l are the weight coefficients of the tree stumps, w_l are the tree stumps in the adaptive boosting and S_L is the boosting model. Adaptive boosting model classifies a new iris as setosa, as it has the highest coefficient of 2.5.

At each model's building step, an added tree stump was concentrating on the samples, misclassified by previous boosting model. As a result, the final boosting model was able to accurately predict all kinds of samples and decrease the overall bias. Thus, an adaptive boosting model can make more accurate predictions on new test samples, compared to an individual decision tree. In the case of the Iris dataset, the adaptive boosting model classifies the new iris sample with higher accuracy compared to the decision tree. [19.]

2.6 Infrared Sensor

An infrared sensor uses infrared radiation for detection of the surrounding objects. According to Planck's radiation law, all objects which have temperature above 0 Kelvin emit radiation in the infrared spectrum of 700 nm to 1mm. Therefore, the infrared sensor can detect the radiation from the surrounding objects based on their temperature and movement. [21].

Active IR sensor include both transmitter and receiver. Transmitter emits the IR radiation, which is reflected from the objects and detected by receiver. Infrared lasers and LEDs usually act as transmitters, while Phototransistors and Photo diodes as receivers. [21].

Typical infrared sensor consists of five parts: infrared transmitter, infrared receiver, transmission medium, optical component and signal processing unit. Transmission medium for IR radiation is usually vacuum, air or optical fiber. Optical component is used to focus the light emitted from the transmitter. The signal processing unit extracts the useful information from the light detected by the receiver. [21]. The IR sensor circuit can be seen in Figure 15 [21].

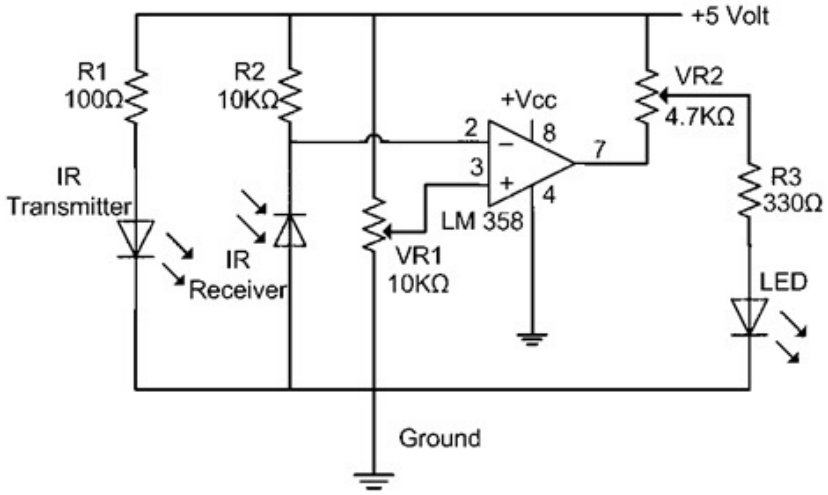

Figure 15. Typical IR sensor circuit [21]

In this circuit, the IR LED emits the infrared radiation, which is reflected by the object and partly received by the photodiode. Whenever, the radiation is detected at the receiver, the inverted input of the amplifier comparator becomes lower than reference voltage at the non-inverted input. The amplifier's output signal will be HIGH and the LED glows up. The variable resistors are used to adjust the reference voltage and the amplifier's output voltage. [21.]

2.7 RGB Camera

The RGB camera records an image by mixing the colours of red, green and blue. It has an array of cavities, which collects the light photons from the surroundings. [22.] The photon collection process can be seen in Figure 16[22].

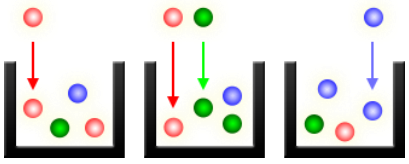

Figure 16. Photon collection in camera's cavities [22]

When a picture is taken by the RGB camera the number of photons in each cavity is quantified and translated into electrical signal. The electrical signals are

approximated into digital data with a specific precision and the grayscale image is created. [22.]

To create the colour image, the Bayer filter and demosaicing algorithm are applied on the cavities. The Bayer filter leaves only one of the RGB colours in each cavity to create the RGB mosaic image. Then, the demosaicing algorithm calculates means of adjacent image pixels to convert the mosaic into the colour image. [23.] The described principle is implemented by Kinect's RGB camera for taking colour images.

2.8 Depth Sensor

A depth sensor is a sensor which measures the distance to the object. There are different types of depth sensors: structured light, stereo depth and time of flight.

The structured light depth sensor projects the laser pattern on the object and measures the distance based on the distortion of the pattern by the object. The stereo depth sensor uses the infrared light from the emitter and the surroundings to measure the distance to the object. It uses the depth information from its two cameras to calculate the distance to the object. [24.]

TOF sensor emits infrared light to the object and measures the time it takes to come back to the sensor. The distance to the object is calculated using formula (12)[25]:

$$d = \frac{ct}{2} \tag{12}$$

where c is the speed of light and t is the time of flight.

Alternatively, the distance can be calculated with the phase difference between the transmitted and the reflected waves using the formula (13)[26]:

$$d = \frac{c\Delta\varphi}{2\times 2\pi f} \quad (13)$$

where c is the speed of light, $\Delta\varphi$ is the phase difference and f is the frequency of the infrared light's power modulation. Kinect's depth sensor utilizes phase difference equation in calculating depth. [26.]

3 Requirements

The objective of this thesis project was to develop a prototype that can recognize behaviours of visitors in an elevator. The behaviours of interest were specified and created by means of video recording and data labelling. After recognizing the behaviours, the information about them is logged into the database. The database data would be analysed by the personnel to deal with emergency situations or to further improve the visitors' experience in the elevator.

The requirements of the prototype were:

- Ability to recognize behaviours
- Custom gestures specification
- Logging of the data for further analysis
- Fast performance
- Optimal placement in an elevator

4 Hardware Components

Mainly Kinect sensor and computer were used for building the prototype. This section contains the functionality descriptions of the devices and their requirements.

4.1 Kinect Sensor

Kinect v2 sensor is used in this study for gesture recording and prediction. The structure of the Kinect sensor is shown in Figure 17 [27].

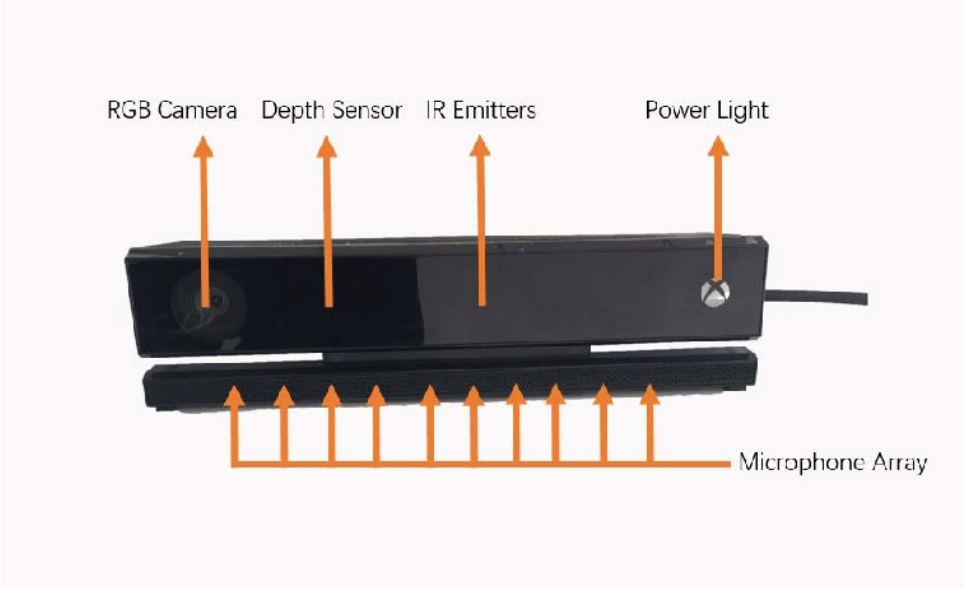

Figure 17. Kinect v2 structure [27]

An RGB camera is used for capturing 2D RGB images, IR emitters for emitting the infrared light and the depth sensor for retrieving the depth images. Part of the infrared light produced by the emitters is reflected from the objects and returns to the depth camera. Distance to the objects is calculated using the phase shift method to create a depth image. A microphone array records the sounds and stores information about their directions. The power light indicates whether the sensor is turned on or off.The specifications of the Kinect sensor are listed in the Table 1 [28]:

Table 1. Kinect v2 specifications [28]

Kinect v2 specifications	
Depth sensor type	Time of flight
RGB camera resolution	1920x1080

RGB camera frame rate	30 fps
RGB camera field of view	84.1° x 53.8°
Depth camera resolution	512x424
Depth camera frame rate	30 fps
Depth camera field of view	70° x 60°
Operating range	0.5 m -4.5 m
Skeleton joints defined	25 joints
Maximum skeletons tracked	6
USB standard	3.0

4.2 Computer

A computer is connected to the Kinect sensor and is used for building the behaviour recognition software. The requirements for the computer are listed in the Table 2 [29].

Table 2. Computer specifications [29]

Computer specifications	
Processor	64 bit (x64)
Memory	4 GB RAM or more
Intel processor	I7 3.1 GHz (or higher)
USB host controller	Built-in USB 3.0 host controller
Graphics card	DX11 capable graphics adapter
Operating system	Windows 8, 8.1,10

5 Software components

A multitude of software programs were used during each step of this thesis project. This section lists the used programs with their descriptions.

5.1 Kinect Studio

Kinect Studio is a software tool that can record a video using the Kinect sensor and playback it [30]. It was used to record the user's gestures in this thesis project.

5.2 Visual Gesture Builder

Visual Gesture Builder lets the user to tag the specific gestures in the recorded clips and create the gesture database [31]. The gesture database was used in real-time gesture detection.

5.3 Microsoft Visual Studio 2017

Microsoft Visual Studio 2017 is an IDE used for writing computer programs. The behaviour recognition software was written in this program.

5.4 NtKinect Library

NtKinect is an open-source library that enables the user to program Kinect using C++ and OpenCV [32]. This library was utilized in writing the behaviour recognition software.

6 Prototype Design

6.1 Behaviour Recognition Process

A block diagram in Figure 18 represents the behaviour recognition process.

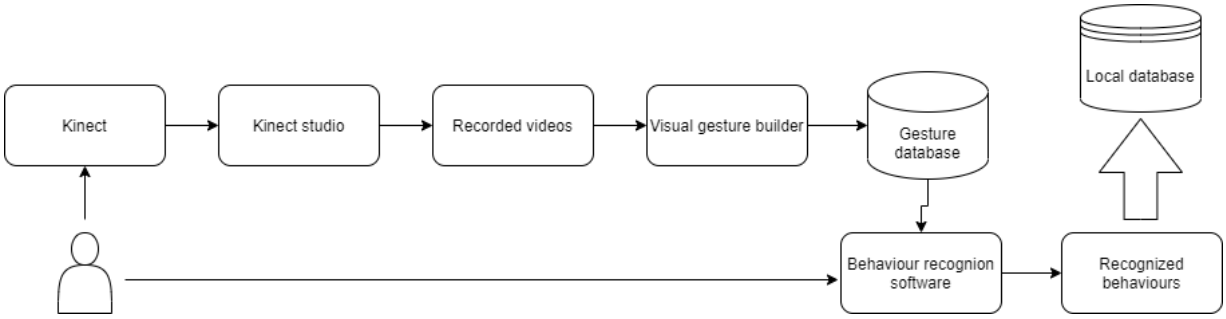

Figure 18. Behaviour recognition process

Kinect was used to record the videos of a person performing the gestures, and with the help of Kinect studio the videos were replayed for errors and saved. The recorded videos were transferred to visual gesture builder to create a gesture database containing the specified gestures. The gesture database was used in behaviour recognition software to recognize the behaviours of the person in the live video. The behaviours with their detection times were saved in a local database for further analysis by specialists.

Based on the above-described workflow, the project work was divided into three parts: gesture database design, building the behaviour recognition software and prototype testing. The gesture database design comprises of recording the videos of gesture performances, tagging the videos and compiling the gesture database. Building the behaviour recognition software includes writing the computer software to detect the behaviours and logging them in the database. Prototype testing is carried out at the end of the project to ensure that the prototype works as intended without errors.

6.2 Kinect Skeletal Tracking

The feature of the Kinect sensor that distinguishes it from other depth sensors is the ability to track joints of a person. The joints are connected to create a tracked skeleton, that repeats the movements of a person standing in front of the Kinect. These skeletons were used for behaviour recognition in the thesis project, and it is important to understand how they were generated. Joint positions are proposed according to the process that can be seen in Figure 19 [33].

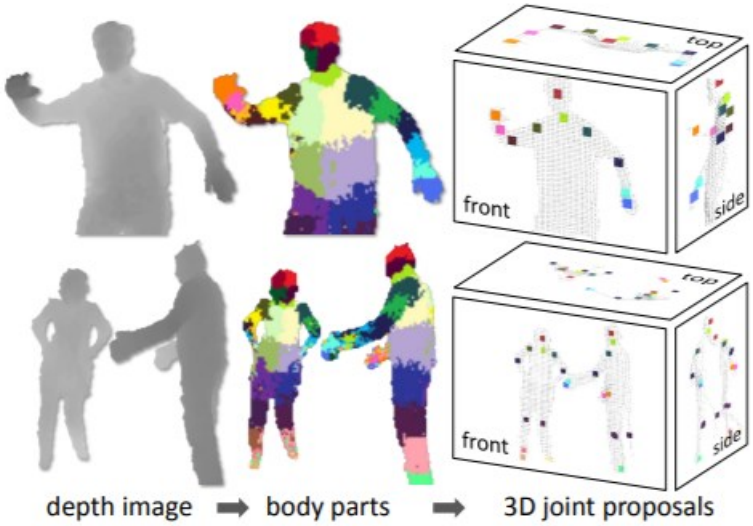

Figure 19. Kinect's joint proposal process [33]

As can be seen from Figure 19, body parts are inferred from a depth image and are used to create 3D joint predictions. The explanation below will describe in detail each step of this process.

Microsoft team created an extensive database of synthetic depth images containing different human postures with variations in human height, body shape, hairstyle, etc. Different postures were created by combining computer graphics models and motion capture data. [33.]

The images were used to train the random forest algorithm, which learned to classify a pixel according to the body part it belongs to. The learning algorithm for a decision tree in a forest was as follows [33]:

1. A set of random images were chosen for training a decision tree and for each image 2000 pixels were picked randomly. This number of pixels was chosen, so that the distributions across body parts would be roughly equal. Pixels served as the input to a decision tree.

2. Set of features θ and thresholds τ were randomly chosen for a root node. The features θ were calculated according to the formula (14) [33]:

$$f_\theta(I,x) = d_I\left(x + \frac{\boldsymbol{u}}{d_I(x)}\right) - d_I\left(x + \frac{\boldsymbol{v}}{d_I(x)}\right) \qquad (14)$$

where $d_I(x)$ is the depth at pixel x in image I and $\boldsymbol{u}, \boldsymbol{v}$ are offsets from the pixel x.

3. Pixels at the root node $Q = \{(I,X)\}$ were divided into right and left branches by each feature and its corresponding threshold. Division is represented by formulas (15) [33].

$$\begin{gathered} Q_l(\varphi) = \{(I,x) | f_\theta(I,x) < \tau\} \\ Q_r(\varphi) = Q \backslash Q_1(\varphi) \end{gathered} \qquad (15)$$

where $Q_l(\varphi)$is set of pixels in the left branch, $Q_r(\varphi)$ are pixels in the right branch and $\varphi = (\theta, \tau)$ represents the set of features θ and thresholds τ.

4. The information gain was calculated for each pair $\varphi = (\theta, \tau)$ to find the best splitting feature:

$$G(\varphi) = H(Q) - \sum_{s\epsilon\{l,r\}} \frac{|Q_s(\varphi)|}{|Q|} H(Q_s(\varphi)) \qquad (16)$$

In formula (16) [33] $H(Q)$ is an entropy calculated over body part labels for all pixels $(I,x)\epsilon\, Q$. The second term is the weighted sum of entropies on left and right branches. This formula says that gain is equal to the difference of entropies on parent and child nodes. Then, the feature θ with largest information gain gave the best split and was chosen as a splitting feature at the node.

5. If the gain had an acceptable value and the tree didn't reach its maximum depth, the steps 2-4 are repeated for the branches.

At the end of the tree's training process, each leaf node contained the probability distribution of the pixel over the body part labels $P(c|I,x)$. Output of the random forest was a mean of the leaves' distributions and it is shown in formula (17)[33]:

$$P(c|I,x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|(I,x) \qquad (17)$$

where T is the number of trees in the forest. [33.]
Random forest uses formula (17) to classify the pixel according to the body part. In Figure 19, each body part is the group of numerous pixels with the same body part label. [33.]

The position proposals for the joints were made based on pixels' probability distributions. The probability density function for a body part was estimated using formula (18)[33]:

$$f_c(\hat{x}) \propto \sum_{i=1}^{N} P(c|I,x_i) * d_I(x_i)^2 * \exp\left(-\left\|\frac{\hat{x}-\hat{x_i}}{b_c}\right\|^2\right) \qquad (18)$$

where N is the number of image pixels, $\hat{x}$ is a coordinate in 3D world space, $\hat{x_i}$ is the reprojection of image pixel x_i in world space and b_c is a Gaussian kernel bandwidth. [33.]

Joint position predictions for a body part are the modes of this density function, shifted by the set offset. The joint prediction with the highest confidence value and above a fixed threshold was displayed on the Kinect skeleton. Final joint predictions are shown in Figure 19. [33.]

6.3 Gesture Database Design

6.3.1 Data Gathering and Cleaning

The first step in designing the gesture database was gathering videos of a person performing the gestures. In total, two behaviours were specified for the

project: ’needing assistance’ and ‘vandalizing’. ‘Needing assistance’ behaviour comprised of the person lying on the floor for some time, while the ‘vandalizing’ behaviour included the person jumping on the elevator floor. For the purpose of detecting those behaviours, two gestures of lying and jumping were recorded on the videos. Due to the inaccessibility to the company’s elevators at the time of the project work, the rectangular small room was chosen as a venue for recording the videos instead of an elevator cabin.

After calibrating the Kinect camera with the room’s floor, the recording of the gestures proceeded, resulting in 51 videos for both gestures. Each video contained a few instances of the person jumping or lying, as well as other unintended gestures [12]. Unintended gestures were included in the clips, so that machine learning algorithm would be able to better differentiate better between wanted and unwanted gestures. The person in the video was overlayed with Kinect tracked skeleton, which was later used in the thesis for gesture detection.

It should be emphasized that Kinect has problems tracking a person in a lying position, as lying position was not included in its synthetic training dataset [34]. To overcome this problem, the recorded person was slightly lifting the hands above the floor, so that the tracking did not disappear.

The videos were replayed in the Kinect Studio with the purpose of detecting errors with skeletal tracking. The videos that contained errors were removed (or ‘cleaned of’) and the videos without errors were grouped for subsequent gesture tagging.

6.3.2 Tagging and Building Gesture Database

The next step was to tag the selected recorded videos for the lying and jumping gestures. This was done using the Visual Gesture Builder software, where the

user can move the cursor through video frames and mark those frames where the gesture is occurring.

The Visual Gesture Builder has a specific hierarchy of files. The files that describe gestures are called projects, and the projects are grouped into solutions, which contain all types of gestures.

In total 5 gesture projects were created: 1 for lying and 4 for jumping. Jumping was viewed as a continuous gesture and was broken into 3 discrete gestures, corresponding to each jumping phase: kneeling down, jumping and kneeling up. The hierarchy can be seen in the Figure 20.

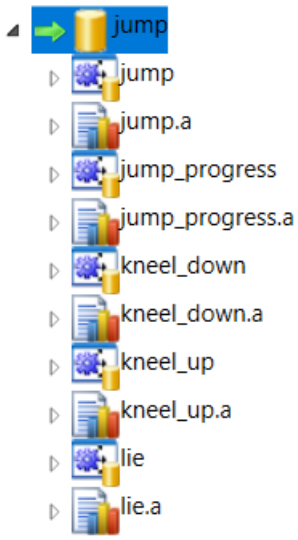

Figure 20. Hierarchy of gesture projects

In the Figure 20, ‘jump_progress’ is a continuous gesture, meaning the progress of the jumping action, while ‘jump’ is a discrete gesture indicating the position of the person in mid-air.
Each project was specified with the parameters for gestures. The specified parameters were:

- gesture name

- discrete or continuous gesture
- body side: left or right or both
- body parts that are not considered for the gesture, i.e., hands, legs, arms

Then, the recorded videos were added to the gesture project for tagging. For each gesture 3 videos were added to the training folder and 2 to the validation folder, in total 25 videos for all gestures. Training folder's videos were used for training the gesture recognition machine learning algorithm, and validation videos were used to test that algorithm. Tags for a discrete lying gesture from one of the videos can be seen in the Figure 21.

Figure 21. Tags for the discrete lying gesture

The person's movements on the infrared image and the skeleton's position in the 3D view are used for reference when tagging. The blue line is applied manually to indicate in which frames the gesture is happening. The areas or frames, where there is no blue line mean that the gesture is not happening there. The same process was repeated for discrete jump, kneel up and kneel down gestures.

The continuous 'jump_progress' gesture was tagged automatically by specifying the limits for each of its discrete gesture. Values from 0 to 0.5 correspond to kneeling up, constant value of 0.5 corresponds to jumping and values 0.5 to 1 match kneeling down. The video tagged for continuous 'jump_progress' can be seen in Figure 22.

Figure 22. Tags for the continuous 'jump_progress' gesture

A wave in Figure 22 logically interprets the jumping process: at the start of the jump a person kneels down, then kneels up and performs the jump, afterwards lands on his feet kneeling down and recovers by kneeling up. Two waves in the Figure 22 indicate that two continuous jumps were performed.

After all the gestures were tagged, the gesture database was built for a solution. Figure 23 shows the output window containing build information for discrete gesture ‘jump’.

```
Step 1 of 4: Loading Labeled Examples
Examples Loaded: Positive(146), Negative(1416)
Done

Feeding Examples...Done. Took (0) seconds
Training Started...

Step 2 of 4: Generating a Pool of Weak Classifiers
       Using Hands data: No
       Using Skeleton data : Yes
        1/38 - Feature: DiffPositionX (90)
        2/38 - Feature: DiffPositionY (60)
        3/38 - Feature: DiffPositionZ (313)
        4/38 - Feature: Angles (283)
        5/38 - Feature: TimeSpaceAngles (181)
        6/38 - Feature: Speed (310)
        7/38 - Feature: VelocityX (174)
        8/38 - Feature: VelocityY (309)
        9/38 - Feature: VelocityZ (263)
       10/38 - Feature: AngleVelocity (369)
       11/38 - Feature: AngleAcceleration (363)
       12/38 - Feature: MuscleForceX (73)
       13/38 - Feature: MuscleForceY (186)
       14/38 - Feature: MuscleForceZ (106)
       15/38 - Feature: MuscleTorqueX (272)
       16/38 - Feature: MuscleTorqueY (44)
       17/38 - Feature: MuscleTorqueZ (214)
       18/38 - Feature: MusclePower (118)
```

Figure 23. Gesture database's build information

From the Figure 23, it can be understood that labeled images were created based on the tagged videos. Those images were either labeled as containing specific gesture or not. In this case, they either were labeled as having ‘jump’ gesture or not. Then, tree stumps would be created based on the labelled images and the skeletal features. Some of the skeletal features shown in the Figure 23 are: position difference, angle, speed, velocity, angular velocity, angular acceleration, muscle force, muscle torque and muscle power. These tree stumps would be combined in adaptive boosting model as shown in Figure 24.

```
Step 3 of 4: Training Strong Classifier
       -Evaluating classifier data for each example skeleton frame...
Done
       -Running AdaBoost using 8 (of 8 available) hardware threads...
Done
       Num weak classifiers: 1000
       Duration: 0 minutes, 0 seconds
```

Figure 24. Building Adaptive boosting model

Overall, four adaptive boosting models were created for discrete gestures ‘jump’, ‘kneeling down’, ‘kneeling up’ and ‘lying’. For a continuous gesture ‘jump progress’, a random forest model was created by grouping decision trees with skeletal features. To summarize, a gesture database contains adaptive boosting or random forest model for each gesture.

6.3.3 Database Testing

The gesture database was tested using the ‘Live Preview’ feature in Visual gesture builder to ensure that the gestures could be correctly detected. ‘Live preview’ uses Kinect’s camera stream for testing. The Figure 25 shows the live preview window for lying gesture.

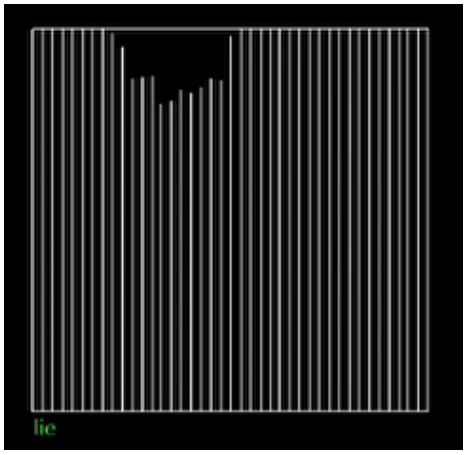

Figure 25. Live preview for lying gesture

In Figure 25 the output window contains a graph, which shows gesture’s detection probabilities over time. In this case, the probabilities are high most of the time meaning that the gesture is correctly recognized using a database. The same

testing process was carried out with other gestures. In case the detection rate for any gesture was low, the videos for that gesture were recorded and tagged again until its performance was acceptable.

6.4 Behaviour Recognition Software

Behaviour recognition software was written in C++ programming language. Its structure can be seen in Figure 26.

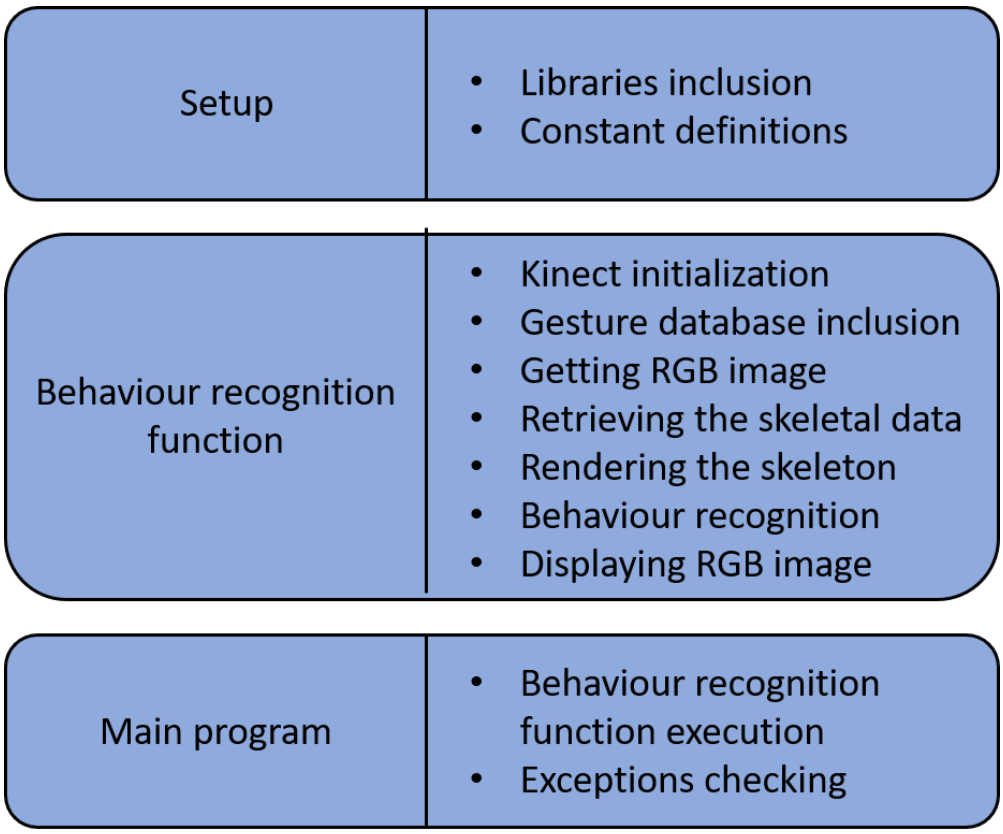

Figure 26. Behaviour recognition software structure

As can be seen from Figure 26 the program was divided into three parts, namely: setup, main function definition and the main program. The first part of the software consisted of setting up the libraries, including NtKinect. In addition, the specific constants were included to enable the gesture recognition functions of NtKinect library.

In the second part, the behaviour recognition function was defined. The function started with Kinect initialization, where object of class Kinect was defined. This Kinect object was used to access the onboard sensors and different functionalities of Kinect sensor. Next, the gesture database, created in Visual gesture builder application, was included. It would be used for behaviour recognition part in the program.

Then, the RGB image was obtained, and the skeletal data was retrieved. The skeletal data contains information about tracked skeletons of multiple people standing in front of Kinect. Therefore, the data was looped through to access the skeleton information of each person. In turn, the skeleton information was a vector, which contained the information about all tracked joints of the person.

Each joint is defined as a C++ data structure and contains information about joint type, position and tracking state. In total, there are 25 joints in a tracked skeleton, and they can be seen in Figure 27 [35].

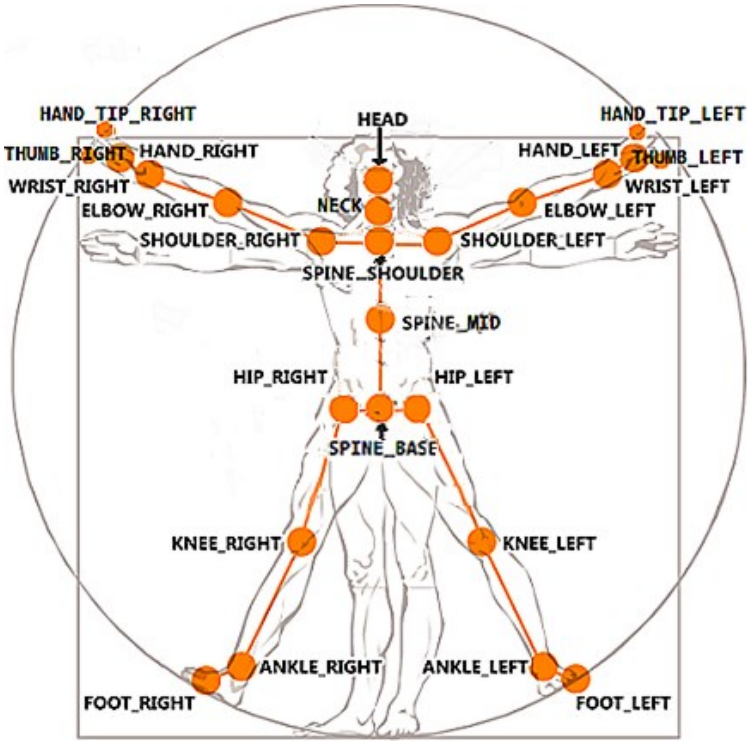

Figure 27. Joints tracked by Kinect sensor [35]

Figure 27 shows the position of Kinect joints in relation to Leonardo da Vinci's Vitruvian Man. The Kinect joints cover the main joints of the human body, which enables Kinect to accurately track people's silhouettes.

Kinect camera utilizes three types of coordinate systems for joints: color, depth and camera. The colour coordinates represent the position of the joint on the colour image captured by the RGB camera. They are values in the range of 0-1, which are percentages of the RGB camera resolution. The resolution of Kinect's RGB camera is 1920x1080. For example, the pair of colour coordinate values (0.25, 0.6) means the position of the joint (1920x0.25, 1080x0.6) = (480,648) on the colour image. [36.]

Depth coordinates represent the position of the joint on the image created by depth sensor. They are also represented by pair of values 0-1, which are the percentages of the depth camera resolution. The resolution of the depth camera is 512x424. For instance, depth coordinates (0.5, 0.8) points to the joint's position (512x0.5, 424x0.8) = (256, 339.2) on the depth image. [36.]

The camera coordinates represent the positioning of the joints in the 3D space relative to the Kinect's infrared sensor. The origin is at the centre of the IR-sensor. The x axis is along the Kinect sensor and increases to the left, the y-axis points along the height of the sensor and increases upwards, while z-axis is perpendicular to the face of the sensor and increases outwards from its face. The position of the joint in the skeletal data is expressed as camera coordinates. [36.]

Each joint in the person's skeletal data was accessed and checked for the tracking state. If the joint was tracked, its camera coordinates were converted into color coordinates and the joint was drawn on the RGB image using the new coordinates. As a result, the whole skeleton was rendered on the RGB image.

Then, using the gesture database the gestures of lying and jumping were detected. Specifically, the discrete gestures of lying and jumping were detected by database's Adaptive Boosting models. Continuous jumping's progress was

determined by Random Forest Regression model. Based on the detected gestures, their duration and confidence value, the behaviours ‘needing assistance’ and ‘vandalizing’ were recognized. The recognition times were obtained by C++ time functions and were logged into the database along with behaviours’ names. This data would be later used by professionals to deal with emergency situations and improve user experience in an elevator.

Finally, the RGB image was displayed on the computer screen. The image showed the person standing in front of the Kinect camera with the overlayed tracked skeleton. The sequence of these images formed a video, which was used to interact with the prototype. The video also displayed names of the detected discrete and continuous gestures with their confidence or progress value.

The third part of the software runs a behaviour recognition function in a main program and checks for possible exceptions. If exception was detected, its name was displayed on the screen.

6.5 Software Testing

The behaviour recognition software was tested, and the recognized behaviours were saved in a database. The logged behaviours with their recognition times can be seen in Figure 28:

```
Needs assistance : 10:52:27
Vandalizing : 10:53:34
Vandalizing : 10:54:16
Vandalizing : 10:54:23
Needs assistance : 10:56:23
Vandalizing : 10:56:36
Vandalizing : 10:57:27
Needs assistance : 11:8:30
Needs assistance : 11:8:39
Vandalizing : 11:8:39
Needs assistance : 11:10:49
Vandalizing : 11:10:56
Needs assistance : 11:10:58
Vandalizing : 11:12:53
Vandalizing : 11:13:2
Needs assistance : 11:19:56
Vandalizing : 11:19:56
Needs assistance : 11:20:0
Vandalizing : 11:20:6
Needs assistance : 11:24:0
Vandalizing : 11:24:10
Needs assistance : 11:31:35
Vandalizing : 11:31:35
Needs assistance : 11:32:11
```

Figure 28. Behaviours logged in a database

Classification results from the database were evaluated using a confusion matrix. Confusion matrix gauges performance of a classification model by comparing the predicted values with actual values [37]. The confusion matrix for the prototype can be seen in the Table 3.

Table 3. Confusion matrix

	Needing assistance	Vandalizing
Needing assistance	9 (TP)	2 (FP)
Vandalizing	2 (FN)	11 (TN)

In Table 3, predictions made by a prototype are represented by rows of a matrix and actual values are represented by columns. Based on this data, the classification metrics of accuracy, precision and recall were calculated. They were used to assess the prediction accuracy for each class.

Notions of true positive (TP), false positive (FP), true negative (TN) and false negative (FN) are used in these metrics. True positive is a prediction that matches the actual value and is positive, true negative is a prediction that matches the actual negative value. False positive is a false prediction about a positive value, while false negative is a false prediction about a negative value. [37.] In the confusion matrix in Table 3, these notions were marked assuming that 'needing assistance' is a positive class.

.

Accuracy is a fraction of correct predictions to all predictions. It was calculated with formula (19)[37].

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (19)$$

Precision specifies how many of the correct positive predictions are made. It was quantified using formula (20)[37].

$$Precision = \frac{TP}{TP+FP} \quad (20)$$

Recall computes the number of positive predictions out of positive predictions, that could possibly be made. Its value is given by formula (21)[37].

$$Recall = \frac{TP}{TP+FN} \quad (21)$$

F1-score combines the features of both precision and recall. It was calculated according to the formula (22)[37]:

$$F1\ score = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} \quad (22)$$

The described metrics were calculated for each behaviour class, while considering that class as positive. The results were summarized in a Table 4.

Table 4. Results for behaviour classes

Class	Precision, %	Recall, %	F1 Score, %	Accuracy, %
Needing assistance	81.82	81.82	81.82	83.33
Vandalizing	84.62	84.62	84.62	
Macro average	83.22	83.22	83.22	

According to the results in Table 4, the prototype had a classification accuracy of 83.33%. This was an acceptable result considering that a small quantity of videos was used for training lying and jumping gestures. To further increase the accuracy, more videos with more instances of the gestures should be recorded and used for algorithm's training.

When evaluating prototype's recognition performance for each behaviour, recall was considered more important than precision, as false negatives were of more interest than false positives. Missing the emergency behaviours was more critical compared to false alarms.

Recall of the 'needing assistance' class was lower compared to 'vandalizing' meaning that it was less accurately classified. To solve this issue, more videos with instances of lying should be recorded and used for training the gesture. Also, recalls for both behaviours can be increased by reducing the number of false negatives. This can be achieved by including both lying and jumping gestures in the recording videos and tagging one gesture as happening and other as not happening. Thus, the classifier will be able to distinct the gestures better compared to when they were recorded separately.

6.6 Possible Placement of the Prototype in an Elevator

Studies were carried out to find the optimal position of the Kinect camera in an elevator. They also covered the cases for the specific family of KONE elevators.

Kinect's placement analysis for capturing standing gestures can be seen in Figure 29.

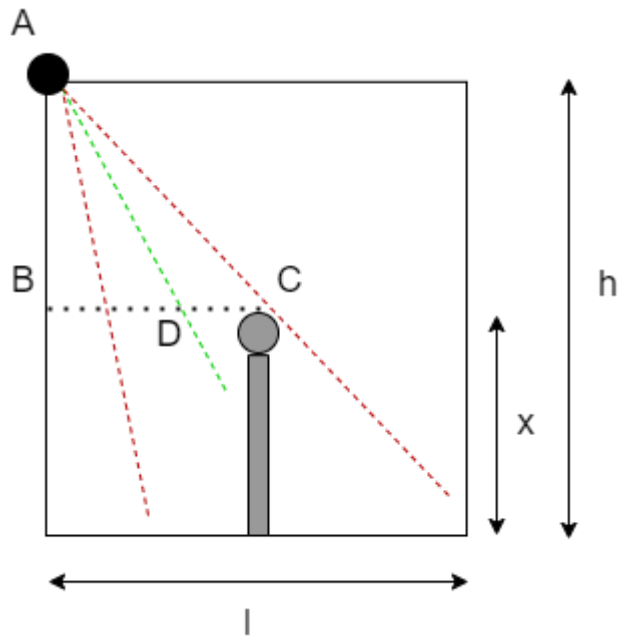

Figure 29. Prototype's placement analysis for standing gestures

Figure 29 shows a cross sectional view of an elevator cabin. Kinect camera is denoted as black dot and a person as a grey figure. Red checked lines represent the field of view of depth sensor and green line represents its direction. Here it was assumed that the person is standing in the center of the elevator.

A requirement for this situation was that field of view of the camera had to cover the top of the head and feet of the person. The problem was to find the angle $\angle BAD$ based on the known elevator dimensions.

Let l be the length of the elevator, and h be the height. The height of the person is assumed x meters to cover possible cases of height variations.

Then from the triangle ΔABC, $\angle BAC = \tan^{-1}\left(\frac{BC}{AB}\right) = \tan^{-1}(\frac{\frac{l}{2}}{h-x}) = \tan^{-1}(\frac{l}{2*(h-x)})$. $\angle DAC = 30°$ as it is a half of vertical field of view. Finally, the angle $\angle BAD$ is calculated according to formula (23):

$$\angle BAD = \tan^{-1}\left(\frac{l}{2*(h-x)}\right) - 30° \qquad (23)$$

The camera's placement analysis for capturing lying gestures is shown in Figure 30.

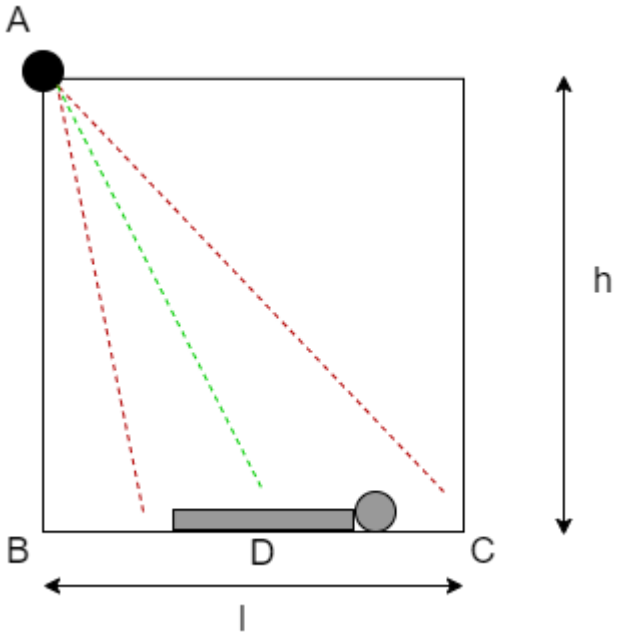

Figure 30. Prototype's placement analysis for lying gestures

In this situation, the main assumption was that the person is lying in the center of the elevator. It was required that the field of view of Kinect had to cover the whole body. The metric of interest was Kinect's vertical placement angle.

Let l be the length of the elevator and h be the height. From triangle ΔABD: $\angle BAD = \tan^{-1}\left(\frac{BD}{AB}\right) = \tan^{-1}\left(\frac{\frac{l}{2}}{h}\right)$. Hence, the angle $\angle BAD$ can be calculated with formula (24):

$$\angle BAD = \tan^{-1}\left(\frac{l}{2h}\right) \qquad (24)$$

The camera's angle formulas were applied to a family of KONE elevators to find the best placement in them. The results are not mentioned due to non-disclosure agreement.

7 Conclusion

The goal of the thesis project was to develop a prototype that could recognize human behaviours in an elevator. The prototype was specified to be able to recognize different behaviours, log their information in the database and have fast performance.

The thesis project was executed in four stages. During the first stage, the videos of the gestures were recorded and tagged. Based on those tagged videos, the gesture database was created and subsequently tested. In the second stage, the behaviour recognition software was written in C++. It provided the prototype's functionality of behaviour detection and database logging. In the third stage, results from the database were analyzed to assess the recognition algorithm's performance. Last stage covered the studies of positioning the prototype in the elevator for the best behaviour recognition performance.

The project's result was the prototype that could accurately predict different behaviours in an elevator. It achieved fast performance and could log information about recognized behaviours in a local database. This data would be later analyzed by professionals to handle emergencies and improve safety and riding experience in the elevator.

A future improvement for the prototype would be creating a more extensive database of different gestures with variations in viewpoint, lighting, and people. A sizeable and more varied gesture database would further increase the prototype's behaviour recognition rate and make it suitable for commercial purposes. A team of people would have to be involved in recording and tagging the gestures, as this process is excessively time-consuming for one person. In addition, a larger testing dataset would be created for algorithm's evaluation. This would provide a more accurate overview of the algorithm's performance and provide more insights into the sources of false negatives.

References

1 Statista [online] Number of IoT connected devices worldwide 2019-2030 URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ Accessed on 01.11.2021

2 VerywellMind [online] Understanding Body Language and Facial Expressions URL: https://www.verywellmind.com/understand-body-language-and-facial-expressions-4147228 Accessed on: 02.09.2021

3 Hussain Zawar, Sheng Quan, Zhang, Wei Emma. Different Approaches for Human Activity Recognition: A Survey. [online] URL: https://www.researchgate.net/publication/333745638_Different_Approaches_for_Human_Activity_Recognition_A_Survey Accessed on: 06.09.2021

4 NerdyElectronics [online] Selection Criteria for Sensors URL: https://nerdyelectronics.com/embedded-systems/sensors/selection-criteria-for-sensors/ Accessed on: 20.09.2021

5 Geeks for Geeks [online] Data Preprocessing in Data Mining URL: https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/ Accessed on: 06.09.2021

6 KDnuggets [online] An easy guide to choose the right Machine Learning algorithm URL: https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html Accessed on: 06.09.2021

7 C. A. Ronao and S.-B. Cho, “Human activity recognition with smartphone sensors using deep learning neural networks,” Expert Systems with Applications, vol. 59, pp. 235–244, 2016

8 Wikipedia [online] Activity recognition URL: https://en.wikipedia.org/wiki/Activity_recognition Accessed on: 06.09.2021

9 J. Han, H. Ding, C. Qian, W. Xi, Z. Wang, Z. Jiang, L. Shangguan, and J. Zhao, “Cbid: A customer behavior identification system using passive tags,” IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2885–2898, 2016.

10 Vrigkas Michalis, Nikou, Christophoros, Kakadiaris, Ioannis [online]. A Review of Human Activity Recognition Methods. URL: https://www.frontiersin.org/articles/10.3389/frobt.2015.00028/full Accessed on: 07.09.2021

11 Heng Wang, Alexander Kläser, Cordelia Schmid, Liu Cheng-Lin. Action Recognition by Dense Trajectories. CVPR 2011 - IEEE Conference on Computer Vision & Pattern Recognition, Jun 2011, Colorado Springs, United States. pp. 3169-3176, 10.1109/CVPR.2011.5995407 . inria-00583818

12 Microsoft Learn TV [online]Custom Gestures End to End with Kinect and Visual Gesture Builder URL: https://channel9.msdn.com/Blogs/k4wdev/Custom-Gestures-End-to-End-with-Kinect-and-Visual-Gesture-Builder Accessed on: 15.08.2021

13 IBM [online] Machine Learning URL: https://www.ibm.com/cloud/learn/machine-learning Accessed on: 21.10.2021

14 Towards Data Science [online] Decision Trees Explained URL: https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6 Accessed on: 04.09.2021

15 Scikit learn[online] Decision trees URL: https://scikit-learn.org/stable/modules/tree.html Accessed on: 14.10.2021

16 Learn Data Science [online] Gini Impurity URL: https://www.learndatasci.com/glossary/gini-impurity/ Accessed on: 14.10.2021

17 Section [online] Entropy and Information Gain to Build Decision Trees in Machine Learning URL: https://www.section.io/engineering-education/entropy-information-gain-machine-learning/ Accessed on: 14.10.2021

18 Analytics Vidhya [online] Regression Trees URL: https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047 Accessed on: 14.10.2021

19 Towards Data Science [online] Ensemble methods: bagging, boosting and stacking URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 Accessed on: 14.10.2021

20 Towards Data Science [online] Random Forest Explained URL: https://towardsdatascience.com/random-forest-explained-7eae084f3ebe Accessed on: 14.10.2021

21 ELPROCUS [online] What is an IR Sensor: Circuit Diagram & Its Working URL: https://www.elprocus.com/infrared-ir-sensor-circuit-and-working/ Accessed on: 04.09.2021

22 Cambridge in color [online] Digital camera sensors URL: https://www.cambridgeincolour.com/tutorials/camera-sensors.htm Accessed on: 04.09.2021

23 How Stuff Works [online] How Digital Cameras Work. URL: https://electronics.howstuffworks.com/cameras-photography/digital/digital-camera5.htm8 Accessed on: 04.09.2021

24 Intel REAL SENSE [online] Beginner's guide to depth URL: https://www.intelrealsense.com/beginners-guide-to-depth/ Accessed on: 06.09.2021

25 Analog Devices [online] Time of Flight Technology Overview URL: https://www.analog.com/en/applications/technology/3d-time-of-flight.html Accessed on: 06.09.2021

26 Microsoft| Azure Depth Platform [online] Understanding Indirect ToF Depth Sensing URL: https://devblogs.microsoft.com/azure-depth-platform/understanding-indirect-tof-depth-sensing/ Accessed on: 06.09.2021

27 Feature Fusion using Extended Jaccard Graph and Stochastic Gradient Descent for Robot - Scientific Figure on ResearchGate. [online] URL: https://www.researchgate.net/figure/The-structure-of-Kinect-V2_fig1_315655916 Accessed on: 06.09.2021

28 Real Time Apnoea Monitoring of Children Using the Microsoft Kinect Sensor: A Pilot Study - Scientific Figure on ResearchGate. URL: https://www.researchgate.net/figure/Comparative-specifications-of-Microsoft-Kinect-v1-and-v2_tbl1_313333776 Accessed on 06.09.2021

29 Microsoft|Docs [online] System requirements URL: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn782036(v=ieb.10)?redirectedfrom=MSDN Accessed on: 04.09.2021

30 Microsoft|Docs [online] Kinect Studio URL: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785306(v=ieb.10) Accessed on: 06.09.2021

31 Microsoft|Docs [online] Visual Gesture Builder: Overview URL: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785529(v=ieb.10) Accessed on: 06.09.2021

32 Yoshihisa Nitta: NtKinect - Kinect V2 C++ Programming with OpenCV on Windows10 [online] URL: http://nw.tsuda.ac.jp/lec/kinect2/ Accessed on: 06.09.2021

33 Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images [online] URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/BodyPartRecognition.pdf Accessed on: 04.09.2021

34 Microsoft| Developer Network [online] Skeleton detection in lying position URL: https://social.msdn.microsoft.com/Forums/en-US/4d5aa21a-e126-4ff9-90d8-7d051418b684/skeleton-detection-in-lying-position?forum=kinectv2sdk Accessed on: 15.09.2021

35 Understanding Kinect V2 Joints and Coordinate System – Figure on Medium [online] URL: https://lisajamhoury.medium.com/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16 Accessed on: 06.09.2021

36 Microsoft|Docs [online] Coordinate mapping URL: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785530(v=ieb.10) Accessed on: 06.09.2021

37 Analytics Vidhya [online] Everything you Should Know about Confusion Matrix for Machine Learning URL: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/ Accessed on: 14.10.2021